Towards Deployable Battery-Free Networked Systems

de Winkel, J.

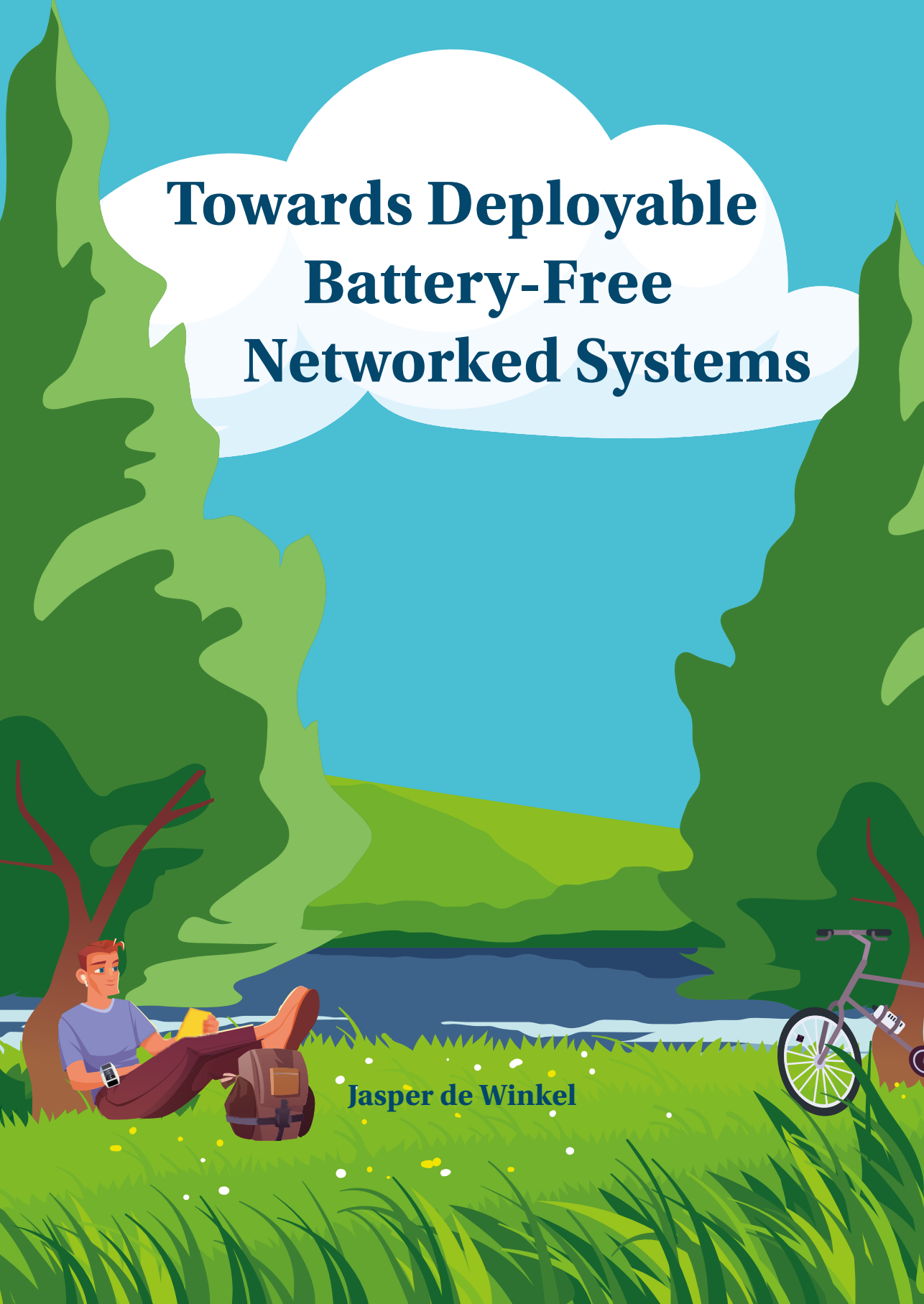
DOI
10.4233/uuid:77e3477b-b76b-429d-9dbd-f44cb457d2fa

Publication date
2023

Document Version
Final published version

Citation (APA)
de Winkel, J. (2023). *Towards Deployable Battery-Free Networked Systems*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:77e3477b-b76b-429d-9dbd-f44cb457d2fa

Towards Deployable Battery-Free Networked Systems



Jasper de Winkel

TOWARDS DEPLOYABLE BATTERY-FREE NETWORKED SYSTEMS

TOWARDS DEPLOYABLE BATTERY-FREE NETWORKED SYSTEMS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on Monday 3 July 2023 at 12:30 o'clock

by

Jasper DE WINKEL

Master of Science in Embedded Systems,
Delft University of Technology, the Netherlands,
born in Wageningen, the Netherlands.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	Chairman
Prof. dr. K.G. Langendoen	Delft University of Technology, promotor
Dr. P. Pawełczak	Delft University of Technology, promotor
Independent members:	
Prof. dr. ir. A. Bozzon	Delft University of Technology
Prof. dr. S. Gollakota	University of Washington
Prof. dr. D. Hughes	KU Leuven
Prof. dr. M. Zimmerling	University of Freiburg
Dr. J. Sorber	Clemson University
Reserve member:	
Prof. dr. ir. F.A. Kuipers	Delft University of Technology

This research was supported by the Netherlands Organisation for Scientific Research (NWO), partly funded by the Dutch Ministry of Economic Affairs, through TTW Perspective program ZERO (P15-06) within Project P1.

Keywords: Battery-Free, Intermittent Computing, Wireless Networking, Internet of Things, Embedded Systems

Printed by: Ipskamp Printing

Cover design: Ria von Hout

ISBN 978-94-6384-453-6

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

CONTENTS

SUMMARY

The ecological impact of today's battery-powered Internet of Things (IoT) is troubling. Technology advancements that reduce the reliance on batteries could blunt the environmental impact of the projected billions of IoT devices. With the emergence of low-cost, small, and high-performance microcontrollers, along with more efficient micro-energy harvesting devices that can harness the power of sunlight, motion, and heat a new revolution in computing has come. That is, IoT devices are increasingly leaving their batteries behind and are relying only on ambient power from sunlight, motion, thermal gradients, and other modalities to power their operation. Unfortunately, harvested energy can fluctuate greatly and is hard to predict, leading to intermittent operation. Intermittently-powered devices form a new class of low-power devices that can guarantee correct and forward-progressing computation despite these frequent power interrupts.

Despite the inconvenience of intermittent operation, the benefit of using intermittently-powered devices instead of 'classical' battery-based ones is threefold. The removal of batteries creates a more environmentally-friendly device, harvesting energy from ambient sources is sustainable and removing the battery can potentially lead towards perpetual operation—as long as there is an ambient energy source, battery-free devices will continue operating.

Challenges of battery-free devices however, still include basic features that are foundational to IoT devices. Interaction with battery-free devices has so far remained largely unexplored although reactive and screen-oriented systems are a significant part of today's and future Internet of Things. Common tools used during development, such as debuggers and testing frameworks, are practically non-existent for intermittent devices. Even basic concepts such as keeping track of time need to be carefully considered on intermittently-powered devices. Finally, wireless networking of intermittently-powered devices is severely limited to only backscatter or one directional communication.

This dissertation addresses the challenges mentioned above by developing and deploying mechanisms that enable connected and fully interactive applications on battery-free devices. These mechanisms alleviate key challenges that hinder actual adoption and infrastructure-less deployment of these battery-free devices.

To address the interactive battery-free device challenge, we present a framework based around energy-aware interactive computing using a reference implementation of a battery-free gaming console, the Game Boy. Using our energy harvesting mechanism, we not only harvest ambient solar energy but also energy from "button mashing" by the user to power its operation. With our from-the-ground-up energy-aware design, we implement an efficient state saving (checkpointing) mechanism, that stores only the modified memory regions since the last checkpoint. This mechanism is used to store the system state prior to and restore the system state after a power failure, allowing the system to continue operation after a power failure from the last successful checkpoint.

We address the debugging and testing challenge by the development of a new tool allowing for debugging of intermittently-powered systems just like any other embedded system. Key mechanisms allow the proposed debugger to automatically reconnect and restore debugging features such as breakpoints, enabling continuous debugging sessions despite intermittency. Using automated testing we found intermittency-related bugs in state-of-the-art battery-free systems, emphasizing the need for adequate testing tools for intermittently-powered devices and their unique bugs.

Next, we explore solutions to keep track of time on intermittently-powered devices in order to address this fundamental requirement for networking. We propose a new flexible and robust timekeeping mechanism, featuring an array of cascaded RC circuits. Low start-up time, high resolution and run-time reconfigurability are the key features of our timekeeping mechanism. As a demonstration we show communication between two intermittently-powered devices using our new timekeeping mechanism.

To enable infrastructure-less wireless networking on battery-free devices, we have proposed a generic battery-free connected device architecture and introduced a new kind of checkpointing mechanism. The architecture and checkpointing mechanism allow the microcontroller to turn off when idle and seamlessly resume any connections when restored, despite intermittent power. The decision on when to switch off, checkpoint and restore is handled by extending the real-time operating system's scheduler. Combined with per-process checkpoints, it enables efficient intermittent operation with pre-existing network stacks with minimal modification. Our architecture and mechanisms allow for fully featured bi-directional communication, as demonstrated with Bluetooth Low Energy (BLE), and lowers the idle power consumption. We design a smartwatch based on our new architecture that connects using BLE with a smartphone, allowing it to tell time and show email notifications.

The mechanisms developed in this dissertation enable a wide span of connected and interactive applications. We demonstrate the capability of our mechanisms in several applications such as battery-free speed monitoring of a bike, battery-free interactive gaming consoles, and even a battery-free smartwatch with BLE. These demonstrations highlight that even for connected and interactive applications, battery-free devices form a viable alternative to their battery-powered counterparts. By enabling viable battery-free alternatives, we hope to change the reliance on batteries for the future billions of IoT devices, mitigating their troubling ecological impact.

SAMENVATTING

De ecologische gevolgen van de huidige op batterijen gebaseerde Internet of Things (IoT) apparaten zijn verontrustend. Door de afhankelijkheid van batterijen te verkleinen, kunnen technologische ontwikkelingen de ecologische gevolgen van de voorspelde miljarden IoT apparaten verkleinen. Met de komst van goedkope, kleine en krachtige microcontrollers, gecombineerd met steeds kleinere en efficiëntere manieren om energie op te wekken uit bronnen zoals zonlicht, beweging en hitte is een revolutie begonnen waarin IoT apparaten hun batterijen achter zich laten. Deze apparaten wekken zelf alle energie op om te functioneren vanuit de omgeving, uit bronnen zoals zonlicht, beweging en thermiek. Helaas fluctueert de hoeveelheid energie die uit deze bronnen gewonnen kan worden sterk en is deze ook moeilijk te voorspellen. Dit leidt tot korte stroomonderbrekingen. Apparaten die om kunnen gaan met deze stroomonderbrekingen, ook wel bekend als 'intermittent' apparaten, vormen een nieuwe groep van energiezuinige apparaten, zij garanderen nauwkeurigheid en continuïteit in hun berekeningen ondanks deze frequente stroomonderbrekingen.

Ondanks deze problematiek is het voordeel van het gebruik van batterijvrije, in tegenstelling tot 'klassieke' op batterij-gebaseerde apparaten, drievoudig. Het verwijderen van de batterij resulteert in een milieubewuster apparaat, het opwekken van energie uit de omgeving is duurzaam en het verwijderen van batterijen kan leiden tot een vrijwel oneindige levensduur van het apparaat. Immers, zolang er energie opgewekt kan worden uit de omgeving, kunnen deze apparaten blijven functioneren.

Echter vormen fundamentele eigenschappen van IoT apparaten nog uitdagingen voor batterijvrije apparaten. Ondanks dat reactieve en schermgebaseerde systemen een significant onderdeel vormen van de huidige en toekomstige Internet of Things, zijn interactieve batterijvrije apparaten nog nauwelijks onderzocht. Gebruikelijke instrumenten zoals debuggers en testoplossingen, zijn praktisch niet beschikbaar voor intermittent apparaten. Zelfs basale concepten zoals de manier waarop tijd bijgehouden wordt moeten zorgvuldig overwogen worden bij intermittent apparaten. Tot slot, draadloze communicatie met intermittent apparaten is nog gelimiteerd tot alleen het reflecteren van radiogolven of tot eenrichtingscommunicatie.

Dit proefschrift draagt oplossingen aan voor de bovengenoemde uitdagingen door de ontwikkeling en inzet van mechanismes die verbonden en volledig interactieve applicaties mogelijk maken op batterijvrije apparaten. Deze mechanismes adresseren de grote uitdagingen die de integratie en infrastructuurvrije inzet van batterijvrije apparaten momenteel verhinderden.

Om de uitdagingen omtrent interactieve batterijvrije apparaten te adresseren, presenteren wij een raamwerk van oplossingen gebaseerd op een energiebewuste, interactieve referentieimplementatie van een batterijvrije spelcomputer, de Game Boy. Ons mechanisme wekt energie op uit zowel zonne-energie als de interactie met de gebruiker om het

systeem van stroom te voorzien. We ontwerpen en implementeren een efficiënt mechanisme om de voortgang van het systeem op te slaan (ook wel bekend als een checkpoint mechanisme), waarin alleen de veranderde geheugenregio's opgeslagen worden sinds het laatste checkpoint. Het mechanisme wordt gebruikt om de voortgang van het systeem op te slaan voor en te herstellen na een stroomonderbreking. Dit stelt het systeem in staat om na een stroomonderbreking vanaf het laatste checkpoint verder te werken.

We adresseren de debugging en testuitdaging door een nieuw instrument te ontwikkelen die het mogelijk maakt om intermittent apparaten te debuggen, op dezelfde manier als ieder ander embedded systeem. Unieke mechanismes maken ononderbroken debuggingsessies mogelijk ondanks stroomonderbrekingen, doordat de debugger automatisch opnieuw verbindt en debuggingeigenschappen zoals breakpoints hersteld. Met geautomatiseerde testen hebben we aan stroomonderbrekingen gerelateerde bugs gevonden in recente batterijvrije systemen. Dit benadrukt de noodzaak voor adequate testinstrumentatie voor batterijvrije systemen en hun specifieke problemen.

Vervolgens exploreren we oplossingen om tijd bij te houden op intermittent apparaten om deze fundamentele eis voor communicatienetwerken te adresseren. We stellen een nieuw flexibel en robuust mechanisme voor om tijd bij te houden door middel van een cascade van RC schakelingen. Snelle opstarttijd, hoge resolutie en herconfigureerbaarheid zijn de centrale eigenschappen van ons timing-mechanisme. Als demonstratie tonen wij aan dat met ons mechanisme het mogelijk is om twee intermittent systemen met elkaar te laten communiceren ondanks frequente stroomonderbrekingen.

Om infrastructuurvrije draadloze netwerken mogelijk te maken op batterijvrije apparaten hebben wij een generieke architectuur voorgesteld samen met een nieuw soort checkpointing mechanisme. De architectuur en het checkpointing mechanisme stellen de microcontroller in staat om zichzelf uit te zetten als hij inactief is, en vervolgens naadloos communicatieverbindingen voort te zetten als hij opnieuw opstart, zelfs onder frequente stroomonderbrekingen. De beslissing wanneer uit te gaan, een checkpoint te maken en de staat te herstellen wordt genomen door een uitbreiding van de scheduler van het realtime besturingssysteem. In combinatie met een per-proces checkpointing aanpak maakt dit het mogelijk om, met minimale wijzigingen aan reeds ontwikkelde communicatiesoftware, efficiënt om te gaan met stroomonderbrekingen. Onze architectuur en mechanismes maken volledige bidirectionele communicatie mogelijk, zoals gedemonstreerd met Bluetooth Low Energy (BLE), en reduceert het stroomverbruik als het systeem inactief is. We ontwerpen een smartwatch op basis van onze nieuwe architectuur die door middel van BLE een verbinding opzet met een smartphone, waarover de tijd en email-notificaties gedeeld worden.

De in dit proefschrift ontwikkelde mechanismes maken een brede span van verbonden en interactieve applicaties mogelijk. Wij demonstreren deze mogelijkheden door een batterijvrije snelheidsmeter voor een fiets, batterijvrije interactieve spelcomputers en zelfs een batterijvrije BLE smartwatch te realiseren. Deze demonstraties onderstrepen dat batterijvrije apparaten zelfs voor verbonden en interactieve applicaties een gedegen alternatief vormen voor hun batterij-gebaseerde concurrenten. Door batterijvrije alternatieven mogelijk te maken, hopen wij de afhankelijkheid van batterijen voor de toekomstige miljarden Internet of Things apparaten te verkleinen en daarmee hun ecologische voetafdruk op de wereld te reduceren.

CHAPTER 1
INTRODUCTION

Figure 1.1: Examples of IoT devices, ranging from smart thermostats and switches to smartwatches. All currently still powered by batteries.

Ever-increasing numbers of Internet of Things (IoT) devices surround us in daily life [80, 123, 235, 14]. Examples of IoT devices (Figure 1.1) range from industrial sensors monitoring an assembly process [27], smartwatches capable of monitoring your exercise [12], to intelligent thermostats [106]. As technology progresses, more computation power becomes available at low power consumption hence, more and more applications will become viable. However, IoT devices come at a cost to society as most IoT devices are currently powered by batteries. These batteries often need to be regularly replaced, monitored, and recycled [81]—inducing a potential environmental impact to our planet and monetary cost to the consumer [78, 110]. While the search for a battery that is longer-operational [81], better recyclable [203] and having high-energy density [274] continues, the road leading to such batteries is still long [328, 35].

Current batteries used in IoT devices have a finite lifetime, usually limited by the number of recharge cycles. Often the battery is the limiting part of an IoT device hindering longer lifetimes or even perpetual operation [270]. Even when every battery is fully recycled, batteries still pose a massive challenge to the environment as more and more batteries are required to power the ever-increasing number of IoT devices.

Battery-free devices offer a more sustainable alternative by getting rid of the battery entirely, solely operating on harvested energy from the (ambient) environment and store this energy in small (super)capacitors [115]. Energy can be harvested from a variety of sources including: solar energy (often harvested using solar panels), kinetic energy (can be harvested from piezoelectric elements or the simple movement of a magnet in a coil), energy from Radio Frequency (RF) waves (using an antenna) and thermal energy (with a Peltier element).

Unfortunately, harvested energy might not be consistent as even a simple device such as a room occupancy sensor powered by harvested solar energy will abruptly harvest more/less energy when the lights are turned on/off in the room. In-road sensors might be covered by a car, covering a solar panel. A person might block a RF antenna by walking in front of it. Together with the limited storage capacity of (super)capacitors, this leads to intermittent operation where at a certain point, not enough energy is present to continue operation and the device ceases operation until enough energy is harvested to resume. For battery-free devices powered by ambient energy on/off times can range

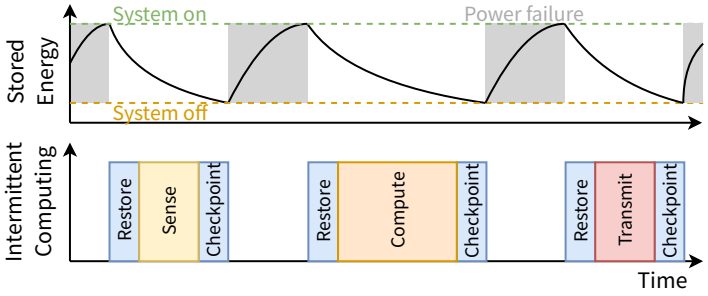


Figure 1.2: Schematic representation of intermittent operation: periods of system 'on' state are intervened by periods of 'power failure' state (when the system's capacitor is being re-charged). The power intermittency is caused by the unpredictable nature of ambient harvested energy. Therefore, intermittently-powered devices need to checkpoint and restore the intermediate state to and from non-volatile memory to guarantee forward progress despite power interrupts.

from microseconds [246, Figure 1] to seconds [159, Figure 2(a)] for RF- and solar-powered devices, respectively. Graceful handling of intermittency is an essential part of developing battery-free/intermittent systems.

1.1. SAVING THE STATE OF BATTERY-FREE SYSTEMS

Researchers have addressed the *intermittent computing* challenge by designing methods to save the system's state upon an imminent power failure. The application developer must programmatically account for two events. That is, whenever a power interrupt happened, at any place in the code, the device (i) must resume operation from the moment that power interrupt happened, and (ii) the state of the device's memory and its peripherals must be correctly restored. Saving of the system state is achieved by storing volatile system elements such as volatile memory, peripheral and processor state to non-volatile memory. When enough energy has been harvested to resume operation, the system state is then restored from the copy in non-volatile memory, allowing the device to continue operation from where it left off, as shown in Figure 1.2. To assure that both events (i) and (ii) happen the programmer can instrument the code by means of two common approaches.

A first approach of storing the system state is based on checkpoints that are inserted into the system's code. At a checkpoint the volatile system state is stored to non-volatile memory and operation resumes from the point of the last checkpoint if a power failure occurs. The restoration process involves restoring the previously stored state in non-volatile memory to the volatile system components. These checkpoints can be inserted manually [66], triggered periodically [164] or placed automatically at compile-time [28, 190, 306]. Commonly checkpoints are double buffered in non-volatile memory, ensuring that even when the system experiences a power failure during checkpointing the system can recover from the last checkpoint. Just-In-Time (JIT) based checkpointing systems [191, 246, 5, 145, 315, 28] only start checkpointing when the system's energy storage is low, avoiding unnecessary checkpoints. The low energy threshold determining

when the system starts checkpointing requires careful configuration. If set too low, the system might not be able to complete a checkpoint, losing any progress made.

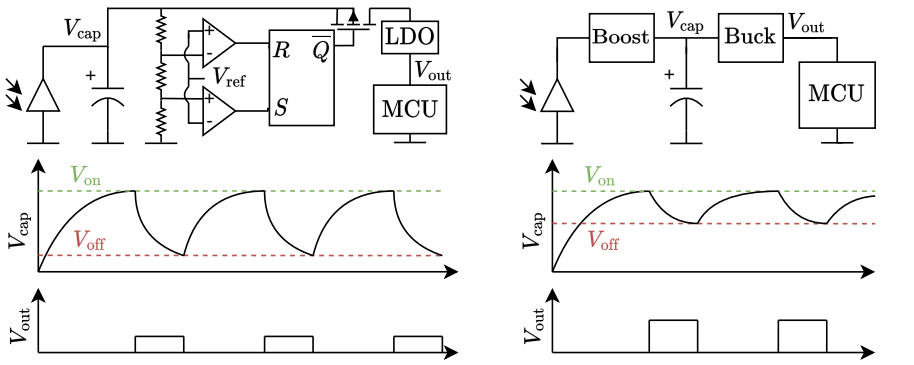
Alternatively, a second approach requires the developer to split the program into atomic blocks called tasks [185, 50, 116, 325, 251, 47, 193] or threads [326]. When considering task based approaches it is important to note that these tasks differ from traditional tasks in embedded systems. Tasks usually are responsible of handling processing of a certain device function. In intermittent systems, tasks are defined as units of code with clearly defined input and output whose execution time matches the specific energy budget of the intermittently-powered device. The energy budget of an intermittently-powered device can be determined by the size of its energy storage and can vary with the amount of harvested energy. The minimum execution time is closely related to the energy budget, as this is the time from the system switching-on until switching-off without harvesting any additional energy. If a task exceeds the minimum execution time based on the device's energy storage, an infinite loop is possible where the task might not be able to complete and no forward progress is made. The same applies for a checkpointing system when checkpoints are placed too far apart.

1.2. DESIGN OF BATTERY-FREE SYSTEMS

As IoT devices span a wide range of applications, from interactive smartwatches and thermostats to passive sensors monitoring temperature or observing industrial motors for faults, implementations vary. However, most IoT devices sense, process the sensed data, and then transmit their results either directly to the cloud or to an intermediate device for further processing such as a smartphone. Despite the available strategies to programmatically save the device's state and resume, on a systems level battery-free devices are fundamentally different. This is due to the limited available energy, limited energy storage and the addition of energy harvesting circuitry when compared to battery-powered IoT devices. Typical battery-free systems consist of five main components. That is (i) an energy harvester, (ii) storage in the form of (super)capacitors, (iii) a Microcontroller Unit (MCU), (iv) sensors and (v) a radio. Not only are there different approaches for harvesting and storing energy on battery-free devices but also communication has to be carefully considered as radio's are one of the most energy consuming elements of any IoT device.

1.2.1. ENERGY HARVESTING AND STORAGE

Simple hardware architectures [64, 114] directly charge the capacitor from the energy harvesting source, such as a solar panel, as shown in Figure 1.3a. Using a set of comparators the power form the capacitor to the regulator powering the MCU is enabled/disabled with hysteresis according to an upper turn-on (V_{on}) and low-cutoff (V_{off}) threshold. The circuit prevents the MCU from operating on too low voltage and ensures a minimal on-time for the system. A more complex arrangement [66, 68] features an energy harvester chip (commonly a boost converter) to more efficiently harvest energy from the ambient source. The boost converter then charges the (super)capacitor. With the addition of a buck converter that is enabled and disabled according to the turn-on and off thresholds, a stable but intermittent power supply to the system is generated from the boosted input, as shown in Figure 1.3b.



(a) Simple architecture, power to the MCU is enabled and disabled using two comparators.

(b) Boost and buck converter to generating a stable but intermittent MCU power source.

Figure 1.3: Common energy harvesting architectures powering battery-free systems.

Dynamic adaptation of (super)capacitor configuration and/or size can result in faster turn on times, by only enabling a small capacitor when little energy is available and sizing up when more energy becomes available [51, 321]. As a alternative to a central energy storage, each system element such as a sensor or radio can have its own dedicated energy storage, providing additional flexibility [114].

1.2.2. NETWORKING

For any IoT device networking is crucial as without its ability to communicate, no actions can be taken on the observations of the device. Imagine an occupancy sensor that is unable to communicate occupancy to the lights or the controller. Communication is required in both directions, from the device to send its findings or results to actuate upon and to the device for configuration and firmware updates. As even the best designed and evaluated IoT devices inevitably contain bugs in their code.

Apart from the differences in powering battery-free systems, wireless networking on battery-free devices is severely restricted by power consumption and energy storage. Continuous communication might require more power than can be harvested from the ambient environment and there might not be enough energy to complete a full transmission. Over the years many wireless technologies have been developed that have been widely adopted, with the most ubiquitous local area network being WiFi. Wide area networks such as NB-IoT and LoRa allow for nationwide coverage. Personal area networks such as Bluetooth and Zigbee offer connectivity for a range of applications such as connecting wireless headsets, monitoring industrial motors and asset tracking. Devices communicating using the previously mentioned networks use radios that are capable of generating their own (carrier) signals and commonly alternate between receiving and transmitting. In this dissertation, we refer to these radios as active radios.

Alternatively, passive communication as used in Radio Frequency Identification (RFID) can be achieved by selectively reflecting an incoming (carrier) signal, a method

commonly referred to as backscatter. This simple and passive method of communication requires little energy as it can be implemented with a simple switch, switching between reflecting or absorbing the signal. From this same carrier signal, energy can be harvested to power the device itself. The carrier signal itself however, is usually transmitted by a reader at a radiated power of 1 W or 2 W, imposing power infrastructure requirements for the reader/signal generator. The attenuation of a wireless signal as it propagates through space between the transmitter and receiver known as path loss forms a fundamental limitation for RFID. As the carrier signal is reflected, the signal is attenuated on both the path to the device and on the reflected path back, limiting its range. Active radio's in contrast only suffer this loss once and operate without the need for external infrastructure to supply a carrier signal.

So far several battery-free backscatter platforms have been developed such as [238, 207], even bi-directional communication between devices [194] has been demonstrated but suffer from limited range and infrastructure requirements. For battery-free devices many networking technologies such as WiFi currently remain out of reach without the use of backscatter due to their power consumption. The use of a large energy storage and a long period of energy harvesting could allow for transmission of a few packets using LoRa as demonstrated in [230]. However, out of all the widely adopted non-backscatter networking technologies, Bluetooth Low Energy (BLE) has one of the lowest power consumption, making it a feasible option for battery-free devices. BLE allows for end devices to transmit beacon packets to a usually continuously receiving device as a form of communication without establishing a connection. This one-directional connection-less beacon packet is usually sent from a single capacitor charge of harvested energy as been demonstrated on various platforms [114, 39, 101, 90, 258]. However, this leaves a gap in the state of the art as intermittently-powered systems still lack the ability to communicate bi-directionally without backscatter through widely adapted networks such as BLE.

1.2.3. APPLICATIONS

Despite the differences in powering, networking and maintaining state, several applications of battery-free devices have been demonstrated. These applications include, sensing at Circus Maximus [3], occupancy detection [254], video streaming [253], cycle computer for bikes [268] and gesture recognition [301]. However, the full proliferation of battery-free devices is still held back. Most of these applications focus on sensing and many potential user-facing IoT applications remain largely unexplored such as interactive devices despite that they form a significant portion of today's IoT devices. Another common feature of these applications is that they do not seamlessly integrate into pre-existing common wireless networks but require specific (custom) infrastructure or lack bi-directional networking capability.

When considering adoption of battery-free devices into applications, the development of battery-free devices itself has to be considered. For replaying pre-recorded energy harvesting conditions to the intermittent device several testbeds and tools have been developed such as [112, 1, 96]. Nevertheless, actual debugging and testing of intermittent devices is still challenging due to the lack of tools such as debuggers supporting intermittent devices.

1.3. PROBLEM STATEMENT

With the current state of battery-free devices, development and deployment of real-world battery-free devices is still challenging as several basic features of their battery-powered counterparts are still missing or lacking. This dissertation aims to address this gap in research and thus centers around the following question:

> What mechanisms must be developed and deployed in battery-free networked systems to enable connected and interactive IoT applications?

In the following section we break down this overarching question into individual challenges that we address in this dissertation.

1.4. CHALLENGES AND OUTLINE

Before battery-free devices can be deployed in practice several challenges still need to be addressed. We outline them below.

► **Interactive Devices.** Prototype battery-free devices have been used to make phone calls [288], deployed for machine learning [169], greenhouse monitoring [113], video streaming [206], eye tracking [173], payment using smart cards [182] and even built into a robot [325]. Although some of these devices present *some* form of interactivity, these techniques or prototypes have not yet enabled fully *interactive battery-free devices*—like a smartwatch, in-place interactive display or even a **handheld video game console**. This is a critical gap in the research around battery-free devices, as these types of *reactive, interactive, and screen-focused* systems are a significant portion of the current and anticipated smart systems. In Chapter 2 we focus specifically on an ignored part of the battery-free device ecosystem, mobile gaming, and use this application to elucidate the essential challenges that must be explored for a future where reactive and user-facing applications can also be battery-free. As a proof by demonstration that battery-free interactive systems are possible, we design a battery-free gaming console named ENGAGE. To power this system we leverage a multi-input energy harvesting mechanism, harvesting both ambient solar energy and energy generated by user interaction with the console itself.

► **Debugging and Testing.** During the implementation of ENGAGE we experienced the difficulty of developing software for battery-free systems. This difficulty is not only due to the lack of adequate tools that allow for debugging and repeatable testing of battery-free intermittently-powered devices, but also due to the fact that not only traditional common embedded systems bugs have to be resolved but also the bugs related to intermittently-powered operation. Common examples of these intermittency-related bugs include checkpoint or restoration bugs where the state is not correctly saved or restored from a checkpoint. Where not only memory consistency has to be verified but also the state of the peripherals. With Just-In-Time based checkpointing systems for example, bugs can even be linked to a specific power trace to the system. Without easy-to-use debugging and testing tools for intermittent systems, market adaptation could suffer. In Chapter 3, we develop a tool named DIPS that enables debugging of intermittent devices just like any normal embedded system. It integrates both a hardware debugger and an energy emulator capable of replaying power traces to the device under test. Unlike traditional

debuggers it features mechanisms to automatically reconnect and restore debugging settings such as breakpoints to intermittently-powered devices when they resume, allowing for a continuous debugging session despite intermittency. Using automated tests, DIPS can find common intermittency-related bugs automatically. These tests are not only able to verify memory consistency after restores but can also verify peripheral state using the integrated hardware debugger.

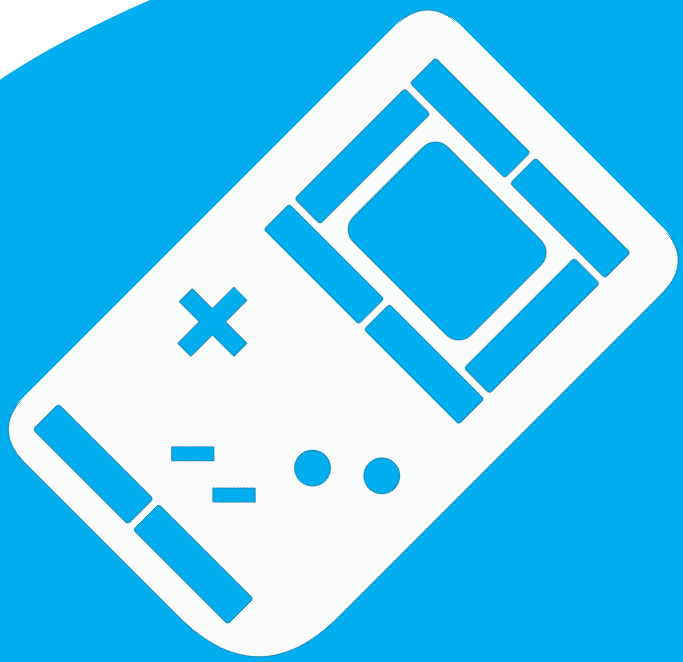
▶ **Keeping Track of Time.** Hardware and platform approaches have focused on reducing the cost of checkpointing [119], managing energy more efficiently to reduce power failures and increase event detection [113, 114, 51], and getting a rough estimation of time elapsed between power failures [245, 117]. However, deploying infrastructure-less intermittently-powered devices still remains challenging due to the lack of a fundamental feature, that is *robust timekeeping*. When operating on intermittent power, solely relying on MCU peripherals, such as timers, is impossible as they are unavailable during power failures. The ability to keep track of time undergirds a multitude of computing and networking primitives such as synchronization and networking, data collection, and real-time operation. One missing feature from our ENGAGE system was its lack of connectivity. In order to enable the use of time-driven features such as the use of a scheduler and Time Division Multiple Access (TDMA) based networking protocols like BLE, we need an accurate reference of time on intermittent devices. In Chapter 4, we explore the pros and cons of different timekeeping solutions and introduce a new ultra-low power flexible timekeeping mechanism.

▶ **Wireless Networking.** Widespread adaptation of battery-free devices is still hindered by the final challenge addressed in this dissertation, *wireless networking*. Although backscatter communication has been demonstrated on intermittently-powered devices, backscatter is fundamentally challenged by limited range and the infrastructure requirement of the external carrier generator. In our vision battery-free devices should replace their battery-powered counterparts without imposing any extra infrastructure requirements. They should integrate seamlessly in common widely adopted wireless networking standards and offer bi-directional communication. In Chapter 5 we explore the design of a new system architecture that allows for maintaining bi-directional communication under intermittent power. Central to this architecture is a new checkpointing mechanism that allows for simple integration of pre-existing non-intermittent software such as network stacks in intermittently-powered applications. As a proof of concept we use BLE and demonstrate the first BLE connection on intermittent power. In addition, we demonstrate the capabilities of our architecture and mechanisms by designing a smartwatch, powered solely by ambient solar energy. The watch connects to a smartphone using BLE and is able to tell time and shows email notifications.

The challenges mentioned above are by no means exhaustive. However we identify these challenges as crucial as they hinder practical widespread deployment of battery-free systems. Additional research challenges such as improving energy harvesting efficiency and further development of ultra-low power MCUs and radios, are not considered within the scope of this dissertation.

CHAPTER 2
BATTERY-FREE INTERACTIVE DEVICES

This chapter is based on:
Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak (2020).
Battery-Free Game Boy
Proceedings of the ACM on Interactive, Mobile,
Wearable and Ubiquitous Technologies 4, 3 (September 2020), 111:1–111:34.



2.1. INTRODUCTION

In this chapter, we focus specifically on an ignored part of the battery-free device ecosystem, *mobile gaming*, and use this application to elucidate the essential challenges that must be explored to enable a future where reactive and user facing applications can also be battery-free. From a market perspective there is deep need to explore this area. The global gaming industry is massive and generates unprecedented revenues, which already exceeded 100 billion USD in 2016 [209]. Handheld console game sales constitute a large portion of the industry [209]. In terms of units sold, as of September 30, 2019 Nintendo sold 41.67 million units of its latest Switch console, since its release in March 3, 2019 [214] and these numbers continued to grow rapidly during the worldwide COVID-19 lockdown [223]. As a comparison, Nintendo's Game Boy handheld, shipped 118.69 million units since its official release in April of 1989.

To enable these types of devices, mobile gaming platforms must be re-imagined at the system and interactivity level. The main challenge is that energy harvesting is dynamic and unpredictable. This is intuitively apparent when considering a solar panel; a cloud, the time of day, weather conditions, movement and orientation of the panel, even the electrical load all change the amount of harvested energy. Because of this dynamism and limited amount of energy, these devices lose power frequently, only computing *intermittently* with the device having to wait seconds or minutes to gain enough energy to turn back on. This long recovery process can be energy and resource intensive, causing responsiveness delays. Worse, it can leave the game in an inconsistent state. Naturally, going through the entire re-loading process (from loading screen of a game to starting play) every time is burdensome, so just blindly replacing batteries in a game console with an energy harvester is not enough to ensure smooth game operation.

To address this challenge, this chapter presents a *framework of solutions based around energy-aware interactive computing* and a *reference implementation of a popular game console*—8 bit Nintendo Game boy [215, 53]—as a demonstration, see Figure 2.1. To reduce the unpredictability of energy harvesting, we take advantage of mechanical energy generated by "button mashing" of the console, harvesting this energy generated by actually playing a game on a handheld, and using it, along with solar panels, to power all operation. We design the system hardware and software from the ground up to be energy-aware and reactive to changing energy situations to mitigate the issues caused by frequent power failures. Specifically we design a technique to create minimal *save games*

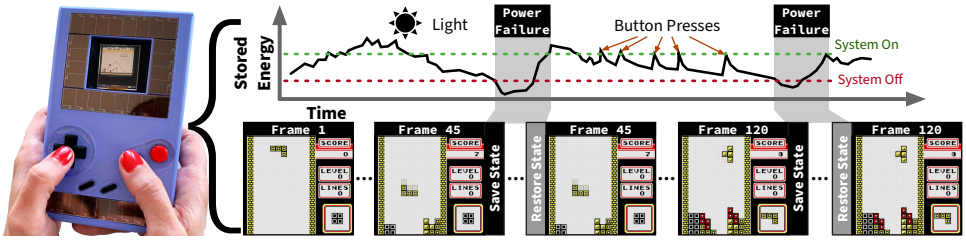


Figure 2.1: Energy harvested from button presses and sunlight powers our custom handheld platform, ENGAGE, running a Nintendo Game Boy emulator which can play classic 8 bit games. ENGAGE efficiently preserves game progress despite power failures, demonstrating for the first time battery-free mobile entertainment.

that can be quickly created, updated, and saved to non-volatile memory before a power failure, then quickly restored once power returns. Unlike save games seen in traditional gaming systems, these capture the entire state of the system, so the player can recover from the exact point of power loss; for example mid-jump in a platform game; all this despite the device fully losing power.

Contributions. In this chapter we present a practical, usable mobile gaming device, *Energy Aware Gaming* (abbreviated as *ENGAGE*). To date this is the first time full system emulation of a real world platform has been done battery-free, and the first intermittently powered interactive gaming platform. Our contributions follow:

1. We introduce the concept of intermittently powered, battery-free mobile gaming;
2. We develop an approach to failure resilient, memory-efficient, fast, whole system save games for interactive, display driven devices. A just-in-time differential checkpointing scheme is used based on the concept of tracking changed memory in *patches*;
3. We develop a hardware platform as a reference implementation with a novel multi-input architecture for harvesting energy from button presses and sunlight. This device also enables any interactive-based system (not necessarily a game);
4. As a stress test and demonstrative exercise of the promise of battery-free gaming, we use these systems and hardware to develop a full system Nintendo Game Boy emulator which plays unmodified Game Boy games despite power failures.

2.2. CHALLENGES

Personal, handheld gaming, has brought entertainment and fun to hundreds of millions of people in the past three decades. During the COVID-19 pandemic, when so many where in lockdown, gaming was one of the activities that reduced stress and boredom [260, 262, 290]. The goal of this work is to develop the systems and hardware foundations for battery-free mobile gaming. This is motivated by two reasons: (i) the enhanced availability and usability of a platform that never needs to be recharged or plugged in—making the platform more convenient for the average user, and more accessible for everyone, and (ii) the need for alternative and sustainable forms of entertainment—a nod to the various industry consortia such as *Playing for the Planet* [243] which aim to reduce the gaming industries ecological impact. A battery-free handheld game console reduces ecological costs and disappointment, as it is always ready to be picked up and played *without needing to be recharged.*

Numerous explorations of battery-free smart devices address the calls for sustainable/carbon-neutral electronic device interaction and electronic design and computing [162, 29, 195, 316, 157] while preparing human-interactive electronics for the "post-collapse society" [300]. Other work has developed core systems [115], hardware [51, 64], and programming languages [190, 325] for serious systems focusing on solving the *intermittent computing problem* caused by energy harvesting and battery-free operation, where frequent power failures prevent a program from finishing a task (see Figure 2.2). In all cases the electronic device is powered by harvested energy from the environment [242]

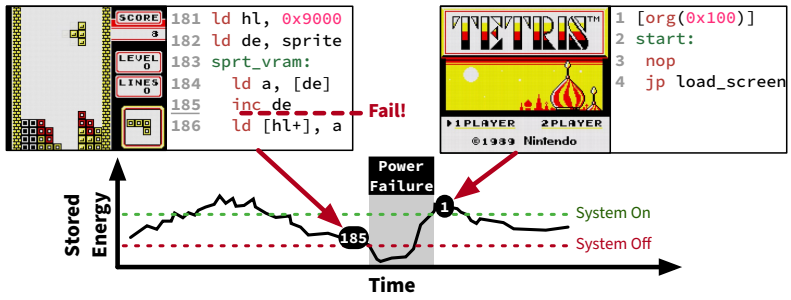


Figure 2.2: Dynamic energy harvesting causes voltage fluctuations which cause frequent power failures. Shown is what would typically happen if a battery was removed from a Game Boy and replaced with solar panels. The game would play until energy is lost (i.e. at line 185) and then restart at the loading screen. *Intermittent computing techniques* seek to make it such that after the power failure, line 186 is then executed proceeding from the exact system state as before the failure.

and stored in (super-)capacitors of much smaller energy density and size than batteries. None have yet explored the question of mobile handheld entertainment, going beyond the simple forms of battery-free gaming devices demonstrated commercially in early 1980's [58]. This is because making such a device is challenging due to complex system difficulties stemming from frequent power failures, listed below.

Challenge 1: Unpredictable Energy Harvesting. Environmental conditions change, this is exacerbated by mobile gaming. When players move from place to place, most forms of ambient energy change drastically (for instance, by moving from sun to shade), or increasing distance from a radio frequency power source. Without a more predictable source of power, it is hard to envision being able to play continuously without a battery.

Challenge 2: Keeping Track of System/Game State. Maintaining state of computation—let alone game state—through power failures from intermittent harvested energy is hard [184, 199]. Many software frameworks that support computation progress despite these power failures exist, saving state in non-volatile memory like FRAM and then restoring state after power resumes (see Figure 2.2), such as TICS [166], TotalRecall [315], and many others. Most systems trade memory efficiency for performance, this approach is opposite of that needed for gaming, where a display buffer and numerous sprites and large game state variables must be saved, requiring high memory efficiency.

Challenge 3: Enormous Variability of Games. These previous issues are compounded by the huge variability of games, both in terms of memory size, number of sprites, actions, difficulty, and even number of button presses per second. Each game is unique, and could pose difficulties when creating a general battery-free solution.

Challenge 4: Gaming's High Computational Load. To date, no full system emulation of any complex system has been attempted on battery-free, intermittently computing devices. Games and gaming platforms require more performant processors even when running natively—when running in emulation, this is compounded. All existing popular runtimes for intermittent computing are based on Texas Instrument's mixed-memory MSP430 MCU [296], which is order of magnitudes slower than the fastest ARM MCU on

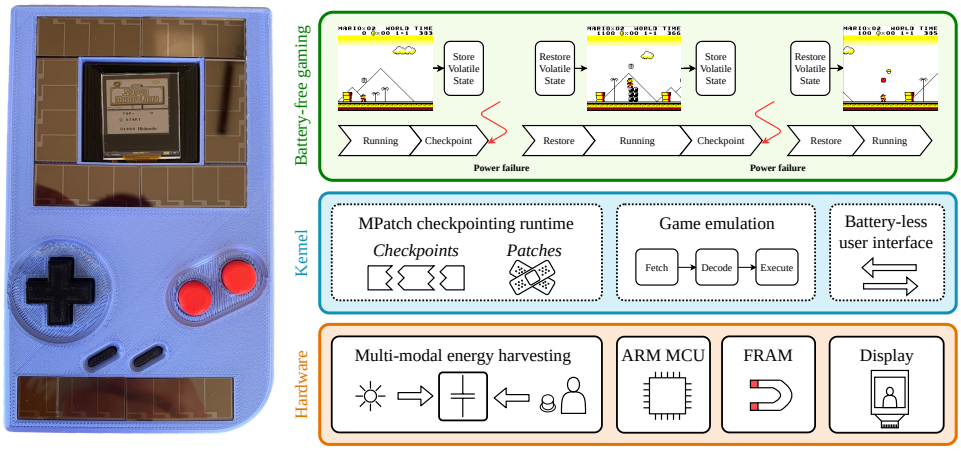


Figure 2.3: ENGAGE hardware platform (left) and its internal architecture (right).

the market. To meet the high computational load of games, a practical runtime for ARM microconrollers must first be built.

Challenge 5: Capturing User Actions. Playing a video game means a system needs to be highly reactive: button presses post immediate updates to a screen. But *none of the existing battery-free interactive devices demonstrated this level of interactive complexity* with continuously reacting buttons and instantly-refreshing screen. Only simple touch button interfaces that do not need constant pressing for interaction are demonstrated with electronic screens that usually present static content not informed by user actions. Guaranteeing reactivity with unpredictable energy is difficult.

Challenge 6: Realistic Demonstration. The over-arching goal is to play a real, unmodified, video game on a battery-free console that everyone around the world knows (like Tetris)—in other words to be able to execute preexisting game code (or any existing code for that matter), *not to design a custom game* only to demonstrate the potential of battery-free gaming (we refer to extensive survey on this topic in Section 2.7). This could be possible only when all above challenges are addressed.

To tackle above *Challenge 1–6* we took one of the most popular gaming consoles of all-time [214]—original 8 bit Nintendo Game Boy [215, 53]—and redesigned its hardware-software, powering gameplay from the solar panels and button presses of the user, building the first ARM based intermittent computing hadware and runtime system, and doing the first full system emulation of a real world platform (Nintendo Game Boy) with intermittent computing techniques.

2.3. BATTERY-FREE HANDHELD GAMING

We designed the *Energy Aware Gaming* (ENGAGE) platform as a proof by demonstration that the discussed challenges could be overcome. The design and architecture of the ENGAGE platforms is shown in Figure 2.3. The *ENGAGE hardware* is the size and form

factor of a Nintendo Game Boy, it is built around (i) user input via mechanical energy harvesting buttons (on the A, B, and D-Pad of the original Game Boy), (ii) a display, (iii) a slot for Game Boy game cartridges to be inserted, and (iv) energy harvesting circuitry from solar cells and the buttons which store energy in a small internal capacitor. The *ENGAGE kernel* consists of (i) a patch-based differential checkpointing system (denoted as MPatch) which handles low level memory movement and automatically saves and restores state of the entire system by efficiently moving necessary data to non-volatile memory (FRAM) and back (SRAM), and (ii) an extensively rewritten full-system Nintendo Game Boy emulator, which can run unmodified Game Boy games. ENGAGE is the first full system emulation, and the first gaming platform built for battery-free, energy harvesting, intermittently powered computing devices.

Usage and Impact. We intend to release the hardware designs, firmware and software as open-source, living repositories on Github [72]. We target a broad audience with our platform. Our battery-free platform (i) must be *open-source*—allowing hobbyists to expand it and improve it, including developing bespoke gaming libraries completely separate from the Game Boy emulator, (ii) have comparable *gameplay and feel* as the original Nintendo Game Boy, and (iii) be able to play any popular retro game.

2.3.1. KEY IDEAS

Existing handheld gaming devices rely on large batteries because they need continuous high power to support high compute load, energy cost, and reactivity. We want to enable playing retro 8 bit console games, such as *Tetris* and *Super Mario Land*, on a battery-free console that is similar in user interface and gameplay to the original Nintendo Game Boy. Removing the battery and only using harvested energy causes intermittent operation, which leads to the challenges discussed in Section 2.2. The ENGAGE platform design navigates these challenges based on four key ideas.

Gather Energy from Gaming Actions and the Environment. ENGAGE harvests energy from button pressing/mashing and solar panels facing towards the player. As opposed to other techniques that rely on nearby dedicated wireless power generation, this approach allows for truly mobile gaming; anywhere a player can find light to see the screen and press/mash buttons. By pairing energy generation with the game actions, and the natural environment where gaming happens, ENGAGE overcomes challenges stemming from unpredictability of energy harvesting, and also ensures that energy is more likely to be available when a user initiates an action (since actions generate energy). This *addresses* **Challenge 1** and **Challenge 5**.

Track and Checkpoint Minimal State at the System Level. We must handle intermittent power failures to maintain state of play. Unfortunately, in games large amounts of memory is moved back and forth to the display, often in the form of sprites, with computation happening in between. Naively checkpointing the entire system state would be impractical, significantly increasing latency of operation. We note that while large memory movements happen, the changes in these memories are often small, meaning we can reduce checkpoints to only the changed memory, save that state just in time before a power failure, and then restore that state and resume game play. This *addresses* **Challenge 2**, **Challenge 3** and **Challenge 4**.

Use Processor Emulation to Play Retro Games. While ENGAGE could be used for custom gaming libraries made specifically for intermittent operation, the more challenging and interesting problem is full system emulation enabling the play of the thousands of existing games, and even home-brewed games. This also allows us to explore and understand the variability of real world games. This *addresses* **Challenge 3** and **Challenge 6**.

Speedup Intermittent Computing. We embrace ultra low powered, high performance ARM Cortex microcontrollers, and external FRAM memory to speed up computation. While a seemingly trivial technology advancement, with this approach we increase compute speed but increase our I/O burden for checkpointing, as the traditional MSP430 FRAM-enabled MCUs have internal FRAM memory accessible at CPU speeds. This is a different tradeoff space than any other intermittent hardware platform [64, 114, 51]. This *addresses* **Challenge 2** and **Challenge 4**.

2.3.2. ENGAGE FULL SYSTEM NINTENDO GAME BOY EMULATOR

A key part of our approach is running a full system emulation on ENGAGE hardware. To be able to run Nintendo Game Boy games an emulator is used to emulate the instruction set of the Game Boy processor, i.e. 8 bit 4.19 MHz custom-built Sharp LR35902 MCU with a processor closely based on the Z80 instruction set [53]. An emulator reads bitcode instructions and executes them in native code, mimicking the emulated CPU as closely as possible to ensure it executes in an identical fashion to the emulated CPU. With the restrictions of battery-free systems additional scenarios are introduced that normally do not exist, such as the loss of power while running a game and then attempting to restore the system to the state it lost power. Additionally emulation efficiency is of critical importance in regards of power consumption.

The emulator allocates non-volatile and volatile Game Boy game memory within the memory space of ENGAGE, removing the need to keep cartridges continuously powered. Only upon loading a new game is the cartridge interface used to retrieve the non-volatile game data.

The process of emulating also requires emulation of the Game Boy I/O for the user to be able to interact with the device, most importantly the buttons and the screen. Changing behavior regarding interaction with the I/O might have an influence on the user experience and interaction with the device.

Emulating Button I/O. As energy can be harvested from the press of a button, the frequency of button presses determines the amount of energy generated. This button press frequency is game dependent: in-game-cut-scenes usually require no game-user interaction through the buttons compared to games like *Space Invaders* where buttons are continuously pressed. As a proof, in Table 2.1 we present statistics about the time between button presses in four popular Nintendo Game Boy games. The maximum time varies greatly depending on the presence of cut-scenes in the game. *Tetris* and *Space Invaders* have few, or short, cut scenes, and thus have a lower maximum time between presses and lower variance. On the other hand, *Super Mario Land* has an animation upon death and at the end of a level, and *Bomberman* has several cut-scenes, hence the higher maximum time between between presses and greater inter-press time variance.

This simple experiment shows that the maximum time between button presses di-

Table 2.1: Measurement statistics of all button pressing during a regular Game Boy game for four example games, showing variation between games depending on the type of game. Data was extracted by logging key presses during game play on a Game Boy emulator running on a stationary personal computer. Three similar three-minute playthroughs are averaged in the presented results.

Game name	Time between button presses (second)			Button presses per second
	Max	*Mean*	*Variance*	
Tetris	3.230	0.508	0.169	1.981
Space Invaders	3.542	0.372	0.129	2.715
Super Mario Land	12.46	0.652	1.091	1.543
Bomberman	7.534	0.765	0.762	1.313

rectly pertains to the required energy buffer size, where button mashing games could run on smaller energy buffers increasing the reactivity of the platform by reducing the required charge time.

By changing emulator behavior of handling buttons more energy can be generated. To generate more energy we prevented the holding of buttons. For example, in *Super Mario Land* game it is common to hold the right button to keep moving, but in *Space Invaders* this is less common. This results in a trade-off space between energy generation and changes to the user interaction with Game Boy[1]. Finding the optimum between energy harvesting without changing the user interaction is therefore game specific. In Section 2.4 we further elaborate on our design choices regarding button emulation.

Emulating Screen Writes. The original Nintendo Game Boy does not employ a frame buffer. Instead, it uses a tile-based approach where tiles are rendered in a CRT-like fashion on the screen to save memory. This means when power to the system is lost, the state of the screen can not be restored since knowledge of the full screen state is not maintained. To combat this we employ a frame buffer and map the Game Boy tile-based rendering into this frame buffer. The buffer is then checkpointed to be able to restore the state of the screen upon power failure.

2.3.3. GAMING THROUGH POWER FAILURES

ENGAGE is protected from the loss of progress by the custom-designed runtime that guarantees data consistency despite power interrupts. The goal of this runtime is to save (i.e. to checkpoint) the current state of the emulator. This entails the current volatile memory content and the registers of both the host processor and the emulated system. Doing this will allow the system to continue execution from this point as if a power failure never happened.

There are multiple intermittent runtime systems (all of them are summarized in Section 2.7) which can be broadly divided into two classes: (i) those that use a *special (C program) code instrumentation* to guarantee correctness of computation despite power interrupts and (ii) those that use a *special version of the checkpointing*, of which a subset

[1]We note that the original Game Boy handles I/O through polling registers, meaning that every game can have a different handling of the I/O all together, since the Game Boy directly executes machine code from the game.

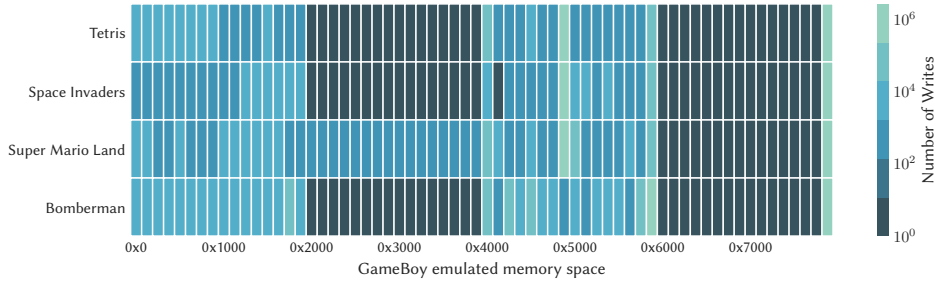


Figure 2.4: Memory writes heat map of four popular 8 bit Game Boy games for one minute of play. Writes tend to cluster in a few large regions; tracking and checkpointing these regions would allow for performant intermittent execution. Note the log-scale of the number of writes.

is designed for systems that use volatile memory—such as SRAM—as their main memory, and use a separate non-volatile memory that contains the checkpointed data. While designing ENGAGE we chose to use a checkpoint based system to allow emulation of arbitrary game code. We did not consider task-based runtimes simply because they are too complex to comprehend by a programmer and more difficult to design than a checkpoint-based system, see related discussion on this topic in [166]. But first and foremost, task-based system cannot execute a binary (machine) code, which ENGAGE is mostly executing.

The main requirement for ENGAGE is responsiveness, hence the checkpointing system needs to be as lightweight as possible. Naturally all of the checkpoint systems have some overhead, so when searching for a good solution we would like to minimize checkpoint size as much as possible—resulting in minimum overhead from data restoration. Checkpointing the entire system state, including game, and emulator, would be impossible. One core idea, proposed first by the DICE runtime [5], is *to checkpoint only parts of device memory that have been changed since the last checkpoint.* To check whether this idea applies to battery-free handheld gaming system we have performed a simple experiment. For four example Nintendo Game Boy games: (i) *Tetris*, (ii) *Space Invaders*, (iii) *Super Mario Land*, and (iii) *Bomberman*, we have measured to which memory regions of an MCU each game was writing to during one minute of game play. The result is presented in Figure 2.4. Indeed, we see that memory writes are very unevenly distributed for each game, hinting that such approach, which we broadly denote as *differential* checkpointing, is well suited for our ENGAGE needs.

The checkpoint runtimes, including differential ones, can be further divided into two unique classes: (i) *corruptible* and (ii) *incorruptible*.

- **Corruptible Checkpoint:** Such systems copy the current state of the MCU (memory, registers, etc.) to a predetermined location in non-volatile memory. This location is the same every time, as this eases the runtime development and reduces the non-volatile memory requirements. However, it is required that a checkpoint operation must guarantee to complete, otherwise part of the previous checkpoint may be overwritten with the current checkpoint[2]. Often these corruptible runtimes include

[2] If the system where to run out of power during the creation of a checkpoint, with the next checkpoint restoration a corrupt state will be restored leading to undefined behaviour—thus to a corrupt system.

a check whether a checkpoint was completed successfully, otherwise they start the program execution from the beginning. Such systems require exact prediction of the energy (required to perform a checkpoint) and the energy currently consumed by the complete system (to be able to guarantee that a checkpoint is only performed when its completion can be guaranteed). Such a requirement is *unrealistic* for a computing platform, such as ENGAGE, that includes many peripherals and components all connected to the same energy buffer, as correctly predicting the required energy—even the CPU alone—is difficult;

- **Incorruptible Checkpoint:** Such systems take a different approach: at all times they *guarantee* that there is a valid checkpoint which can be restored. This means that a new checkpoint will never overwrite part of the previous checkpoint in non-volatile memory. Such guarantee is often implemented through double-buffering.

As of now, there are *no known incorruptible differential checkpoint systems*, and just one corruptible differential checkpoint system, DICE [5], refer also to Table 2.2 where existing intermittent runtimes are qualitatively compared from ENGAGE requirements point of view. Therefore, to realize a working ENGAGE we developed a new checkpointing runtime, denoted as MPatch, that performs incorruptible differential checkpoints. The proposed runtime is aided by a new concept of *patch checkpointing*, discussed below.

MPatch—a Patch Checkpointing Intermittent Runtime. Memory is constantly being modified during the execution of a program. However, as Figure 2.4 clearly illustrates, it is unlikely that during an on-period of any intermittently powered embedded system, including ENGAGE, all memory is modified. Therefore, when creating a checkpoint containing all the known or active memory regions of the system, one will inevitably copy memory locations that have not changed since the last checkpoint.

It is thus desirable to copy as little of the (embedded) system state as possible while keeping the checkpointing incorruptible. The most fundamental method to do this efficiently is to track which memory regions have been changed since the last checkpoint, in other words, to see memory modification *difference* in-between checkpoints. As mentioned earlier, the only checkpoint runtime that has employed this form of differential checkpoint so far was DICE [5], see again Table 2.2. It is, however, difficult to apply the techniques used by DICE while maintaining an incorruptible system (that uses double buffering). Specifically, assuming that only one of the buffers is active, if part of the checkpoint resides in the previous buffer, and yet another checkpoint occurs, then it is impossible to keep the incorruptibility trait with DICE without still copying all checkpoint data between the two buffers. Therefore, to achieve differential checkpointing that is incorruptible, a new system has to be designed, which resulted in MPatch.

MPatch Just-in-Time Checkpoints. As we have shown in Figure 2.4 not all of the emulator and display memory is written to at every MCU clock cycle. Hence we only checkpoint the modified memory regions, which we denote as *patches*. Then, we monitor the voltage level of the storage capacitor, as in other existing runtimes, e.g. [22, 5, 315, 145] and only checkpoint the state when nearing a power failure—we call this *just-in-time checkpoint*. We purposefully do not perform checkpoints at an interval timer: game players are susceptible to lagging in a game, hence interval-based checkpointing (which introduces frequent fixed-interval delay) is not desirable.

Table 2.2: Comparison of MPatch with state-of-the-art intermittent checkpointing runtimes.

System	Incorruptible	Differential	Just-in-time	Volatile main memory	ARM support
Mementos [246]	Yes ✓	No ✗	Yes ✓	Yes ✓	Yes ✓
Hibernus++ [22]	No[1] ✗	No ✗	Partially	Yes ✓	No ✗
QuickRecall [145]	No ✗	No ✗	Yes ✓	No ✗	No ✗
Chinchilla [190]	Yes ✓	N/A	No ✗	No ✗	No ✗
Rachet [306]	Yes ✓	No ✗	No ✗	Yes ✓	Yes ✓
HarvOS [28]	No[1] ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
TICS [166]	Yes ✓	N/A	No ✗	No ✗	No ✗
TotalRecall [315]	No[1] ✗	No ✗	Yes ✓	Yes ✓	No ✗
Elastin [46]	Yes ✓	N/A	No ✗	No ✗	No ✗
WhatsNext [95]	Yes ✓	No ✗	No ✗	Yes ✓	Yes ✓
DICE [5]	No [1] ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
MPatch	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓

[1] These systems require perfect energy prediction to not get corrupted. Any changes in, for example, capacitor size [46], power consumption due to peripheral use, or harvested energy, **will** lead to incorrect predictions and therefore **corruption**.

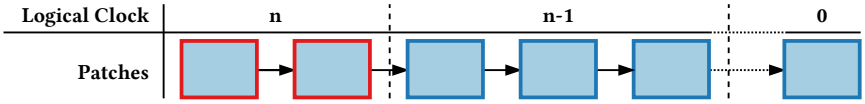
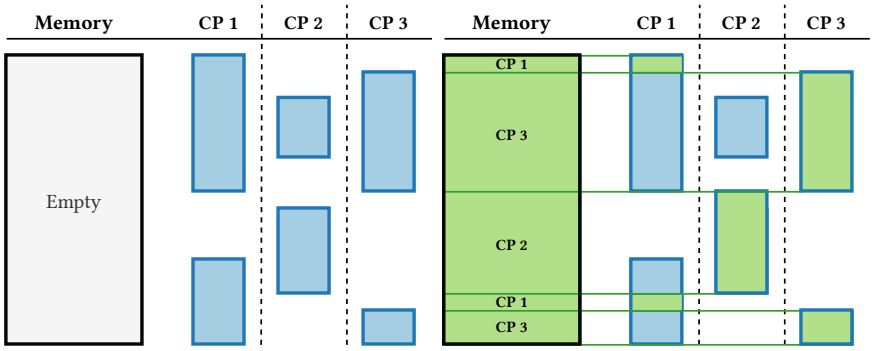


Figure 2.5: MPatch stage operation. Patches outlined with red are staged, but not committed. Patches outlined with blue signify committed patches.

Patch Handling. A patch is a non-volatile copy of a *consecutive* region of volatile memory that has changed since the last successfully created checkpoint. As different memory regions are modified during execution, multiple patches of different memory sections might be required for a complete checkpoint. During the restoration, the most recent patches (in combination with the pre-existing patches) are used to restore the volatile memory to the state it was in during the last checkpoint. By only storing the modified regions the checkpoint time is significantly reduced, as often only a small part of the memory is changed between the two consecutive checkpoints (we will investigate this further in Section 2.5).

As with traditional checkpoint-based systems that use double-buffering, an atomic variable n, determines which of the two buffers should be used to restore the system in case of a power failure [246, 166]. This variable n is changed—often incremented—to mark the completion of a checkpoint. The requirement on n is that for its increment, $n+1$, it holds that $(n \bmod 2) \neq (n+1 \bmod 2)$. MPatch patch management is also built around the atomic variable. However, MPatch extends the function of this variable to act as a *logical clock*, with the additional requirement that $n \neq n+1$.



(a) MPatch memory **before** restoration.

(b) MPatch memory **after** restoration. The restore sequence applies only the parts of patches that are required to reconstruct the memory.

Figure 2.6: MPatch patch restore procedure after three successful checkpoints (CP). For illustration, we assume that the memory is initiated as empty; blue rectangles depict patches that have been successfully committed to non-volatile memory and green rectangles signify the parts of the patches that are applied during restoration.

We now define three fundamental patch operations (i) *Patch Stage*, (ii) *Patch Commit*, and (iii) *Patch Restore*.

- *Patch Stage:* When a patch is created, the required amount of non-volatile memory is allocated and the volatile-memory is copied to the patch. Next, the patch is *staged* by signing it with the current logical clock n and added to the front of the *patch chain*, i.e. the list of patches, ordered from newest to oldest, that will be applied during restoration. Staged patches are outlined in red in Figure 2.5. While a patch is staged it will be discarded if a power failure (and thus a restoration procedure) occurs.

- *Patch Commit:* When the logical clock n is incremented, all previously staged patches will become *committed*. These patches are outlined in blue in Figure 2.5. Committed patches will be considered during the *patch restore* procedure.

- *Patch Restore:* When ENGAGE inevitably fails due to a lack of energy, it should be restored to the last completed checkpoint. Patches hold copies of consecutive volatile memory regions and are linked together to form the patch chain. This moves the complication of deciding what *part* of the patch to apply, if any, to the restore operation. To reconstruct the state of the most recent checkpoint the (partial) content of multiple patches has to be combined. This reconstruction, due to the implicit ordering in the patch chain, starts from newest to oldest. For each patch, only the parts that were not already applied during the current restore operation are copied to volatile memory, as illustrated in Figure 2.6. In contrast, for a traditional incorruptible checkpoint runtime, restoring a checkpoint means reading the logical clock n and copying the checkpoint content from the selected buffer to the corresponding volatile memory and registers.

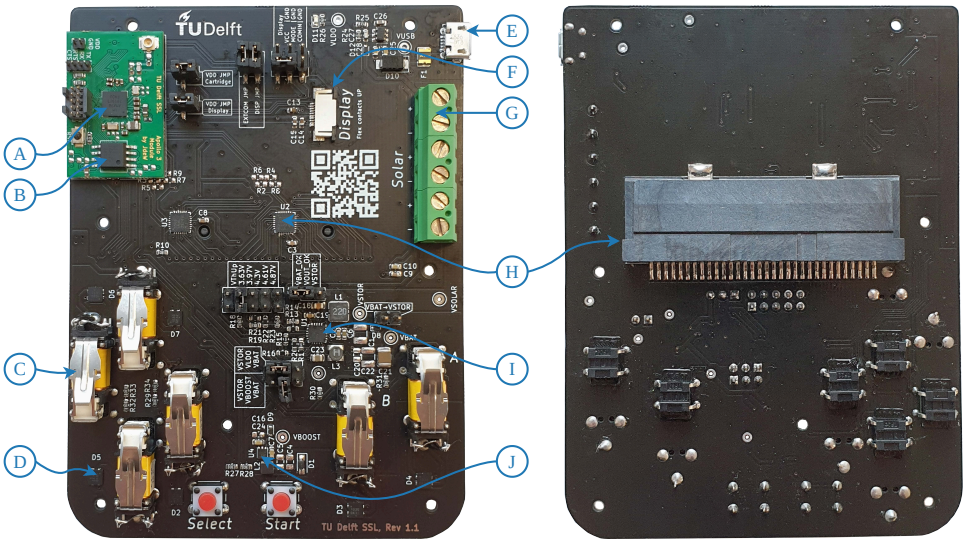


Figure 2.7: Energy Aware Gaming (ENGAGE) fabrication details. The main components are: (A) Ambiq Apollo3 Blue ARM Cortex-M4 MCU, (B) Fujitsu MB85RS4MT 512 KB FRAM, (C) ZF AFIG-0007 energy harvesting switch, (D) Semiconductor Components Industries NSR1030QMUTWG low forward voltage diode bridge, (E) micro USB debugging port, (F) display connector, (G) solar panels connector, (H) cartridge interface, (I) Texas Instruments BQ25570 harvester/power management chip, and (J) Texas Instruments TPS61099 boost converter.

2.4. ENGAGE IMPLEMENTATION

We proceed with the implementation details of ENGAGE, following from the design description provided in Section 2.3. All hardware, software and tools, as well as documentation for ENGAGE are publicly available via [72].

2.4.1. ENGAGE HARDWARE

We built a handheld, energy harvesting, battery-free hardware platform to enable the development and testing of our approach to battery-free mobile gaming. ENGAGE is built using the following components.

PROCESSING AND MEMORY

Stemming from the requirements (Section 2.2), for compatibility and popularity reasons, we build our ENGAGE around an ARM MCU architecture. However, none of the ARM architecture MCUs we are aware of contains on-chip fast, byte-addressable non-volatile memory—such as FRAM—serving as a main memory. Only slow and energy-expensive FLASH memory is present. Therefore we equip our battery-free console with external dedicated FRAM. Central to ENGAGE is the *Ambiq Apollo3 Blue ARM Cortex-M4 MCU* operating at a clock frequency of 96 MHz [7], chosen for its good energy efficiency. The Apollo3 runs the Game Boy emulator and MPatch software. External *Fujitsu MB85RS4MT 512 KB FRAM* [93] is connected through SPI to the MCU providing a fast and durable method of non-volatile storage for patch checkpoints, see Figure 2.7. With the availability of power switches within the MCU, it gates power to the screen and cartridge interface.

To be able to read game cartridges, a cartridge connector is placed on the back of the ENGAGE platform. The cartridge interfaces with the MCU using *Semtech SX1503 I/O expanders* [267], in this case the extenders also translate the 3 V system voltage to 5 V logic required by the cartridge.

MIXED SOURCE ENERGY HARVESTING

We extract energy from two sources: (i) button presses of a regular Game Boy, using mechanical off-the-shelf button press harvesters, and (ii) a set of solar panels attached to the front of the Game Boy chassis. The selected buttons were *ZF AFIG-0007 energy harvesting switches* [331]. Six of these kinetic harvesting switches are used (see Figure 2.7) located at the position of the D-pad (four switches) and "A", "B" operation buttons (two switches). The energy harvesting switches generate energy by moving a magnet inside a coil. Since both up and downward motion generates energy, the output of the switches is rectified using a *Semiconductor Components Industries NSR1030QMUTWG low forward voltage diode bridge* [265] before being boosted using a *Texas Instruments TPS61099 boost converter* [295] to be stored in a small intermediate energy storage capacitor. When the intermediate energy storage reaches 4 V the system turns on and the buck converter of the power management chip steps down the voltage to 3 V to power the system. Additionally solar energy is harvested from eight small solar panels [233], each measuring 35.0 × 13.9 mm, affixed on the front of the ENGAGE chassis (see Figure 2.7).

Harvesting of solar energy is managed by a *Texas Instruments BQ25570 harvester/power management chip* [292], integrating both a buck and a boost converter. The harvester employs a boost converter and maximum power point tracking to harvest the solar energy and stores the harvested energy in the intermediate energy storage capacitor. All energy from energy harvesting is stored in a main 3.3 mF capacitor, chosen to enable even in the worst energy harvesting conditions a few seconds of game play before the system reaches a critical voltage and powers down to harvest more energy.

ULTRA-LOW POWER DISPLAY

Displaying game content consumes the most energy of any embedded platform. Making energy-efficient display is research on its own, which is beyond the scope of this work. At the same time, we want ENGAGE to be accessible to any hobbyist by being built out of easily available and inexpensive components. Therefore we have relied on super-low power state-of-the-art off-the-shelf commercial display. Note that we have excluded e-ink displays as their refresh rates are too low for a good gaming experience (especially with rapidly changing game states such as *Super Mario Land*).

We have chosen a low power non-back lit reflective *Japan Display LPM013M126A LCD* [144], noting that the Game Boy also does not have a backlight. The LCD measures 26.02 × 27.82 mm, which means the display is smaller compared to the original Game Boy screen by 47 × 43 mm. Our chosen display offers a greater resolution of 176 × 176 compared to 160 × 144 pixels of the original Nintendo Game Boy and is capable of displaying eight colors compared to the four shades of gray the Game Boy uses. Just like in case of the cartridge interface, the MCU can enable and disable power to the screen.

FORM FACTOR AND FABRICATION

For the same gaming feel we encapsulate the electronics of ENGAGE in a 3D-printed chassis reminiscent of the original Game Boy. The only differences are: (i) the removal of sound outlet, as *we do not support sound generation on an intermittent power supply*, (ii) addition of slits to house the solar panels, (iii) slit for the USB port to provide constant power supply to ENGAGE for debugging, and (iv) removal of slits for battery charge cable and an on/off switch—as they are obviously *not needed in a battery-free system*. Since the Apollo3 Blue MCU is only available in BGA packaging, we separated the MCU from the main PCB creating a separate module containing this MCU only—see the green PCB in the top left corner of Figure 2.7—to reduce the risk of soldering errors during small batch manufacturing. This module is connected through connectors to the main ENGAGE PCB. Complete fabricated PCB, front and back, are shown in Figure 2.7.

2.4.2. ENGAGE EMULATOR IMPLEMENTATION

As many Nintendo Game Boy emulators have already been written we have decided not to build yet another one and relied on the existing emulator implementation that targets a different MCU. Specifically, to run with ENGAGE we extensively modified and rewrote a pre-existing freely-available implementation of original Nintendo Game Boy emulator targeting a STM32F7 MCU [26]. All the modifications to this emulator, enabling to reproduce our work, are part of our open-source repository freely available to download from [72].

ENGAGE Screen Handling. Due to the availability of displaying colours on the chosen display we remapped the default gray-scale colour palette (white, light gray, dark gray and black) of the Game Boy to a colour version (white, yellow, red and black). This approach is similar to popular emulators that enable colour remapping by default, transforming Nintendo Game Boy games into the modern era where most games are rendered in colour.

ENGAGE Button Handling. In Section 2.3.2 we outlined the trade-off between energy generation and user satisfaction by altering the emulators handling of the buttons, for example removing the option to hold buttons for an extended duration of time. We limit the duration a button can be held to 300 ms. This is a duration similar to the button presses per second of *Space Invaders* as shown in Table 2.1. This approach forces the user to press buttons in a frequent pace but does not require excessive button pressing, balancing user satisfaction with energy generation. Due to the flexible nature of the ENGAGE platform future work can focus on user interaction with intermittent gaming devices. We will discuss this further in Section 2.6.1.

ENGAGE Memory Configuration. The Apollo3 ARM Cortex-M4 features flash and SRAM as on-board memory, where the Flash memory contains all the code (MPatch and Game Boy game emulator code) and non-volatile game data copied from the Game Boy game cartridge. SRAM contains memory of the whole ENGAGE platform and the volatile game memory—both separated from each other. Two buffers, *Checkpoint A* and *Checkpoint B*, for double buffering during checkpointing, as well as all patches created by MPatch reside in external FRAM. The complete memory map of ENGAGE is presented also in Figure 2.8.

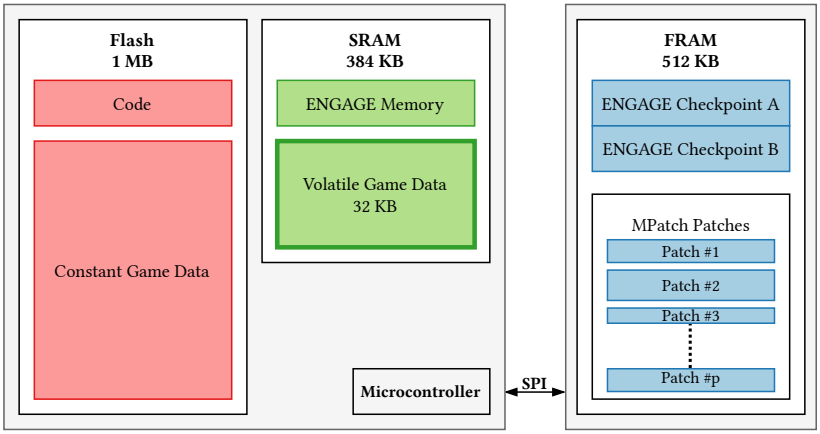


Figure 2.8: ENGAGE physical memory structure. Constant game data is executed from Flash with its volatile memory in SRAM, avoiding overhead from accessing the external FRAM. Only checkpoints and patches are stored in external FRAM.

2.4.3. MPATCH IMPLEMENTATION

ENGAGE CORE CHECKPOINTS

MPatch is built upon a basic double-buffered checkpoint scheme which we denote as the **core checkpoint** system. The core checkpoint encompasses all the emulation management logic of ENGAGE, except for the emulated game memory, which is checkpointed using patches as described in Section 2.4.3. Specifically, the core checkpoint system checkpoints the .data, .bss and active stack sections of the MCU's volatile memory as well as the registers of the MCU, as can be seen in Algorithm 2.1. All this is double-buffered in the external non-volatile memory of ENGAGE. Naturally, this means that for every byte of volatile memory in the checkpoint, we need twice as much bytes in non-volatile memory. We remark that not all memory of ENGAGE is checkpointed. Specifically, we do not checkpoint memory buffers required for peripherals (as the peripheral state needs to be re-initialized every ENGAGE reboot). The restoration of a checkpoint will restore the state of the system to that of the last successful checkpoint. If the system does not experience a *first time boot*, the default memory initialization step (which traditionally runs before any user code) will be skipped. After this, the steps listed in Algorithm 2.2 are performed to continue executing as if no power failure had occurred. In line 3 of Algorithm 2.2 the MPatch patch restoration process is started to restore the emulated game memory which will be discussed further in Section 2.4.3.

We designed the core checkpoint system from the ground up, implementing special keywords enabling the exclusion of certain volatile memory parts from a checkpoint. Also, the core checkpoint provides hooks for every stage of the checkpoint for ease of extension, which is required to incorporate patches from MPatch.

PATCH CHECKPOINTS IMPLEMENTATION

The emulated memory, i.e. the memory used by the Game Boy games, is a region in SRAM accessed only by emulated read and write instructions form the emulator. Leveraging this

Algorithm 2.1 Checkpoint Creation

```
1: function CHECKPOINTCREATE
2:     CORECHECKPOINT                    ▷ Checkpoint memory not manged by MPatch
3:     PATCHESCREATE      ▷ Create and stage patches; see Section 2.4.3 and Alg. 2.3
4:     REGISTERCHECKPOINT                          ▷ Checkpoint the CPU registers
5:     RESTOREPOINT                ▷ Continuation point after a restore operation
6:     if ISNOTRESTORE then
7:         CHECKPOINTCOMMIT                ▷ Call function that commits the checkpoint
8:     end if
9: end function
```

Algorithm 2.2 Checkpoint Restoration

```
1: function CHECKPOINTRESTORE
2:     CORECHECKPOINTRESTORE             ▷ Restore memory not manged by MPatch
3:     PATCHESRESTORE                    ▷ Restore committed patches; see Alg. 2.5
4:     PERIPHERALRESTORE                                ▷ Restore peripherals
5:     REGISTERCHECKPOINTRESTORE                   ▷ Restore the CPU registers
6:     RESTOREPOINT                ▷ Continue at the restore point; see Alg. 2.1
7: end function
```

fact makes tracking modification to the emulated memory straightforward, and doing so has little impact on the overall performance. ENGAGE tracks these modifications, and when a checkpoint is created, this information is used to create the required patches as can be seen in Algorithm 2.3. Tracking of these modifications is done using the memory protection unit of the MCU. Upon writing to a region of emulated game memory, the memory protection unit triggers an interrupt allowing the memory region to be marked as modified. After a region is marked as modified the interrupt for the region is disabled. This results in an efficient method of tracking memory writes since the introduced overhead is only present during the first write after a reboot. The memory protection unit features eight regions which each have eight sub-regions, for a total of 64 sub-regions. We equally divided the memory space of the emulated Game Boy memory between these sub-regions resulting in patches containing 32 kB / 64 = 512 B of emulated memory.

Content of a Patch. In addition to the copy of a volatile memory region, a patch contains accompanying metadata required to successfully manage and restore a patch. This metadata is: (i) the *value of the logical clock n* from when the patch was staged, (ii) the *interval of the volatile memory* that is stored within the patch, (iii) the *next patch* in the patch chain, (iv) the *metadata to build an augmented interval tree* to speed up the restoration procedure, which will be discussed later in this section.

Patch Allocation. Patch sizes are allowed to differ, therefore some form of dynamic memory allocation is required. This brings challenges, as dynamic allocation leads to fragmentation, which is undesirable in an embedded system. Therefore patches are allocated using a *fixed-size block allocator* [156]. These allocated blocks are chained together to create enough room required to store the volatile memory within the non-

Algorithm 2.3 Patch Creation

```
1: function PATCHESCREATE
2:     while p ← MODIFIEDMEMORY do ▷ For each of the modified regions of memory
3:         PATCHSTAGE(address_start, address_end) ▷ Create and stage the patch; see Alg. 2.4
4:     end while
5: end function
```

Algorithm 2.4 Patch Staging

```
1: function PATCHSTAGE(address_start, address_end)
2:     patch ← ALLOCATEPATCH(address_start, address_end) ▷ Allocate memory for a patch
3:     PATCHCREATE(patch)                ▷ Copy the volatile memory region into the patch
4: end function
```

volatile blocks. Each block contains: (i) a link to the *next block* in the chain, and (ii) a link to the *next free block* in the chain. All blocks are stored and managed in non-volatile memory. This creates challenges when trying to synchronize its non-volatile and volatile state. If these are not kept in sync, blocks will be lost, and the system may become corrupt. Additionally, write-after-read (WAR) violations [62] should be avoided when interacting with the non-volatile state. These two separate links in a block are required to eliminate one of these WAR violations, this violation could also be eliminated by introducing forced checkpoints, as inserting a checkpoint will break a WAR violation [62]. The total memory overhead of a patch in ENGAGE as it is currently implemented is 29 B. By excluding the interval tree required for the metadata, this can be reduced further to 17 B, but this would require an additional dynamic memory allocator to allocate this memory in volatile memory during a restoration (e.g. standard heap). For the final version used in ENGAGE, this was deemed undesirable, and therefore we integrated the interval tree metadata within non-volatile patches.

Patch Restoration. Restoring patches involves first discarding all staged—but not yet committed—patches, and then iterating through the patch chain while applying only the regions of a patch that were not previously applied during the restoration process. To keep track of the regions of volatile memory that were already restored we maintain an augmented *interval tree* during the restoration process. After a patch is applied, its range is added to the interval tree, and when a patch is applied, the interval tree is queried to detect overlaps. If there are no overlaps, the path is applied (i.e. written to the corresponding region in volatile memory). However, if the patch region overlaps with any region in the interval tree, the patch is split-up and all sub-patches are attempted to be applied. The complete algorithm for patch restoration is shown in Algorithm 2.5, with its accompanying patch apply algorithm shown in Algorithm 2.6.

Memory Recovery. One of the features of MPatch is its constant time patch creation while being incorruptible. However, patches that are no longer useful, i.e. that will not be applied during restoration, should be deleted. To avoid WAR violations, removing a patch (reclaiming its memory), consists of two operations. Firstly, the patch is freed, and secondly, the patch is deleted. Between these two operations, a checkpoint of only

Algorithm 2.5 Patch Restoration (note: *low*(*p*), *high*(*p*) denote the low, high component of range *p*, respectively)

```
1: function PATCHESRESTORE
2:     DISCARDUNCOMMITTED                          ▷ Call function that discards uncommitted patches
3:     while p_apply ← next(PatchChain) do         ▷ Extract next patch from patch chain to restore it
4:         PATCHAPPLY(p_apply, low(p_apply), high(p_apply))   ▷ Apply patch; see Alg. 2.6
5:         INTERVALINSERT(low(p_apply), high(p_apply))       ▷ Insert the patch range into the interval tree
6:     end while
7: end function
```

Algorithm 2.6 Patch Apply (note: *low*(*p*), *high*(*p*) denote the low, high component of range *p*, respectively)

```
1: function PATCHAPPLY(p_apply, low, high)
2:     if p_overlap ← INTERVALOVERLAP(low, high) then   ▷ Check for overlapping region in interval tree
3:         if low < low(p_overlap) then
4:             PATCHAPPLY(p_apply, low, low(p_overlap) − 1)   ▷ Recursively apply patch with a new partial range
5:         end if
6:         if high > high(p_overlap) then
7:             PATCHAPPLY(p_apply, high(p_overlap) + 1, high)   ▷ Recursively apply patch with a new partial range
8:         end if
9:     else
10:        WRITE(p_apply, low, high)                ▷ Write patch content between low and high to the volatile memory
11:    end if
12: end function
```

the MPatch management state is made containing patch and block allocation related metadata. During the deletion of a patch special care is taken to avoid WAR violations when modifying non-volatile memory in the patch chain. Memory recovery is not needed during every time a checkpoint is created or restored, is automatically done when there is no more non-volatile memory available to allocate a patch.

2.5. ENGAGE EVALUATION

We built ENGAGE as a proof by demonstration that battery-free mobile gaming was possible. In this section we demonstrate that the system can play unmodified retro games despite intermittent power failures. We analyze the real-world execution of the platform while playing *Tetris* in different lighting scenarios (i.e. with different energy scarcity) to show the effect of energy availability. We then benchmark the ENGAGE hardware platform for power consumption and, investigate the performance of the MPatch system. We find that in well-lit environments playing games that require at least moderate amounts of

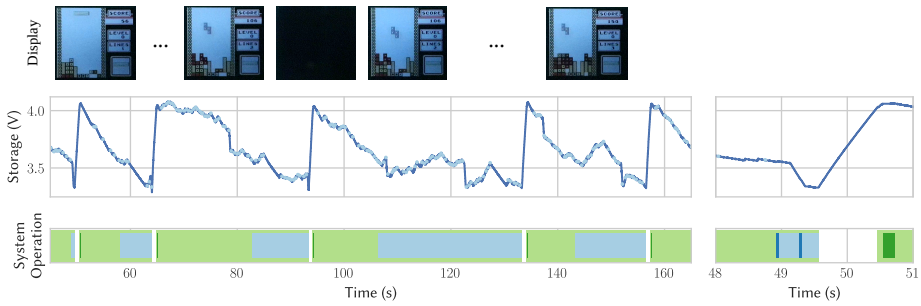


Figure 2.9: End-to-end evaluation of ENGAGE operating in 'daylight' (approximately 40 klx during *Tetris* gameplay using harvested energy only. Storage capacitor voltage is shown, overlaid by unique button presses (marked as light blue dots). Additionally the following system events are shown at the bottom of the figure: initialization time (marked in dark green), system on time (marked in light green), low energy state (marked in light blue, denoting moments of ENGAGE periodically checkpointing due to critical system voltage) and checkpoint time (shown in dark blue in the separate zoomed-in window on the right). The actual game frames are shown on top, taken from recording the ENGAGE display during the evaluation scenario. The scenario shows that user interaction prolongs the on time of ENGAGE, by pressing buttons during gameplay—achieving ten seconds or more of on time with small off times. We consider this to be a playable *Tetris* scenario.

clicking, play is only slightly interrupted by power failures (less than one second of failure per every ten seconds of play). Our measurements of MPatch across four different games show that checkpoints are fast (less than 50 ms and restoration time after a power failures is not noticeable (average of 140 ms).

2.5.1. END-TO-END ENGAGE PERFORMANCE

First, we look at the typical play of ENGAGE executing an example Nintendo Game Boy game *Tetris*, chosen due to its requirement for moderate/high button presses and a small number of cut-scenes. We show how the system operates only on harvested energy. We execute two experiments, each in different lighting conditions: (i) 'daylight' with approximately 40 klx and (ii) 'shade' with approximately 20 klx, where a gamer plays ENGAGE fully untethered, operating on harvested energy only. In the experiment the voltage of the main supply capacitor of ENGAGE is recorded together with various debugging signals indicating different system states. The system state and button presses are recorded using a Saleae logic pro 8 logic analyzer [256]. The ENGAGE platform was placed in a light box with two remotely controllable lights generating the two different light exposure conditions. The luminance of both scenarios was verified using a UNI-T UT383 lux meter [302].

In the first scenario ('daylight', Figure 2.9) we show a period of execution with both little and many button presses. Here clearly the contribution of the energy harvesting by the switches is shown, significantly prolonging the on time of the device (marked in green). The figure shows the complete sequence from startup until the ENGAGE reaches a critically low energy level when it starts checkpointing. Due to the variability in the incoming energy pattern, ENGAGE can spend some time in this state, since it always needs to account for the worst-case scenario of no additional incoming energy. This

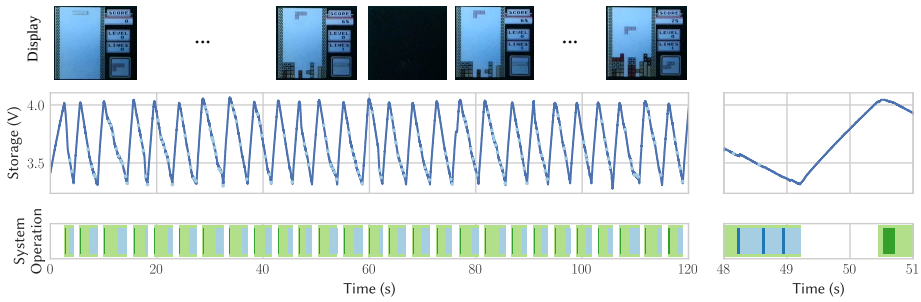


Figure 2.10: End-to-end evaluation of ENGAGE operating in 'shade' (approximately 20 klx. Description of figure elements is the same as in Figure 2.9. With less energy available to ENGAGE as in the scenario in Figure 2.9, on times are reduced to around 3.5 s, with off times of more than a second. This scenario creates a noticeable impact to the user experience.

scenario results in on times of ten seconds or more with small off times of less then a second, making it a very playable experience.

In the second scenario ('shade', Figure 2.10) we halved the amount of light the solar panels are exposed to compared to 'daylight', a more challenging condition for ENGAGE. This reduces on times to around 3.5 s with off times of more than a second. Despite the system still functioning correctly the lack of incoming energy becomes noticeable and even button mashing cannot compensate for the lack of energy. As with any energy harvesting platform, the limits of operation are defined to a major degree by the available energy in the environment. Full-system emulation is challenging and energy-intensive, but the game is still playable and functional; just with longer intermittent outages. We note that the downward peaks of storage voltage in Figure 2.9 and Figure 2.10 are caused by the energy harvester: during maximum power point tracking no energy is harvested causing the quick drop in the storage capacitor voltage.

Full-System Restoration Time. We have also measured end-to-end time of ENGAGE restoration: from the moment of applying power to the MCU to the moment of executing game code within the Game Boy emulator. In the case of *Tetris* this is 264 ms. The other games we tested resulted in comparable restore times, the main difference resulting from MPatch operations, as is further described in Section 2.5.3.

2.5.2. ENGAGE POWER CONSUMPTION AND ENERGY GENERATION

We have measured ENGAGE's power consumption, looking into overall power consumption whilst first measuring the consumption of MCU together with the FRAM and display. The MCU and FRAM combined consume 11.15 mW and the screen consumes 344.31 µW during game execution taking a ten second average. During idle time the screen only consumes 3.90 µW, resulting in a combined system average power consumption of 11.50 mW. As a comparison, the original Nintendo Game Boy consumes 232.08 mW during game execution, varying slightly per game and cartridge architecture. While not necessarily a useful or meaningful number, we conclude that our platform is *more than 20 times* more power-efficient than the original Nintendo Game Boy (representing normal tech-

nology advancement, but noting that ENGAGE is an emulator). The measurements were conducted using a Fluke 87V [89] multimeter and the X-NUCLEO-LPM01A [281] programmable power supply source with power consumption measurement capability.

Energy Generation. Then, to give more insight in the energy harvesting on the ENGAGE platform we have measured the amount of energy the solar panels generate using a Fluke 87V [89] multimeter and compare this to the energy generated from the buttons. For the buttons we use the minimal energy generation figures from the specification of the harvester [331, summary] as a worst case scenario[3]. Assuming that a single button press generates a minimum of 0.66 mJ and knowing the amount of button presses per game is specific to the game as per Table 2.1, we can assume the buttons generate between 0.66 mJ for one press per second and 1.98 mJ for three presses per second. At 40 klx and 20 klx, the solar panels generated an average of 10.14 mW and 8.33 mW, respectively, i.e. less than the required system average power consumption of 11.50 mW. We can conclude that ENGAGE is mostly powered by solar panels and supplemented by the button presses although the button presses can significantly increase the on-time of the platform, as shown in Section 2.5.1.

2.5.3. MPATCH PERFORMANCE

To better understand and quantify the effect of patches on the checkpoint and restore time, we evaluate MPatch against a *naive* approach—comparable in operation to Mementos [246]—where all active memory in the system is copied to non-volatile memory during a checkpoint, even if it was not modified since the last checkpoint. We compare these two strategies, MPatch and *naive*, by running multiple different games on ENGAGE. These games include: (i) *Tetris*, (ii) *Super Mario Land*, (iii) *Space Invaders*, and (iv) *Bomberman*. These games represent a wide variety of play styles, developers, and even release dates.

MPATCH CHECKPOINT TIME

To measure only the impact of the MPatch patch checkpoints, we disable the just-in-time checkpoints—used in Section 2.5.1—and run the system on constant power during these measurements. Instead, we perform a checkpoint every c execution cycles of the emulator, we chose three different values for c, which correspond to different *on times*, i.e. 1 s, 5 s, and 10 s. During normal operation checkpoints will only be created when the voltage reaches a critical threshold, as seen in Section 2.5.1, these fixed on times represent a simplified scenario where the critical voltage threshold is reached after the specified on time. The *on time* affects the number and size of the checkpoints, as it allows for more memory writes between two consecutive checkpoints. The on time does not affect the *naive* checkpoint, as it always checkpoints all memory, with the only variable size being the system stack of ENGAGE. However, because of the way ENGAGE works—as an emulation loop—the system stack size is virtually constant.

During the emulation of each game, with the three different on times, we measured the cost of each component of the checkpoint using the same logic analyzer as used in experiments in Section 2.5.1. A checkpoint of ENGAGE consists of a core checkpoint of

[3] Harvesting energy from the button energy harvesters is highly dependent on numerous factors such as the force applied and the manner of pressing the button hence the choice for the minimal figure.

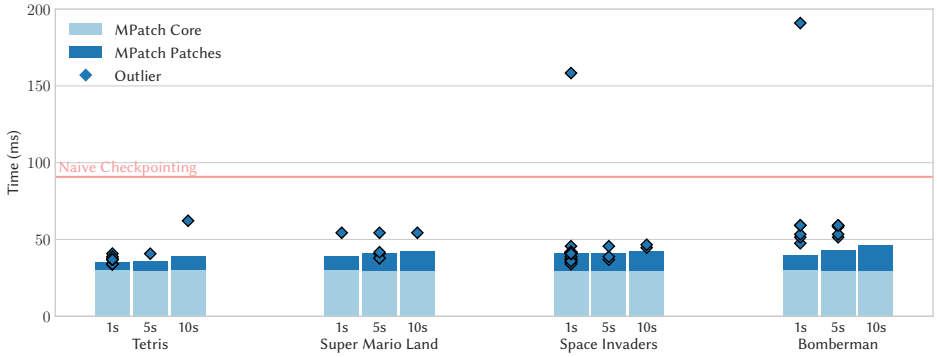


Figure 2.11: MPatch checkpoint time comparison of approximately two minutes of game play per game using three different on times (1 s, 5 s, and 10 s) between successive checkpoints. ENGAGE has noticeably better performance than naive system, across all on times and games.

ENGAGE (Section 2.4.3) and additionally patches created by MPatch. The core checkpoint includes the management of both the emulator and the emulated memory, but excludes the emulated memory itself. This emulated memory is the largest memory component of the system, and therefore also the largest component of a naive checkpoint. For this reason we checkpoint this part of the system using MPatch, as the other components of ENGAGE are virtually constant in the amount of memory that is modified and are thus covered by the core checkpoint.

Figure 2.11 illustrates the naive checkpoint time as the horizontal line, the average checkpoint time of a core checkpoint (light blue bar), the differential component of a checkpoint using MPatch (dark blue bar), and the outliers (blue diamonds). As can be seen, the cost of the core checkpoint is around 30 % of the complete *naive* checkpoint, the rest being the emulated memory. However, when using MPatch to checkpoint the emulated memory, the core checkpoint dominates the total checkpoint time. In total MPatch is on average *more than two times faster* than the *naive* approach. This confirms our hypothesis that only a small amount of emulated memory is modified during execution. This reduction in checkpoint time directly leads to a lower energy requirement for each checkpoint and leaves more time for game emulation. Interestingly this assumption seems to hold even when the on time approaches 10 s, which is substantial for intermittent devices. Some outliers take longer than a *naive* checkpoint, this is due to a periodically performed memory recovery procedure (Section 2.4.3)—which was introduced to keep the creation of patches constant while keeping the system incorruptible.

MPATCH RESTORATION TIME

We also evaluate restoration time of patch checkpointing of MPatch, in a similar manner as in the previous section (i.e. the same set of games, comparison against three other reference mechanisms). The results are presented in Figure 2.12.

Restoring a patch-based checkpoint requires more time than the creation of a patch, as described in Section 2.4.3, due to the need to apply only the parts of the patches that are required, and because all the volatile memory has to be restored. Additionally, the

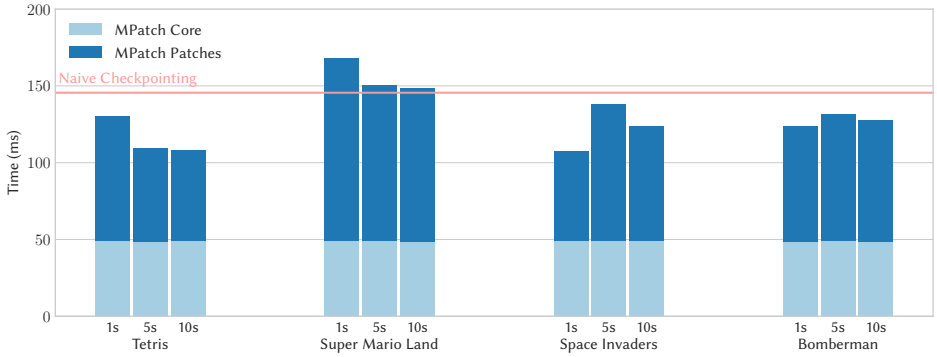


Figure 2.12: Restoration time comparison of after approximately two minutes of game play per game using three different on times (1 s, 5 s, and 10 s) between successive checkpoints. ENGAGE has comparable or better performance than naive system, across all on times and games.

restoration procedure must take into account all the committed patches when trying to restore the volatile memory, as each of these might hold some region that was only checkpointed using that specific patch. Therefore it is not directly influenced by the on-period, but influenced by the time since a *memory recovery*. Nevertheless, as can be seen in the figure, MPatch often *reduces the restoration time* compared to the *naive* restoration. We can also conclude from this that tested games often only modify a portion of their memory (in this case the emulated memory), as can also be seen in Figure 2.4.

2.6. DISCUSSION AND FUTURE WORK

Our evaluation of ENGAGE has shown that retro games are playable without batteries, making a next step in self-sustainable gaming made first decades ago by e.g. Bandai Corporation's LCD Solarpower game series [58]. Although the core gameplay mechanisms of the mobile handheld gaming have been successfully implemented, i.e. interaction with a screen-displayed data (for the original Nintendo Game Boy), other forms of interaction that make game experience complete are waiting to be researched and implemented.

2.6.1. LIMITATIONS, ALTERNATIVES AND FUTURE WORK

Of course ENGAGE is just a first step in the direction of battery-free gaming and the proposed platform has still many limitations that need to be addressed. First, our battery-free platform *plays no sound*. We agree that no sound play is the main hurdle of complete game immersion. How to make sound enjoyable despite power supply intermittency is the core research question, but at the same time (in our opinion) an exciting research area. Some approaches to the sound problem we anticipate as worth-considering are (i) to include separate storage for sound buffering and play, following the architecture of [51, 113], (ii) introduce superficial pauses in the original game tone—effectively making the game sounds identical to the original battery-based game but punctured by silence at pre-selected moments—to make sound interrupts less irritating during gameplay, or (iii)

to create intermittent system-specific game sounds—sounds that inform the user that the system is about to die or has just become operational again—to enrich battery-free gameplay.

Second, *playing in the dark has not been addressed*, as screen we used has no backlight[4]. In the context of battery-free gaming, provision of a backlight for screen is very difficult. Simply, lack of light reduces amount of energy from harvesting, which in consequence reduces chance to perform *any* task—let alone supporting LEDs that are the most energy-consuming components of any embedded system.

Third, *haptics for battery-free games needs deeper investigation.* The energy harvesting buttons we have used in our prototype [331] are designed for industrial *sporadic* single-press cases (think of a battery-free wireless light switch). In frequent pressing cases these switches are much sturdy than the original Game Boy buttons, which for gamer can be a distracting feature. This necessitates a quest for more natural press buttons with equal (or better) energy harvesting. Furthermore, solar panels have to be placed on console's chassis such that their obstruction by fingers is minimized (as in our prototype). This, however, might downgrade the aesthetics of the device or require to make it bigger.

Fourth, *networking with battery-free game consoles* is another important point to consider, which was not addressed by us. Original Game Boy had an ability to connect, via cable, to another Game Boy for tandem gaming. This cable connection can be actually used for energy sharing and balancing between two consoles. Wireless networking of battery-free is an ongoing research task, which we did not want to cover with this work (for state of the art battery-free networking overview we refer to Section 2.7).

Fifth, *we cannot claim that all games will have the same playability* when ported to the intermittently-powered domain. Only when the off-times are negligible for the player we can safely assume that any existing game could be played intermittently/battery-free. Negligible off-times will cause no irritation to the person who is accustomed to always-on style of play. This observation would hold for any game system—not only classical (but old) Nintendo GameBoy we used as a basis for ENGAGE, but also recent systems such as PlayStation Portable or Nintendo Switch. An open research question is to find how long this off time is (less than a second or maybe less than a millisecond)? Our intuition says that this time is game-dependent and the longer the off times are present in a battery-free console, the set of games that can be ported to the battery-free platform gets smaller. Games that do not need frequent button pushes intuitively would be less irritating to play intermittently (e.g. *Chess*) or *Solitaire*), refer also to qualitative comparison of 8-bit Nintendo Games portability in Table 2.3. However, this creates an interesting paradox of button-based interaction. More button presses during the game result in more energy supplied to the game console, see also Table 2.1 and Section 2.3.2 (in extreme case games that are based on button bashing, such as classical *Track & Field* arcade game from Konami Corporation, gamer would be able to continuously generate energy purely from gameplay). At the same time less button presses result in less energy being created, causing reduction in continuous duration of play. To verify the above claims *detailed user studies considering large pool of gamers and games* need to be performed, where users play different games with artificially-induced intermittent operation (varying duration of on and off times).

[4]To be fair, the original Game Boy had the same deficiency, so do some of the upcoming gaming consoles [234].

Table 2.3: This table describes the difficulty (or irritability) of playing types of Nintendo GameBoy games on intermittent power, assuming the intermittent effect is noticeable to the player and that enough energy is available for some level of play.

Game name	Type	Button presses	Intermittent play	Comments
Baseball	Sports	Very High	Hard	Reaction time is part of the game
Super Mario Land	Platformer	High	Hard	Button press order is crucial
Tetris	Puzzle	High	Medium	Tile rotation is often infrequent
Solitaire	Cards	Low	Easy	No penalty for missing a press
WordZap	Puzzle	Low	Easy	Easy with "no solving time" penalty
Chess	Strategy	Very Low	Easy	Most time spent on thinking

Sixth, *screen retention needs to be introduced* (keeping screen state in-between on times) which our ENGAGE has not implemented yet. This simple extension would significantly reduce perceived negative effect of intermittent operation (think of a *Chess* game where state of the screen does not change much when the player is thinking and often user would not distinguish between off time and regular game operation, see again Table 2.3).

Finally, the overarching goal is to be able to *play state of the art 21 century handheld game consoles battery-free*, such as the Playstation Vita or Nintendo Switch—going beyond 8 bit architecture. This however requires years of research and can only be achieved by further advances in intermittently-powered software frameworks and ultra-low power electronics, which hopefully this work made a first step in achieving this goal.

General Software Framework for Battery-free Games. The goal of being able to run any existing or future game battery-free requires the introduction of general software framework for such games, going beyond checkpoining mechanism or a driver design presented in this paper, which is of course tailored towards the 8 bit Game Boy emulator. We envision a game engine, inspired by game engines of existing video games, such as the *Source* game engine [305] used in first-person shooter games such as *Counter-Strike*, that supports battery-free interaction abstracting underlying frameworks for intermittent operation form an actual game design.

Further Reduction of Game Console Carbon Footprint. We have made a first step towards making Game Console fabrication more environmentally friendly, however this is just a first step. Needless to say, original game cartridges of Nintendo Game Boy contain battery, and our console is based on many electronic components that are responsible for large CO_2 emissions in production [107], not to mention chassis that is made of plastic. More radical ideas need to be exploited, such as on the electronic level design with minimum amount of components (e.g. crystal-free design), going beyond policy changes in electronic fabrication enlisted, e.g. in [99].

Behavior Nudges to Generate More Energy. Many types of games have natural gaming mechanics that could be leveraged to increase energy harvesting actions. *Dance Dance Revolution*, *Bop-It*, and others, exploring this gaming induced behavior change for increasing energy is an interesting research direction. For example, a specific rapid button pressing sequence can trigger new game events (new levels, extra game points, etc.). Then, there are great user interfaces for battery-free interaction, for instance a crank[5], that can be researched further.

Native Execution. We chose the hard path: running a game emulator on an intermittent platform. This was to demonstrate the range of capabilities available to intermittent computing, and to leverage the vast amount of pre-built games that can play unchanged on the platform. However, one could imagine that native gameplay would significantly increase the performance of the platform, by orders of magnitude, since a single emulated instruction has significant overhead over native code for the platform. This could be accomplished by compiling game binaries to native ARM code, or by leveraging a bespoke gaming API from bare-metal C code. The latter is intriguing as an exercises to take advantage of the unique aspects of intermittently-powered and battery-free gaming, where the situation and context, as well as the gameplay, will effect how much energy is harvested. Game mechanics leveraging this system attribute might increase engagement.

2.6.2. GAMING AND THE ENVIRONMENT

Electronic games are an important part of the world's economy [209]. First and foremost they are crucial to mental well being of many people around the world. Especially in the time of the COVID-19 pandemic, when millions of people are stranded at home, various forms of electronic gaming are one of the activities that reduce stress and boredom due to lockdown implemented by most of world's governments [260, 262, 290]. At the same time, the electronic gaming industry is and important job creator, and although being a financially non-struggling industry, to say the least [223, 262, 209], is also actively supported by international governments': as an latest example refer to CD Projekt—creator of *The Witcher* video game series—and its list of European Union-funded projects the company participated in [41]. At the same time it is apparent that gaming industry contributes significantly to global warming. In the United States alone gaming is responsible for "24 MT/year of associated carbon-dioxide emissions equivalent to that of 85 million refrigerators" [202]. To tackle that challenge the gaming industry is joining various industry consortia such as *Playing for the Planet* [243] aiming at reducing its ecological impact. Independently, some national governments aim at influencing the gaming industry requesting content providers to throttle-down data rate of streaming services with too high demand [99]—resulting in smaller electricity consumption of data centers.

But all the above actions to address climate impact of the gaming industry do not tackle the effect of battery-based/handheld/mobile gaming (the above-mentioned study of [202] explicitly excludes such devices from the analysis). Beyond any doubt handheld gaming devices, while extremely popular [214], contribute independently to increased worldwide CO_2 emissions. While we are not aware of any detailed studies on the carbon

[5]Which is already used in the upcoming post-retro *Playdate* console [234], which sadly is not used for internal battery charging.

footprint of popular handheld gaming consoles, such as Nintendo Switch[6], its impact is beyond negligible. For example, Nintendo was *the least* environmentally-friendly company of Greenpeace 2010 *Guide to Greener Electronics* ranking [329], while none of the video-game oriented companies are listed among the world's most sustainable corporations in year 2020 [57].

There are numerous components that gaming handheld console/mobile phone is made of that cause substantial environmental impact [107] so removing some of them without compromising the usability would by highly appreciated from the environment point of view. A first potent candidate for such removal is a battery. Production of batteries has great environmental impact by itself and many research projects are devoted to making *batteries-only* more sustainable [81]. But even if most of the goals of more sustainable batteries are met by 2030, they will *still* have to be produced, collected and recycled. And while we conjecture that majority of console game players do not consider reliance on batteries as a problem[7] it is the *environmental responsibility of the electronic designers* to address the battery issue for the users of handheld gaming consoles.

2.7. RELATED WORK

Battery-free Sensors. Long before our idea of a batter-free gaming console, non-gaming embedded platforms were realized in a battery-free manner—making these sensor more environmentally-friendly. The first such battery-free platforms were wireless sensors [238]. First battery-free sensors were based on the idea of computational RFID tags: programmable RFID tags with on-board sensors (such as accelerometers or temperature sensors) communicating with the outside world by radio frequency backscatter to a RFID reader. WISP [304, 257] and Moo [303] are the first realization of such RFID tags. Since the introduction of WISP and Moo many research groups have focused on making battery-free backscatter communication more efficient [317], for instance, by making it free from dedicated energy sources [236], by enabling communication with non-backscatter networks such as IEEE 802.11 [155] or LoRa [287], or by improving backcatter-based networks—either based on standard RFID protocols [188], or based on dedicated backscatter network stack [111]. A separate line of research focused on introducing camera-based image processing to backscatter-based sensors. First, a backscatter-based battery-less cameras, as an extension to WISP platform, has been demonstrated in [207, 208], later followed by a dedicated (non-WISP) backscatter-based system [206, 253]. Additionally, non-radio frequency backscatter systems based on passive visible light communication backscatter, such as PassiveVLC mote [319], have also been demonstrated. It is important to remark that the biggest drawback of backscatter-based systems is the reliance on external energy source (itself powered by batteries or

[6] None of the handheld gaming platforms are listed in the Electronic Product Environmental Assessment register [108]; the closest study of environmental impact of Nintendo Switch we are aware of is given in [167]. As a reference, in-depth analysis of carbon footprint of one of the most popular non-handheld gaming console, Sony's Play Station 4, is available in [107]. To quote from this study: "(s)ince the PlayStation 4's release in 2013, approximately 8.9 billion kilograms of carbon dioxide have been generated and subsequently released into the atmosphere".

[7] Actually, in many cases game console players are close to a power socket playing their games tethered, making a problem of battery replacement or recharge even less profound.

power line) that downscales the benefit of removing battery from a complete system.

Additionally, battery-free sensors that communicate using non-backscatter, i.e. active, communication techniques also become actively researched. These include simple sense and transmit sensor powered by ambient temperature differences [336], UFoP [113] and Capybara [51]—energy-harvesting storage-adaptive sensors, Battery Free Phone [288], SkinnyPower—wearable sensor powered by intra-body power transfer [272], Camaroptera — image-inferring sensor [210] or SoZu—battery-free activity detector [335]. Non-wireless/non-communicating battery-free sensors include CapHarvester—local energy monitor powered by harvesting stray voltage from AC power lines-[109], self-powered step motion counter [149], Saturn—battery-free microphone [17], and active radio battery-less eye tracker [173].

Battery-free Interactive Devices. It is imperative to extend battery-less devices beyond a simple 'sense-and-transmit' functionality (as summarized above) demonstrating simple forms of user interaction. The same RFID technology that lay the foundation for battery-free sensing was also used to demonstrate battery-less interaction. Such systems include RFID-based tags displaying external information [226], elderly monitoring based on embedded-in-clothes RFID tags [146], surface shape detection [148], speech recognition [312], augmented reality with (i) unmodified RFID tags [172] and (ii) modified RFID tags (to enable touch sensing) [120], interactive building block system with augmented RFID tags[8] [178, 121] or finger gesture measurement [152]. It needs to be emphasized that any RFID tags-based interaction is sensitive to interference as demonstrated in [311].

Separately from RFID-based battery-free interactive devices, non-RFID counterparts are also actively researched. Most of these devices focus on remote device control through touch. Examples of such devices are capacitance-based touch sensor (although communicating with FM radio receiver through backscatter) [310], Ohmic-Sticker—force-to-capacitance sensor attachable to laptop touchpad [125], aesthetically pleasing self-powered interactive surfaces based on photovoltaic cells [197] and self-powered gesture recognition based on (i) photovoltaic panels [186][9], (ii) photodiodes [175] and (iii) capacitance sensing [301]. E-ink battery-free wearable displays embedded in clothes, energized by NFC-enabled smartphones were demonstrated in [79].

Another approach for battery-free embedded devices is to equip the area where the sensor resides in some form of wireless power transfer system. Many end-to-end wireless power solutions can be found in the literature, including recent systems build on top of capacitive power transfer [332], magnetic resonant coupling [284], quasistatic cavity resonance [259], lasers [140] or distributed RF beamforming [87]. As in the case of backscatter-based sensors, wirelesly-powered sensors require external (complex, bulky and still having not fully resolved safety issues) infrastructure. This limits applicability of this approach to ubiquitous battery-free gaming.

Battery-free Gaming. An ultimate form of interaction is through a gaming system. A first, commercial battery-free/solar-powered gaming platform was Bandai's LCD Solarpower [58], released already in 1982, that enabled manipulation of hard-coded elements on a liquid crystal display. Unfortunately, Bandai's console and modern existing

[8] A similar concept for NFC-based tags has been presented in [36].
[9] System claims to be battery-less, while in evaluation a battery-based version was used.

academic-grade battery-free gaming systems are limited to a simple game forms, such as attachable touch pad extenders for better (but still battery-powered) mobile game experience [44, 327] (similar to an earlier referred design [125]), extra controllers for smartphones based on its front/rear cameras [320], or based on RFID technology that requires heavy-lifting of battery-less features by an expensive RFID reader using either (i) computational RFID tags [303, 304] as for instance in [196], or (ii) using commercial off-the-shelf RFID tags as in [171]. Battery-free non-RFID touch pad extender for the introduction of physical manipulation into touch screen-based games was prototyped in [212]. Battery-free gaming aimed at children includes system based on rubbing/-touching electrostatic surfaces to power simple electronics [150, 43, 42] and attachable energy harvester mote for learning and understanding concepts of energy generation and consumption [252].

New Electronic Game Forms. Battery-less handheld gaming console such as ENGAGE presented in this chapter introduce a novel form of self-powering play, where user (to continue playing a normal electronic handheld game platform) is (sometimes) required to push buttons to continuously power a device. This is a twist on movement-inducing (exer)games [139, 24] such as Pokémon GO [161] where movement is required only to *perform better* in game instead of *perform better and continue* to play. This is a new form of game interaction that use the human body as an immanent component of gaming experience, as advocated in [205]. We note that novel forms of games with dedicated hardware (albeit battery-powered) are introduced, where the energy of the body is used to introduce a novel form of interaction. A recent example of such game is based on swallowable temperature measurement pills [177] or through-body electric field propagation [308, 307].

Gaming as a Behavioral Intervention. Research community is in constant search for new forms of gaming interaction and our battery-less gaming console aims at defining yet another gaming behavior. Such new forms of gaming are for instance, 'idle games' [6] or new game forms with custom-made haptics, such as virtual reality games for blind people [261]). Design challenges in behaviour-inducing games (such as exergames referred earlier) have been discussed recently in [161].

Considering classical gaming behavioral studies we can refer to game design that activate children to play outdoors [225], study on the effect of 'gamification' of cognitive tasks [313] or observation of gaming experience as an indication of cognitive abilities [137]. We are not aware of any non-orthodox gaming behavioral studies.

A separate line of research, although not strictly related to games, touches upon behavioral change of battery-powered smartphones usage. These studies include crowdsensing of battery usage for suggestion of better user behaviour extending battery lifetime [48], optimization of frame rate for mobile (smartphone-based) games saving energy on frame rendering [124], or a proposal for new form of interaction with mobile devices with turned-off screen to conserve energy [318]. Our battery-free game console is the first study that considers a behavioral intervention for *battery-free* device.

Sustainable Design of Interactive Devices. Design of any future interactive devices must consider sustainability and reuse, as advocated already a decade ago in [29, 195]. The same plea, but in the context of pervasive devices, was presented in [142]. Since

almost a decade many studies call for sustainable 'upstream' HCI by making conscious choices in HCI design process in selecting materials that are sustainable, recyclable and reusable [157] or using post-apocalyptic terms— HCI "designed for use after the industrialized context has begun to decay" [300]. We are unaware of any studies on whether the (handheld) gaming community considers sustainable gaming as important, let alone existing, problem. Loosely related study to our posted problem is the study on the motivations behind leading green households [316].

Intermittent Computing Systems. The goal of intermittent computing frameworks is to guarantee correctness and completion of the computation of battery-less energy harvesting embedded platforms *despite* frequent power interrupts[10]. Such framework is essential for the usability of battery-free gaming platform.

From the publication of a first framework supporting intermittently-powered devices, Mementos [246]—voltage threshold-triggered checkpoining system, more efficient checkpoint systems are being published. These include Hibernus++ [22] and QuickRecall [145] (just like Mementos, both hardware-activated checkpoints), Chinchilla [190], Rachet [306] and HarvOS [28] (all three compiler-instrumented checkpoints), TICS (time-aware checkpoints) [166], TotalRecall (checkpoints using volatile memory) [315], Elastin (adaptive checkpoints) [46], DICE (differential checkpoints) [4, 5] and WhatsNext (checkpointing augmented with approximate computing) [95]. A second class of systems include runtimes based on specially instrumented code (by form for a *task*) such as Dino [185], Chain [50], Alpaca [189], MayFly [116], InK [325], Coati [251], CoSpec [47] and Coala [193]. A third class of intermittent computation support systems are hardware-assisted systems such as Clank [119] that check for memory inconsistencies. Important to recall are workload-specific computation systems such as on-device inference on intermittently-powered devices with off-line and on-line learning, see [100] and [169].

A separate stream of work targets peripheral support for intermittently-powered devices, such as Restop (through dedicated middleware) [18], Samoyed (through just-in-time checkpoints) [191] and Karma (supporting parallel or asynchronous peripheral operations) [34], or targeting handling of dedicated peripherals such as e-displays (to improve their update rate) [198].

2.8. CONCLUSIONS

This chapter presented a first working example of a battery-free gaming console, and the first full system emulation on intermittent power: ENGAGE. We demonstrate we can port existing battery-based gaming platforms—such as in our case 8 bit Nintendo Game Boy—to the battery-free domain. With this platform we have shown that deeply interactive devices, like gaming platforms, are possible to create without batteries, and in spite of frequent power failures, addressing the interactive devices challenge. We developed a novel hardware and software platform to facilitate this new class of device: (i) a hybrid energy harvesting device tailored towards battery-free gaming and (ii) a new system for persistent computation across power failures based on a novel concept of patch checkpointing of volatile memory state into non-volatile memory regions. ENGAGE represents a bright future of deeply interactive, maintenance and battery-free devices.

[10] For a good overview of intermittent computing concepts we independently refer to [199, 184, 115].

CHAPTER 3
BATTERY-FREE DEBUGGING

This chapter is based on:
Jasper de Winkel, Tom Hoefnagel, Boris Blokland, and Przemysław Pawełczak (2022).
DIPS: Debug Intermittently-Powered Systems Like Any Embedded System
Proceedings of the 20th Conference on Embedded Networked Sensor Systems. ACM, Boston, MA, USA, 222–235.

3.1. INTRODUCTION

Despite the increasing number of battery-free intermittently-powered platforms, they are still difficult to program [164, Section 7], as we found out during the development of ENGAGE (Chapter 2). This difficulty stems from ensuring correct continuation of program execution after restarting from a power interrupt. For intermittently-powered devices the application developer must programmatically account for two events. That is, whenever a power interrupt happened, at any place in the code, the device (i) must resume operation from the moment that power interrupt happened, and (ii) the state of the device's memory and its peripherals must be correctly restored. Sadly, debugging of software written for battery-free intermittently-powered devices is itself hard [49, Section 2.2]. This is because above the existence of 'normal' bugs (not related to intermittently-powered operation) one has to additionally deal with bugs resulting from these power failures. Unfortunately, the debuggers developed for battery-powered embedded systems, such as [263], assume the Device Under Test (DUT) is continuously powered in order to debug. This effectively removes the ability of code debugging, as with every power failure the debugger has to be reconnected manually.

To the best of our knowledge there is only one dedicated debugger targeting intermittently-powered devices, i.e. EDB [49] that addresses some of the core limitations of existing debuggers for embedded systems. Nonetheless, to debug code with EDB one has to instrument the code manually with EDB-specific API for software based assertions and breakpoints. This results in a time-consuming debug process, as for each new assertion or breakpoint the code must be recompiled and the bug scenario has to be recreated. Then, each breakpoint has to be manually enabled when starting the debugging session. When an assertion is triggered, each variable responsible for triggering the assertion has to be individually investigated by first looking up the address of the variable and then reading the memory at that address. This does not allow the user to (i) easily inspect all memory variables or the call stack in a breakpoint or (ii) transitions from one task to another. But what is more important, EDB breakpoints themselves might mask intermittency-specific code bugs—as we will show later in this chapter—which is detrimental to the debugging process.

Furthermore, EDB does not allow for replay of energy traces powering the battery-free device. Instead, EDB makes sure that the device storage capacitor is charged from the instrumented assertion/breakpoint to keep the device alive, discharging it after the assertion as if no assertion was included in the code. This allows for code checking without an energy penalty to the device. Energy trace replay, however, would allow for repeatable results and the ability to induce time-specific power interrupts. This is unfortunately impossible with EDB-style debugging where the DUT is powered from uncontrollable energy harvesting sources (in the case of EDB—an RFID transmitter).

To solve the debugging problem for intermittently-powered devices our idea is to bring two necessary embedded debugging components together in a single debugging platform. These components being: (i) a fully featured hardware debugger based on GNU Debugger (GDB) [229, 278]—to enable step-by-step debugging in the way the majority of existing (non-intermittently-powered) embedded platforms are being debugged right now, and (ii) an energy emulator capable of replaying energy traces, allowing to power battery-free platforms from the same energy trace (either pre-recorded or synthetically

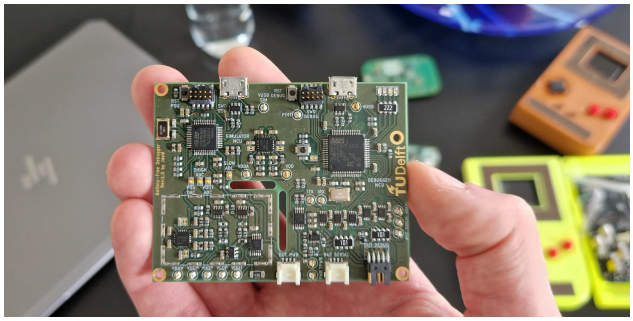


Figure 3.1: Photography of Debugger for Intermittently-Powered Systems (DIPS) hardware. DIPS is a new hardware/software ecosystem designed for debugging and testing intermittently-powered battery-free devices.

generated) repeatably—to emulate specific intermittency patterns, such as in [96]. The result is a new debugging platform named Debugger for Intermittently-Powered Systems (DIPS), as shown in Figure 3.1.

The contributions presented in this chapter are as follows.

- **Hardware-based debugging on intermittently-powered systems**: We introduce the first hardware based debugger for intermittently-powered systems, capable of utilizing the hardware debugging features of the microcontroller under test, despite being intermittently-powered. Our debugger does not require any software modification to the Device Under Test (DUT) and is based on GDB, resulting in direct integration into most Integrated Developer Environments (IDEs). This allows for rapid debugging of code for intermittently-powered devices—in an identical fashion compared to classical embedded devices. This observation is echoed by the user experience study of DIPS vis-a-vis state-of-the-art debugger: EDB [49].
- **Tightly-coupled energy emulator**: Unlike other systems our emulator tightly interconnects with the debugger and pauses emulation when, e.g. breakpoints are triggered, keeping the DUT powered and seamlessly resuming emulation after the user resumes execution. Our emulator is not only capable of providing synthetic test patterns to power the DUT but is also capable of mimicking the power supply circuit commonly used in state-of-the-art intermittently-powered systems: the buck-boost converter and the storage (super-) capacitor. This allows for easy and quick experimentation to determine the energy input requirements of the DUT and to find an optimal capacitor size for the system. For the first time, these features allow the developers to debug and test energy-related bugs in a repeatable fashion.
- **Automated testing for intermittency-related bugs**: Intermittent systems rely on saving and restoring the state of the system to a non-volatile on-board memory. Using an automated scripting framework we are able to verify if the volatile memory of the intermittently-powered device has correctly been restored from the last checkpoint. Not only volatile memory consistency is automatically checked using DIPS but also peripheral state is verified by comparing peripheral configuration registers prior to a checkpoint and post restoration.

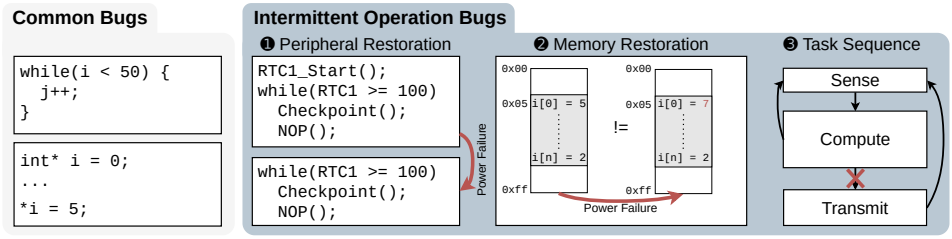


Figure 3.2: Two major classes of code bugs are present in intermittently-powered battery-free systems: common bugs such as using the wrong iterator or writing to null pointers and bugs related to intermittent operation. These include peripheral restoration, memory restoration and task sequence bugs.

All hardware, software and tools pertaining to DIPS, together with its documentation, will be made available open-source to the research community via our artifact [73]. We will also provide fully assembled and calibrated DIPS boards to the community. We envision DIPS becoming the de facto standard debugging tool for intermittently-powered systems, simplifying testing and debugging of these novel embedded systems.

3.2. DEBUGGING INTERMITTENTLY-POWERED SYSTEMS

Debugging of embedded systems code is different from PC-based code debugging [10, Chapter 8]. As PC-based code can mostly be directly debugged using tools such as GDB in an IDE as it runs on the same device, unlike embedded systems where the code is running on an external embedded system. Therefore, the developer must rely on external hardware—such as [263]—acting as an interface between GDB and the MCU on-board debug hardware.

3.2.1. BUGS TYPE CLASSIFICATION

We can categorise bugs present in the software for intermittently-powered devices into two classes presented in Figure 3.2. First class are the common programming language and embedded system-related bugs. Example of such bugs are algorithm implementation bugs (for example increment of the wrong variable in a while loop `while (i < 50){j++;}`) or code errors in embedded system-related functionalities (for example, inaccurate peripheral initialisation). These bugs are extensively analysed since the dawn of programming languages and will not be discussed here. The second class are the intermittent operation-related bugs and these are the ones which the designed debugger will specifically target.

We can further categorise intermittent operation bugs into: (i) peripheral restoration bugs, (ii) memory restoration bugs, and (iii) task sequence bugs (see again Figure 3.2).

1. **Peripheral restoration bugs** occur due to inaccurate reinitialization after checkpoint restoration, as seen in Figure 3.2. In this example: after the power failure the program restarts from within the while loop without reinitializing the peripheral causing an infinite loop from a not re-initiated Real Time Clock (RTC). Peripheral bugs also occur when the state of any external peripherals such as displays,

Listing 3.1 Masked Write After Read (WAR) error due to breakpoint insertion (listing (b)). When function calls are instrumented as checkpoints such as in [165], the addition of software-based breakpoints as required by the EDB [49] debugger can mask WAR-related errors, see listing (a), since the breakpoint itself will be instrumented with a checkpoint. `nv_x` refers to a variable stored in non-volatile memory.

(a) WAR error

```
1 Checkpoint()
2 y = nv_x // wrong
3 // after restart
4 z = y + 1
5
6 nv_x=z
7 # Power failure
8 ...
9 Checkpoint()
```

(b) Masked WAR error

```
1 Checkpoint()
2 y = nv_x // correct
3 // after restart
4 z = y + 1
5 EDB_Breakpoint(0)
6 nv_x = z
7 # Power failure
8 ...
9 Checkpoint()
```


sensors and radios is not carefully considered, especially when these peripherals have persistent state or are continuously powered. Examples of such bugs include: (i) persistent configuration registers where the process of configuring the register could not be confirmed due to a power failure, (ii) a failure to gracefully power down an E-Ink display resulting in a faded background, and (iii) synchronization issues where the external peripherals state is not aligned with the expected state.

2. **Memory restoration bugs** come from errors in the checkpoint process. In the first place they can come from the wrong placement of checkpoints (function `Checkpoint()`), as illustrated in Listing 3.1 (a) where a write after read error occurs due to the lack of a checkpoint in-between reading from and writing to non-volatile memory. But the implementation of a checkpoint itself can also contain bugs. For example, checkpointing is often based on double buffering (such as in [164] where the whole memory is checkpointed to a non-volatile memory at a predefined time interval), where usually a binary flag specifies to which memory region a checkpoint needs to be stored and from which region data needs to be restored. If a power failure happens at the moment of the flag update, the checkpoint will be corrupted. The more complicated checkpointing routines, like differential-checkpointing of [66] where the only changed memory regions since the last checkpoint are checkpointed, or undo logging-based checkpointing as used in [166]—the higher the probability of error in the implementation of checkpoint.

3. **Task sequence bugs** are specific to special type of run-time systems for intermittently-powered devices where input code is transformed into tasks (such as 'Sense', 'Compute' and 'Transmit') and checkpointing is performed always at the task transition. Examples of such systems include InK [325], Alpaca [189] or ImmortalThreads [326]. Bugs can not only occur with incorrect implementation of the task state machine (as in case of example in Figure 3.2 'Compute' task connects back to 'Sense' instead of 'Transmit'). Bugs can also occur when defining the volatile memory associated with each task—if not accurately defined it could result in writing to and reading from unrestored memory.

Table 3.1: Feature comparison of DIPS (i.e. this work) against EDB [49]—debugger for intermittently-powered battery-free embedded systems and J-link [263]—popular debugger for battery-based embedded systems.

Feature	EDB	J-Link	DIPS
Energy breakpoints	Yes ✓	No ✗	Yes ✓
Software breakpoints	Yes ✓	Yes ✓	Yes ✓
Hardware breakpoints	No ✗	Yes ✓	Yes ✓
Single step	No ✗	Yes ✓	Yes ✓
Watchpoints	Yes ✓	Yes ✓	Yes ✓
GDB support	No ✗	Yes ✓	Yes ✓
IDE support	No ✗	Yes ✓	Yes ✓
ARM support	No ✗	Yes ✓	Yes ✓
MSP430 support	Yes ✓	No ✗	Pending
Software testing	No ✗	No ✗	Yes ✓
Energy trace emulation	No ✗	No ✗	Yes ✓

3.2.2. WHY DEBUGGING INTERMITTENTLY-POWERED SYSTEMS IS HARD

We need a dedicated debugger that would aid in spotting all errors shown in Figure 3.2 and EDB [49] was the first one that addressed this need. EDB introduced new debugging features, as listed in Table 3.1. Sadly, in special cases EDB can hinder bug finding. As we mentioned in Section 3.1 EDB debugger [49] inspects the code by inserting software based assertion flags and breakpoints (to trace potential intermittent operation-related bugs). These however might mask the write after read bugs, as shown Listing 3.1 (b), as software breakpoints are implemented as a function this might interact with compiler-based runtime systems [165] that apply compiler based optimizations and instrument each function. A software breakpoint (`EDB_Breakpoint(0)`) would then result in an undesired checkpoint on the breakpoint location masking the write after read issue. A release build without the software debugging functions would then re-expose the hidden 'write after read bug'. Apart from introducing potentially unwanted checkpoints, software debugging calls could also prevent further compiler optimization such as loop unrolling.

For the record, one can consider using a CPU emulator, such as [227] as used in [165], as a replacement for EDB's inability to fulfill its debugging task completely. However, the timing of the instructions is not always perfect in emulation. Most importantly however, in most emulators the peripheral state is mocked—disallowing detection of peripheral-related bugs. Even with a cycle accurate emulation of the CPU and the associated peripherals forming the complete MCU, emulators do not emulate the starting sequence that occurs when power is applied to the actual chip. The CPU only starts execution after, e.g., voltage rails stabilize and stable clocks are present. These processes determine the start-up time and are subject to per component/design variation.

We thus conjecture that a debugger for intermittently-powered systems needs to have the same list of functionalities as debuggers for 'classical' embedded systems, e.g. [263], which are listed in Table 3.1. Moreover, it needs to support intermittently-powered systems specific features, which we denote as *energy breakpoints*, *software testing* and *energy trace emulation*. Inspecting Table 3.1 neither EDB, nor J-Link supports complete set of debugging features needed.

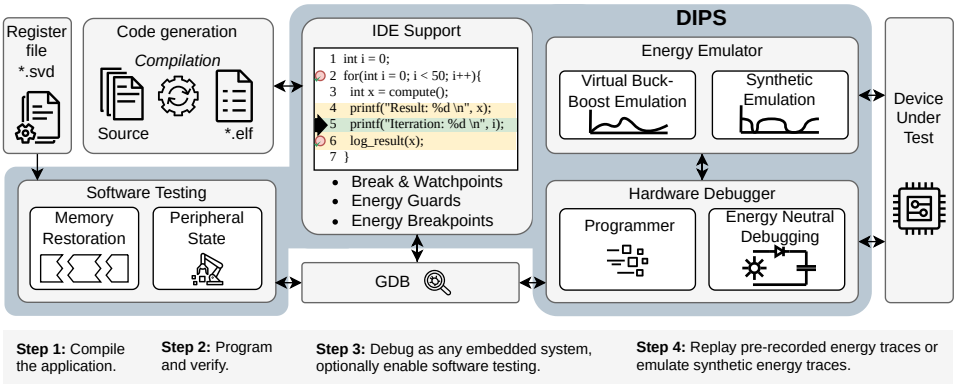


Figure 3.3: DIPS architecture and workflow, enabling seamless debugging of intermittent systems. In the code view marks an energy neutral section, marks the current line and hardware breakpoints are indicated by . Arrows in the figure denote information flow between individual blocks.

3.3. DEBUGGER FOR INTERMITTENTLY-POWERED SYSTEMS

Driven by requirements listed in Table 3.1 we propose a new method of developing and testing battery-free intermittently-powered devices. These development and testing methods are implemented as a new debugger named DIPS. DIPS combines a hardware debugger (described in Section 3.3.1) and an energy emulator (described in Section 3.3.2), enabling seamless debugging of intermittently-powered systems. The energy emulator acts as a controllable power source capable of emulating intermittent operation to the Device Under Test (DUT). The architecture of DIPS is shown in Figure 3.3.

3.3.1. DIPS HARDWARE DEBUGGER

A core part of debugging any embedded system is the hardware debugger. It interfaces with the DUT's MCU enabling the use of the MCU's debugging features. These features usually include (i) halting, (ii) reading and writing memory, (iii) setting breakpoints, and (iv) setting watchpoints.

As intermittently-powered systems switch on and off repeatedly, any state that is not specifically stored prior to a power failure is lost. This includes the configuration of the debugging registers. Even worse, these debugging registers are usually not configurable from within the MCU itself due to the security risk associated. Hence the hardware debugger must be able to quickly reconnect after power failures on intermittently-powered systems. To address this requirement DIPS keeps track of all debugging attributes, such as breakpoints and watchpoints, restoring those when the MCUs recovers from the power failure. To implement these features we have taken a popular open-source hardware debugger—the Black Magic Debug Probe [228]—as a base and build upon its functionality, adding the required features to debug and test intermittently-powered systems. Many popular MCUs are supported—for a full list please refer again to [228].

ENERGY NEUTRAL DEBUGGING

One core feature of DIPS is the energy isolated interface between DUT and DIPS. This allows DIPS to monitor the DUT whilst not interfering with the power consumption of the DUT. If the DUT is paused by any debugging action e.g., a breakpoint, the hardware debugger automatically pauses the energy emulator, making sure the DUT remains powered. When execution is resumed the energy emulator restores the energy state prior to the breakpoint and continues from where it paused.

We introduce two debugging modes with DIPS, (i) attached and (ii) detached, where each of them is described below. The hardware implementation of the energy isolation is further described in Section 3.3.4.

Attached Debugging. In the attached debugging mode, the debugger reconnects to DUT after every power failure and any debugging attributes such as breakpoints are restored. When the DUT is connected to the hardware debugger, additional power will be consumed by the MCUs on-board debugging hardware. This is compensated for during emulation by measuring the power consumption at idle with and without the debugger attached. The attached mode gives most flexibility to the user as the intermittently-powered system appears as a normal embedded system to the developer, masking any effects of intermittency. We envision this mode to be used in a scenario of active software development and during preliminary testing/evaluation of intermittent systems. For example, during development of new checkpoint frameworks or when testing if peripheral configuration is correctly restored after power failures.

Detached Debugging. This mode is intended for when the DUT powers itself, for example by an external harvested energy source. In this mode the debugger only connects when it receives the hardware interrupt from the DUT, generated by e.g. the `DIPS_ATTACH` call to the C API listed in Table 3.2, if not already connected. When an interrupt is generated, the emulator takes over powering DUT. Next the debugger connects, allowing the debugger to interact with the DUT. After the user resumes code execution or when the debug operation finishes, the debugger detaches and the energy state of the DUT is restored to the level prior to intervention as further described in Section 3.3.2. This mode is intended for debugging scenario's close to final deployment where minimal interference is desired. More specifically, in scenarios where the device operates on its own, whilst still offering an option to debug the system when, for example, an assert fails.

ENERGY-AWARE DEBUGGING FEATURES

Most of DIPS's debugging features utilize the build-in MCUs debugging hardware. Thus unlike the software-based debuggers DIPS does not require the usage of a specific API to debug the DUT. We extend GDB with the CLI commands listed in Table 3.2 to implement two key intermittent specific debugging functions: (i) energy breakpoints (`energy_breakpoint`)—extending traditional breakpoints by only triggering when the DUT's capacitor voltage is lower than the provided threshold, and (ii) energy neutral sections defined by energy guards (`energy_guard`)—allowing users to execute debug code whilst emulation is paused and steady power is provided.

Apart from the GDB extensions, we introduce a limited C API listed in Table 3.2, providing some optional convenience functions to the user. The function `DIPS_PRINTF` implements a print to the console. When `DIPS_ASSERT` parameters assert to false, code

Table 3.2: DIPS extensions to the GDB Command Line Interface (CLI) implementing debugging functionality for intermittently-powered devices and an optional C language API for quick and simple debugging of intermittently-powered systems. An extended description is provided in [73].

GDB CLI	Description
`energy_breakpoint`	Defines a voltage-dependant breakpoint
`energy_guard`	Defines an energy neutral section
C API	**Description**
`DIPS_PRINTF`	Energy-neutral `printf`
`DIPS_ASSERT`	Halts code execution upon assertion
`DIPS_ATTACH`	Connect debugger (Detached mode)

execution is halted. `DIPS_ATTACH` triggers the debugger to connect and halts execution until the debugger is connected.

All hardware debugging features and C API calls cause the emulator to pause whilst keeping the DUT powered. When normal code execution is resumed, the emulator also resumes. Compensating for the power consumption of the debug features itself.

Programmer. DIPS is also capable of programming/flashing of supported MCUs. Although generic support for debugging ARM Cortex-M chips is provided, programming might require additional vendor-specific or even chip-specific implementations. For a list of supported MCUs please refer again to [228]. The programming/flashing feature completes the all-in-one suite of features served by DIPS for the developer of intermittently-powered systems.

3.3.2. DIPS ENERGY EMULATOR

The second core part of our design is the energy trace emulator. State-of-the-art intermittently-powered systems harvest energy and store this energy into a (super-)capacitor. The voltage of this capacitor is often used as a threshold to determine when the voltage regulator powering the MCUs turns on and off. In this case the MCU only is provided a regulated supply that is switched on and off according to the voltage of the (super-)capacitor. Often the harvesting and regulator circuit is implemented using a boost converter and buck converter to generate the MCU supply stepping down the voltage of the (super)capacitor.

Unlike other emulation platforms such as Ekho [112] and Shepherd [96] we emulate the buck-boost converter and the storage capacitor, and directly provide the resulting on/off output to the DUT. We do not aim to fully replicate the buck-boost converter but to attain similar behavior with a simplified model. This approach only requires a voltage/current input trace, greatly simplifying capacitor size selection and makes it capable of simulating any buck-boost converter that outputs a steady supply by adjusting the converters specific parameters such as efficiency.

The emulator is implemented as configurable power supply and is able to quickly switch off/on its supply to DUT. It is also capable of accurately measuring the power consumption of the DUT, as depicted in Figure 3.4.

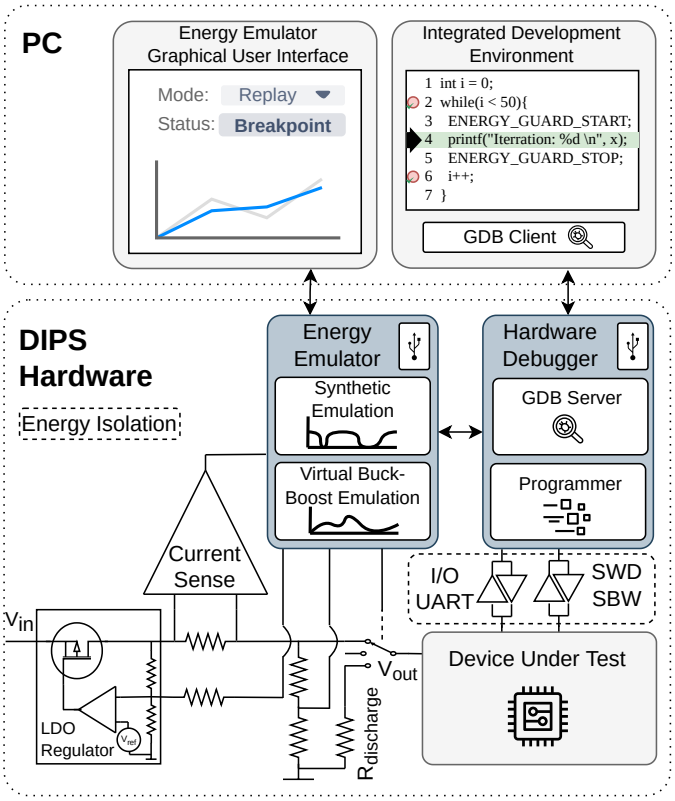


Figure 3.4: DIPS simplified implementation overview of both hardware and software. The full hardware and software implementation of DIPS can be found in DIPS's artifact [73].

Virtual Buck-Boost Emulation. We have chosen to emulate the Texas Instruments' BQ25570 ultra low power harvester power management IC [292], as it is one of the most frequently used buck-boost converters in intermittently-powered systems. Our approach is centered around the current voltage relation of a capacitor defined as

$$V_{\text{cap}}(t) = \frac{1}{C}\int_{t_0}^{t} I(\tau)d\tau + V_{\text{cap}}(t_0), \tag{3.1}$$

where $V_{\text{cap}}(t)$ is the capacitor voltage, C the capacitance, $I(\tau)$ is the *instantaneous* current flowing into (i.e. harvested) and out (i.e. consumed) of the capacitor, and $V_{\text{cap}}(t_0)$ is the initial capacitor voltage at $t = 0$. Using a look up table we compensate the input current according to the efficiency of the boost converter in the BQ25570 according to the input voltage. The outgoing current is the current measured by the emulator which is also compensated by the efficiency of the buck converter. Adding thresholds for turning the output on (V_{high}) and off (V_{low}), over-voltage protection (V_{max}) and compensating for leakage and quiescent current completes our simplified model. If the hardware debugger is running in attached mode, the additional quiescent current of the MCUs debug hardware is

also compensated for as mentioned in Section 3.3.1. Our emulator is capable of operating with a static input current and with replaying pre-recorded voltage/current input traces. The emulator is compatible with Shepherd's [96] Hierarchical Data Format (HDF) traces.

Synthetic Emulation Modes. Apart from operating as a virtual buck-boost converter, our emulator is also able to generate arbitrary signals. These signals include square wave and sawtooth modes with adjustable frequency and duty cycle. Synthetic emulation is needed to stress test any intermittently-powered device.

HARDWARE DEBUGGER INTEGRATION

If the energy emulator is actively powering the DUT when a debugging feature is triggered such as a breakpoint, emulation is paused whilst keeping the DUT powered. When execution is resumed, emulation also resumes. Any calls to the DIPS API also pauses emulation until completed. The energy emulator also implements a passive mode compatible with the debuggers detached mode, intended for a scenario of debugging an intermittently-powered system operating using its own (harvested) energy supply. In this mode when a DIPS API call occurs, the DUT voltage is first sampled. Then the emulator supplies a safe slightly higher voltage than the system voltage to the DUT—taking over and powering the DUT until the debugging action is completed. Then the original voltage of the DUT is restored. This mode should be used with care as back-feeding could occur.

PC CLIENT SOFTWARE ARCHITECTURE

To control, configure and monitor the emulator we have designed a PC Graphical User Interface (GUI) client build around the QT framework [244]. The client communicates with DIPS through USB using an extendable Protobuf [105] interface. When the emulator is connected to the PC, the client automatically attempts to connect to DIPS. Through the Protobuf interface the client is able to select and configure the emulation modes. For example, in square wave mode (i) the duty cycle, (ii) period and (iii) voltage are configurable. One notable option specific to the virtual buck-boost mode is to stream HDF voltage/current input traces to the emulator for replay. The emulator also has an option to stream the measured voltage and current of the DUT to the client, where this data is visualized by an interactive chart. Finally, a status indicator is present in the GUI, indicating when emulation is paused by the hardware debugger.

3.3.3. DIPS AUTOMATED SOFTWARE TESTING

As DIPS integrates a hardware debugger it is able to provide full access to the memory space of the MCU under test. By leveraging GDB and its interpretation of the debugging symbols in the compiled code, DIPS is able to provide a full debugging context to the developer, including rendering of the call stack, variable values and all other default debugging features of normal embedded systems. Leveraging the emulator we are able to detect issues that are traditionally present in intermittently-powered systems. Central to the hardware debugger is the GDB server. Through the use of GDB many popular IDEs can directly integrate with DIPS. In the debug environment of the PC, a GDB client interfaces with the GDB server on the debugger. We utilize a transparent Python wrapper around the GDB client to extend the interface and automate specific testing for intermittent systems.

SOFTWARE TESTING SCRIPTS

Through the wrapper's extended interface we introduce two software testing scripts. The first script verifies checkpoint correctness by comparing the volatile memory of the DUT before the last checkpoint prior to a power failure and after restoration, at which point the memory should be identical. The second script compares peripheral configurations of the DUT prior to the checkpoint and after restoration, by comparing the relative configuration registers. Both scripts are designed to run in attached mode of the hardware debugger and are implemented in an extendable fashion so that more scripts could be added in the future.

Memory Restoration. To check memory restoration correctness through power failures, the user must specify the checkpointing function and the first function called after restoration when using the script. Additionally the volatile memory regions that should be checked need to be specified.

When running GDB with the script, the script automatically downloads and saves the specified memory ranges at every checkpoint. It then compares the latest stored memory against the memory after a restore. When a mismatch occurs, the symbolic name is retrieved through GDB of the offending memory address and the debugger remains in a breakpoint. The memory address together with its current contents and the content at the point of the latest checkpoint are then presented to the user for further investigation. The script also is optionally able to to monitor the time between checkpoints, if no checkpoints are made within a user definable time, code execution is halted for further investigation by the user. As an extended period without checkpoints on intermittent systems is often a good indicator for the DUT getting stuck.

Peripheral State Restoration. Since DIPS has full access to the address space, including the peripheral address space, we are also able to monitor peripheral configurations during checkpoints and verify if these are properly restored. In addition to specifying the checkpoint and restore functions, the user also needs to specify the configuration registers to be checked. Then based on the register name, the configuration registers addresses are retrieved by parsing the DUT MCU's .svd file—a .svd file that is commonly provided as part of a software development kit for MCU's. Again, prior to the checkpoint, the state of the peripheral configuration registers is retrieved and stored. Upon restoration the register state is compared against the stored state. Any issues are reported to the user and the debugger remains in a breakpoint.

3.3.4. DIPS HARDWARE IMPLEMENTATION

As described earlier DIPS is composed of two subsystems: (i) *the hardware debugger* and (ii) *energy emulator*. Both subsystems, shown in Figure 3.5 and marked by blue and orange polygon, respectively. The details of each subsystem hardware implementation are as follows.

HARDWARE DEBUGGER

The hardware design of the debugger centers around a STM32F103RET [280] MCU Ⓐ and is based on the Black Magic Debug Probe [228]. It is able to communicate with the energy emulator through SPI. The MCU interfaces with the DUT through SWD/JTAG or

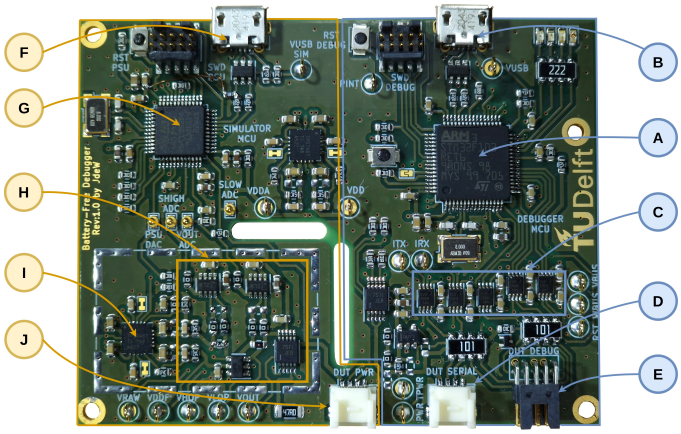


Figure 3.5: The DIPS hardware debugger and emulator PCB. The hardware debugger components marked as Ⓐ–Ⓔ and energy emulator components as Ⓕ–Ⓙ are explained in Section 3.3.4.

SBW Ⓔ, I/O interrupt pins and acts as a UART-USB bridge Ⓓ. To translate the signals to the DUT voltage, first, the DUT voltage is buffered using a low input bias current buffer amplifier OPA192 [134]. Next, all the interfaces with the DUT are level shifted by level translators Ⓒ [211] using the buffered DUT voltage. The debugger connects to the PC with USB Ⓑ.

ENERGY EMULATOR

Central to the energy emulator is the low noise TPS7A87 [135] Ⓘ linear regulator. The regulator generates the adjustable supply rail to the DUT Ⓙ. Power consumption by the DUT is measured by two INA186 current sense amplifiers [136] Ⓗ. The first amplifier with a 5.6 Ω sense resistor measures large currents without imposing a high burden voltage. The second amplifier with a 1000 Ω sense resistor measures low currents and is able to be bypassed at large currents preventing high burden voltages. Two analog switches [133] allow for quickly disabling the output and discharging the output through a 47 Ω resistor. The emulator is controlled by a STM32F373 [279] MCU Ⓖ. With its on-board DAC DIPS is able to adjust the linear regulator and samples the output voltage and the output of the current sense amplifiers (each using one of its dedicated on-board Sigma Delta ADCs). The energy emulator connects to the PC with USB Ⓕ.

3.4. DIPS EVALUATION

We now proceed with the evaluation of DIPS. The evaluation is split in three parts. First, we characterise DIPS. Second, we perform user studies aiding in finding whether DIPS is a useful (and better then state of the art) tool for debugging battery free systems. Finally, we show how DIPS can be used to find bugs in recently presented battery-free intermittently-powered systems.

Table 3.3: DIPS energy emulator specification. Measurements were performed with a Keithley 2450 Source Measurement Unit [154] and a Saleae Logic Pro 8 [256].

Feature	Parameter	Specification
Replay Resolution		1 ms
Sampling rate	Voltage	50 kHz
	Current	50 kHz
Range	Voltage	0.1 V–3.6 V
	Current	1 µA–20 mA
Accuracy	Voltage	±50 mV
	Current (1 µA–100 µA)	5 % ±1 µA
	Current (100 µA–20 mA)	5 %
Rise Time	0–3 V (Switch)	28 µs
	0.1–3 V (Adjust)	836 µs

3.4.1. DIPS CHARACTERIZATION

To evaluate DIPS we conduct several measurements to evaluate the performance of our debugger. These measurements are divided into two categories: (i) the specification of the energy emulator and (ii) characterization of the hardware debugger.

Energy Emulator Characterization. In Table 3.3 a specification of DIPS' energy emulator is provided. Notable attributes are the fast sample rate, wide current measurement range capability and quick rise time. These attributes enable DIPS to accurately emulate the simplified buck-boost converter behaviour using real-world energy traces as input; the 1 ms resolution enables dynamic scenarios emulating abrupt energy changes at DUT. For other synthetic operation modes such as sawtooth or square wave, voltage accuracy is crucial to trigger voltage-based thresholds for the DUT.

Hardware Debugger Characterization. An overhead of DIPS' hardware debugger operating in attached mode is the requirement of establishing a connection to the debug hardware of the DUT. This can occur prior to starting execution after a reboot, or when the system is running. When the hardware debugger connects whilst the device is running, early breakpoints might be missed. When this is unacceptable, `DIPS_ATTACH` can be placed at the start of the program. The hardware debugger then connects prior to any code execution at the cost of a slight delay. The time required to establish a connection is listed in Table 3.4.

3.4.2. DIPS USER EXPERIENCE STUDY

To assess the effectiveness of bug finding in code written for intermittently-powered systems, we have designed a user experience study. In this study, participants were asked to experiment with DIPS and EDB [49]—the state-of-the-art debugger for intermittently-powered systems. In particular, we asked to search for three bugs in a single simple program consisting of multiple files (written separately for both debugging platforms, containing bugs of similar complexity—DIPS and EDB) using two respective debuggers. After the bug search process participants were asked to assess their debugging experience

Table 3.4: DIPS debugger characterization: t_{init} (initial connection time) and t_{rec} (re-connection time) while connected to different devices. Data points were collected using a Saleae Logic Pro 8 [256], and averaged over ten measurements.

Device Under Test	t_{init} (ms)	t_{rec} (ms)
nRF52 [Arm-M4] [217]	311.1	72.7
SAM4L8 [Arm-M4] [201]	324.7	75.8
MKL05Z [Arm-M0+] [224]	309.6	105.8
STM32F3 [Arm M4] [279]	318.6	68.2
Apollo 3 [Arm M4] [277]	331.1	95.6

with each platform through an anonymous survey. The study was approved by the human ethics committee of the institution the authors of this work are associated with.

We have performed two versions of experience studies: (i) a pre-study (denoted as *Study 1*) with small number of participants, with limited time given to find bugs in each program and (ii) main study (denoted as *Study 2*), with twice the size of the user pool of the *Study 2* and with double the time allowed to find bugs in each program.

USER EXPERIENCE STUDY PARTICIPANTS

We have invited seven participants to *Study 1* and 16 participants to *Study 2*. Participants were recruited through professional mailing lists and personal contacts. Special care was taken of not recruiting people that are in a current or former relation with the responsible persons for this study.

Based on the anonymous post-study online self-assessment survey, among all study participants the following information was found: *Study 1's* participants included six men and one woman and for *Study 2* fourteen men and two women. The median age of participants was 26 (youngest: 23, oldest: 44) for *Study 1*, and 26,5 (youngest: 20, oldest: 36) for *Study 2*. The most comfortable programming language in which participants code was C/C++ (four participants in both studies) followed by Python (three participants in *Study 1* and four participants in *Study 2*). All participants in *Study 1* and all but one in *Study 2* have used an IDE before when developing their applications and all but one participant from *Study 1* and five out of 16 participants from *Study 2* preferred to develop their applications using an IDE. In *Study 1* and *Study 2*, respectively: four and three participants self-assessed themselves as having a lot embedded programming skills, two and six—some experience, one and five—little experience, and none and two—no experience. Large majority (i.e. five) participants used hardware-based debuggers for their embedded project (such as Segger J-link [263]) among *Study 1* participants, while only 5 out of 16 for *Study 2*.

All participants used at least one of the following debugging techniques while debugging an embedded system, such as breakpoints, watchpoints, memory views, peripheral views, etc. of *Study 1*, while for *Study 2* 11 out of 16 participants were exposed to these debugging techniques. The most used debugging techniques by participants were breakpoints (six and ten participants of *Study 1* and *Study 2*, respectively), `printf` (mentioned by six participants of *Study 1* and six participants of *Study 2*), single step (three participants of *Study 1* and nine participants of *Study 2*) and memory inspection (mentioned by

three participants of *Study 1* and three participants of *Study 2*) and 'measuring voltage' (mentioned by one participant of *Study 1*).

Only two participants did not hear about battery-free intermittently powered systems before the start of *Study 1*, while only two out of 16 participants of *Study 2* heard about such systems, while just one participant of *Study 2* programmed them before. Based on a Likert scale, the question 'how difficult is the application development for battery-free intermittently-powered systems?' gave the following responses: five participants of the *Study 1* and 13 of *Study 2* answered 'somewhat difficult' and two participants of *Study 1* and three of *Study 2* answered 'similar to battery-powered embedded systems'. For the record, nobody found them either 'very difficult', 'easy' or 'very easy' to program.

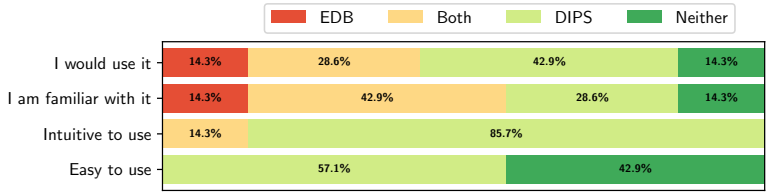
Based on the above information we can conclude that user experience study participants were skilled (considering their background and experience) and well informed to take part in this user experience study.

USER EXPERIENCE STUDY SETUP

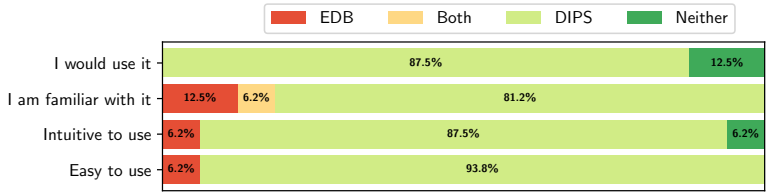
We asked each participant to enter a room with two pre-configured PCs. One PC was connected to DIPS that in turn connected to a Nordic Semiconductors NRF52 development kit [217]. Another PC was connected to EDB, where EDB was connected to a Wireless Identification and Sensing Platform (WISP) [257] (version 5.1). Both platforms were powered from a DIPS emulator configured either as constant voltage or square wave supply simulating intermittency. This emulation was not part of the user study but required in order to power the devices under test. Both PCs had VSCode [55] as a code editor and all requisite tooling to compile and use the debuggers installed. The PC with DIPS-only had the IDE open, while the other PC also displayed a console environment with the EDB program and means to recompile the code.

Before the debugging session started each participant was requested to read a short description on intermittently-powered systems and to read an instruction how to use DIPS and EDB.[1] After reading the instruction, the participant was asked to debug a piece of code for one of the systems (with which debugger system the user starts the study was randomly determined for each participant). After 15 minutes of debugging in case of *Study 1* and 30 minutes in the case of *Study 2* the participant was asked to fill-in the survey with a questionnaire regarding the debugging experience. This survey section was enabled only when the participant asked for a password—this reduced the chance that the participant would answer survey questions without first debugging with the system. The same process (bug finding and password-protected survey fill-in) was repeated for the second debugger. During the survey neither DIPS nor EDB was mentioned and both PCs were referred to as 'System A' and 'System B' with name cards attached to the PC's monitor—to remove any bias in assessing both systems and not reveal which system originates from the institution with which all experience study participants were associated with. We note that the questions in the questionnaire were given as comparative (i.e. how system A performed against system B).

[1] The exact text given to the participants, with the code given for debugging for both systems, together with the user experience study results, is available as part of the open-source repository of DIPS [73].



(a) *Study 1*, i.e pre-study



(b) *Study 2*, i.e main study

Figure 3.6: Responses to questions given to user experience study participants after the completion of bug finding sessions for DIPS and EDB. Note that numbers in this figure and in Figure 3.7 are rounded to the nearest decimal digit.

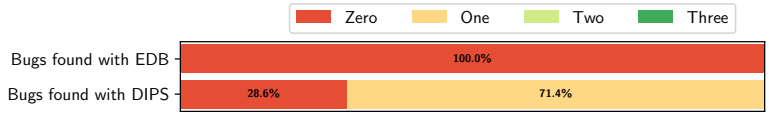
USER EXPERIENCE STUDY RESULTS—PRE-STUDY

The first result of the user experience pre-study (*Study 1*) is presented in Figure 3.6a. We see that more users would use DIPS than EDB. Moreover, a large majority of users found DIPS more intuitive to use than EDB. Only a small minority of participants would use EDB instead of DIPS. No participants stated that they are only familiar with the way that the EDB debugging process works—the majority of them are familiar with the 'regular' way programs are debugged. All these results hint that DIPS suits debugging tasks of intermittently-powered devices better than the state-of-the-art system.

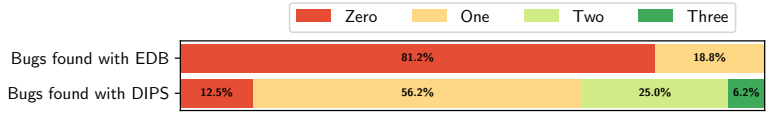
The results of the bug session finding are presented in Figure 3.7a. One surprising result is that nobody was able to find any bug with EDB, while majority the participants were able to localise at least one bug with DIPS. We speculate that such extremely low bug finding rate for EDB (and inability to localize two or more bugs with DIPS) was due to insufficient time allocated to find all bugs in a session. On the other hand, a short time for bug finding was a stress test for both systems, suggesting that DIPS is more useful in code debugging compared to EDB (even for complex and still unexplored systems such as intermittently-powered devices). With the main study (*Study 2*), with more time allocated to debugging, we shall find whether this extra time would result in significantly better perception of EDB. The results are presented in the next section.

USER EXPERIENCE STUDY RESULTS—MAIN STUDY

The results of the main study (*Study 2*) are presented in Figure 3.6b and Figure 3.7b. Comparing them with Figure 3.6a and Figure 3.7a, respectively, we can conclude that the increased time to find bugs, from 15 minutes to 30 minutes per debugging session, did not significantly affect the perception of which system is better (Figure 3.6b). Actually, the main study shows that participants are more positive about DIPS than about EDB,



(a) *Study 1*, i.e pre-study



(b) *Study 2*, i.e. main study

Figure 3.7: Number of bugs found by users participating in both studies, categorized per debugger system.

as less participants were pointing to EDB or pointing to both debugging systems (Figure 3.6b). Most importantly, however, the additional time assigned to the participants of the user study resulted in more bugs to be found with EDB, but also *more* bugs with DIPS (Figure 3.7b).

GENERIC OBSERVATIONS BY STUDY PARTICIPANTS

In addition to closed questions given to the participants, which results are presented in Figure 3.6a and Figure 3.7a for *Study 1* and in Figure 3.6b and Figure 3.7b for *Study 2*, we have asked four open-ended questions asking to specify positive and negative aspects of DIPS and EDB, respectively. Considering EDB, the positive aspects listed were as follows: it suits those developers better who prefer terminals over GUIs allowing for scripting and automation (which, on the other hand, other study participants found as negative for developers used to GUI-driven development[2]); one person found ability to set breakpoints and seeing the capacitor voltage as valuable. The negative aspects of EDB listed were as follows: not intuitive as a whole and having non-intuitive commands; forcing to re-flash code for breakpoints; being erroneous when debugging and has no ability to resolve symbols. Conflicting points were listed however—one person described EDB as 'programmer friendly' while other found no positive aspects of EDB.

Considering DIPS, the positive aspects listed were: very similar to existing debuggers—"people will have an easier time learning how to use [it]"—being able to use already-accustomed GUI debugging buttons; being "tightly integrated to IDE" and being able to "set breakpoints in editor"; no need to compile code for every debug session. The negative aspects of DIPS listed: two users were expecting even more user-friendly system (not requiring to "switching between tabs for building/loading" whereas "restart button doesn't seem to function correctly"); one user still found DIPS difficult to use (who nota bene made the same remark about the EDB).

As an overarching conclusion stemming from all questions posed to the participants—users found DIPS *easier to use, more intuitive and more familiar* than EDB.

[2]We speculate that due to the setup of the user study participants thought that DIPS was developed to be integrated only with IDE. However, we note that DIPS can also be used as stand-alone debugger in the console.

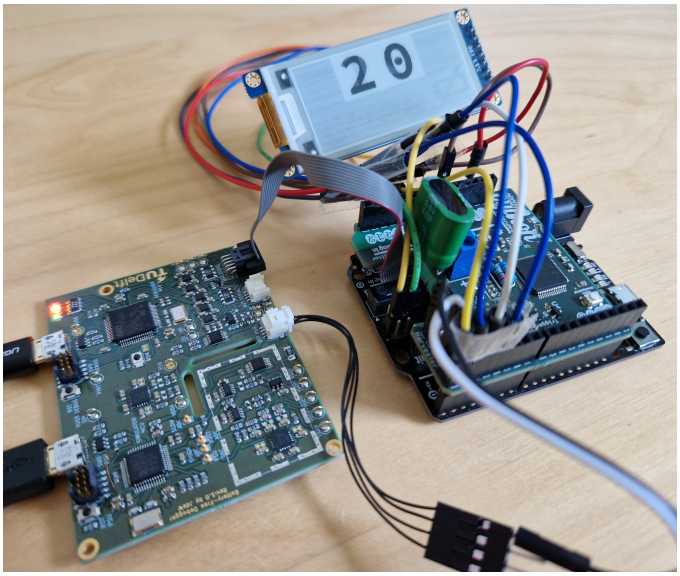


Figure 3.8: DIPS connected to BFree [164] aiding in finding BFree's peripheral bug. The bug causes the background of the E-Ink display to be almost completely faded.

3.4.3. SOFTWARE TESTING WITH DIPS

Next, we show how DIPS helps in finding bugs in existing intermittently powered systems. We will demonstrate this ability of finding bugs using DIPS and its features based on two case studies.

CASE STUDY: BFREE—PERIPHERAL BUG

Experimental Setup. We start with a state-of-the-art intermittently powered system, BFree [164], as the selected DUT with the E-Ink application programmed. We have connected DIPS to the DUT, as shown in Figure 3.8 and powered the system using DIPS's energy emulator in a square wave mode.

Symptoms. When inducing frequent power failures to the BFree, the background of the E-Ink display fades. This fading seems to occur when BFree experiences a power failure during an update of the screen.

Diagnosis. First to rule out any obvious issues we ran DIPS's software testing memory restoration script for 15 minutes. During testing no discrepancy was detected between BFree's volatile memory prior to checkpointing and after restoration. The peripheral state script of DIPS also did not find any related mismatches in BFree's peripheral register configuration.

In Listing 3.2 the BFree code is shown that partially updates the display. Checkpoints are temporarily disabled during the execution of this function (see line 5 and 12 in Listing 3.2), meaning the code has to be executed in full without any power failures. However, if a power failure would occur in-between line 5 and 12, the peripheral could be

Listing 3.2 BFree code snippet partially updating E-Ink display.

```
1  void epd_draw_temp(uint8_t temp) {
2      uint8_t unit = temp % 10;
3      uint8_t tens = (temp/10) % 10;
4
5      checkpoint_disable();
6
7      epd_init_temp();
8      epd_font_show(0, 2, unit);
9      epd_font_show(0, 1, tens);
10     epd_deep_sleep();
11
12     checkpoint_enable();
13 }
```


left with an unknown state. When BFree restarts, BFree will resume at the last checkpoint and the code will be executed again from that checkpoint.

By single stepping through this code of BFree using the DIPS debugger and forcing a power failure at each code line, we discovered that this partial execution causes the fading of the display. That is, if a power failure occurs between `epd_font_show()` and `epd_deep_sleep()`, the state of the display is not "locked" and remains in a high voltage state causing the fading, potentially damaging the E-Ink display over time. This issue could be resolved by for example, checking if enough energy is present before starting the screen update or to gracefully power down the BFree peripherals when a power failure is immanent.

In this case study DIPS assisted by quickly finding a major problem with the DUT, i.e. a peripheral-related bug[3]. Using the ability of DIPS to single step through code and the ability to generate power failure patterns using DIPS's energy emulator, we quickly identified the underlying issue within BFree.

CASE STUDY: ENGAGE—MEMORY RESTORATION BUG

Experimental Setup. As a second case study we chose Engage, the system used in the Battery-Free Game Boy [66]—a battery-free intermittently-powered handheld gaming console, as the selected DUT. Engage uses an optimized version of checkpoining, where only the memory regions that were changed since the last checkpoint (denoted as patches) are stored in a non-volatile memory at each new checkpoint. We have connected DIPS to the DUT, as shown in Figure 3.9 and powered Engage using the energy emulator of DIPS in square wave mode.

Symptoms. After an extended period of time when power failures are induced frequently to Engage, no game content is displayed on the screen when powered and Engage appears

[3] DIPS' software testing scripts do not test the state of external peripherals connected to the DUT—only the configuration of the MCU's peripherals of the DUT.

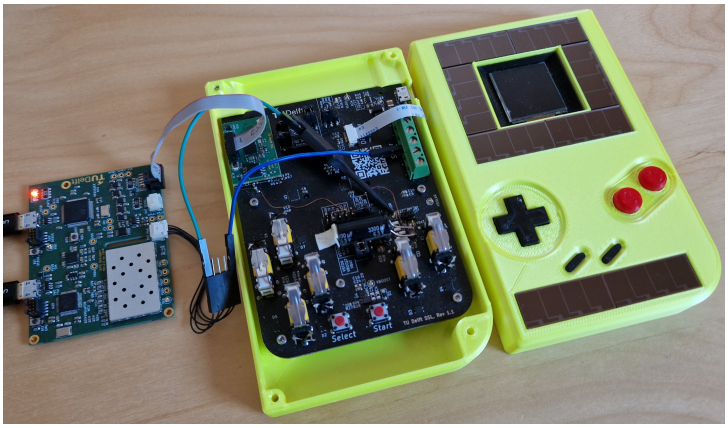


Figure 3.9: DIPS connected to Engage [66] aiding in finding a memory restoration bug. After a certain time of continuous intermittent execution a checkpoint of Engage is corrupted, resulting in the handheld console failing to start.

not being able to start (i.e., only black screen is seen instead of game content). After this failure has occurred—even when continuous power is supplied to Engage—Engage fails to start the game it was intended to play.

Diagnosis. The symptoms hint at a memory restoration issue, where either Engage's memory gets corrupted or something is preventing the Engage to boot. First, we ran our software testing memory restoration script for 15 minutes, verifying that the volatile memory is correctly checkpointed and restored. Whilst running the test we did not detect any discrepancies in Engage's memory. Then, by running the peripheral state script for another 15 minutes we ruled out any inconsistencies between peripheral configuration that could prevent Engage from, for example, accessing the external non-volatile FRAM where the checkpoints are stored.

As these 15 minute long tests were unable to reproduce the symptoms we have extended the testing time. After extended testing using the memory restoration script, DIPS paused code execution. This pause was triggered as no checkpoint has occurred within the predefined time window of five minutes, hinting that most likely no forward progress has been made. At this point Engage exhibited the previously mentioned symptoms.

Engage's execution was paused by the hardware debugger of DIPS at the moment of the restoration process, i.e. where memory is restored from a chain of memory patches. By further manual investigation with DIPS by breakpoints and stepping through the code we deduced that the process of applying the patches could not finish and formed an infinite loop preventing the system from starting.

In this case study DIPS assisted by quickly ruling out major problems. The description of the process (from unsuccessful 15 minute tests to a successful automated test) shows the usefulness of DIPS in directly pointing the developer to the issue (which preventing Engage from starting).

3.5. LIMITATIONS AND FUTURE WORK

Despite the advantages DIPS is bringing in debugging intermittently-powered systems, the research on debugging platforms for such intermittently-powered systems is not over. We list the most important limitations of DIPS, with its current study below.

Support for non-ARM MCU Architectures. DIPS does not yet support of debugging of non-ARM MCU architectures, refer again to Table 3.1. One particular MCU series that requires immediate support from DIPS is Texas Instruments' MSP430 MCUs [296]—used in numerous previous projects on intermittently-powered systems, including [325, 189, 113]. Luckily there are no technical limitations that would disallow to support MSP430 by DIPS. DIPS' support for MSP430 and its implementation is further described in [73, DIPS Support for MSP430].

Per-Line Code Inspection. What DIPS currently cannot do is to point to the exact code line that caused the program error. Such per-line code inspection for intermittently-powered systems was presented in [192], where WAR dependencies are found using code analysis. Then at each of these dependencies a power interrupt was emulated and memory regions were inspected for any inconsistencies. With additional scripting, DIPS could single step through the code and generate a power failure at every potential WAR, this method however, will be significantly slower than simulation based methods.

Further Development of DIPS. The overarching aim of the DIPS project is to be *useful* to the developers working on intermittently-powered systems. This can only be achieved by the introduction of new functionalities and support for new platforms, such as including MSP430 MCUs [296] support as mentioned above. Since we make DIPS available to the wider community, additional features, further improvements and evaluation can be contributed to the project by the community itself.

3.6. RELATED WORK

The field of testing intermittently-powered embedded systems can be categorised into (i) energy trace generation—for harvested energy trace replay and synthesis, (ii) testbeds—for controllable performance assessment of battery-free systems, and (iii) debugging systems (both hardware and software)—for finding individual errors in the code. For the record, a high-level introduction to embedded systems testing (thus also conventional battery-based systems) can be found in [23].

Energy Trace Generation. Considering the first category, Ekho [112] is a platform capable of replaying pre-recorded current/voltage traces that targets energy-harvesting devices. Such platforms aid in providing repeatable conditions during testing and could operate as a stress test by feeding different power supply traces to the DUT, e.g. to see at which conditions DUT stops working. Energy trace generation is also an integral feature of DIPS.

Battery-Free Systems Testbeds. Considering the second category, Shepherd [96] is the first (and the only one, to the best of our knowledge) complete battery-free intermittently-powered testbed[4]. It extends the energy trace recording and replay features introduced by

[4] For the record, the first idea of such testbed in a preliminary form was presented in [1].

Ekho [112], allowing to replay energy traces simultaneously and in synchrony for different spatially-separated sensors.

Hardware and Software Debugging Systems. Considering the third category, the reference point for DIPS (and the only available debugger for intermittently-powered systems) is EDB [49], which has already been discussed extensively in this chapter. To the best of our knowledge, the systems that target software-only techniques of bug finding in intermittently-powered systems are [192, 285]. Please note however that [192] does not work on a real embedded system, so memory region and peripheral inspection of the actual DUT is impossible. Then, work of [285] considered the problem of bugs caused by I/O operations of intermittently-powered devices, which was addressed by the static code analysis and dynamic information flow tracking to detect bugs at runtime. However, the bug detection of [285] needs (i) code instrumentation, (ii) targets a specific framework for intermittently-powered systems (i.e. task-based system [189]), and (iii) requires complete code compilation for each new memory inspection. Another example of static analysis tool for task-based programs is CleanCut [52]. CleanCut optimizes placing of task boundaries to reduce task-specific bugs, i.e. non-terminating tasks due to too few task boundaries. Still, unlike DIPS, CleanCut does not allow for real code debugging of the resulting transformed code.

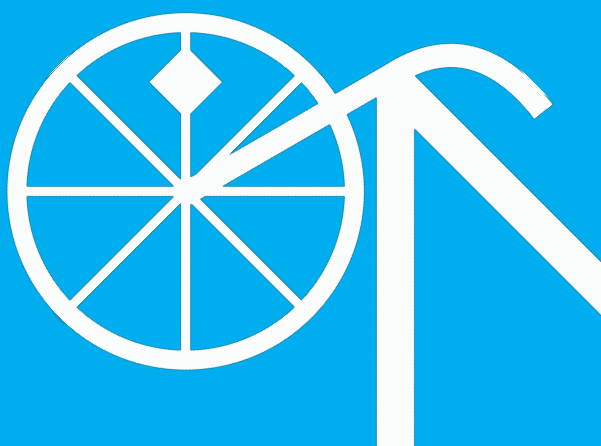
3.7. CONCLUSIONS

We have presented DIPS, a new debugging platform for intermittently powered battery-free devices. Making it easier to debug and test intermittently-powered devices, addressing the debugging and testing challenge. It closely couples a hardware debugger for embedded systems capable of debugging intermittent systems and an energy emulator allowing to replay real-life and synthesised energy traces. The close interaction of the debugger and emulator allows for seamless pausing of emulation during debugging actions (such as a breakpoints) whilst keeping the Device Under Test (DUT) powered and resumes emulation as the DUT continues operation. DIPS' software testing scripts allow for automatic verification of memory/peripheral restoration during intermittent operation. User experience studies have shown that DIPS enables debugging of intermittently-powered devices the same way as one would debug battery-powered embedded devices. Moreover, as a case study, using DIPS we were able to identify bugs in state-of-the-art intermittently-powered battery-free computing systems such as ENGAGE presented in Chapter 2.

CHAPTER 4

BATTERY-FREE TIMEKEEPING

This chapter is based on:
Jasper de Winkel, Carlo Delle Donne, Kasım Sinan Yıldırım,
Przemysław Pawełczak, and Josiah Hester (2020).
Reliable Timekeeping for Intermittent Computing
Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, Lausanne, Switzerland, 53–67.



4.1. INTRODUCTION

In previous chapters, we demonstrated the feasibility of interactive battery-free devices and how battery-free devices can be debugged and tested. However, the need for infrastructure-less wireless connectivity on battery-free devices remains unaddressed. Before we can address the challenge of wireless networking to enable interactive and connected applications, we first need to address a fundamental feature of traditional battery-powered systems, that is *robust timekeeping*. The ability to keep track of time undergirds a multitude of computing and networking primitives, such as synchronization and networking, data collection, and real-time operation.

Problem Statement. Having access to an accurate, continuous notion of time has always been taken for granted in embedded system development. Timeouts and timestamps constitute the backbone of embedded applications, particularly those targeting sensing, communication and actuation. The intermittent operation of ultra-low-energy batteryless devices makes it impossible to track time solely relying on on-chip digital timers, as they are not available during a power outage, as shown in Figure 4.1. 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 cover 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 expected outage, even if short outages are the common occurrence. This wastes energy maintaining the large energy buffer.

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] For instance, more than a second for Abracon AB18*X*5 [200] ultra-low-power real-time clock.

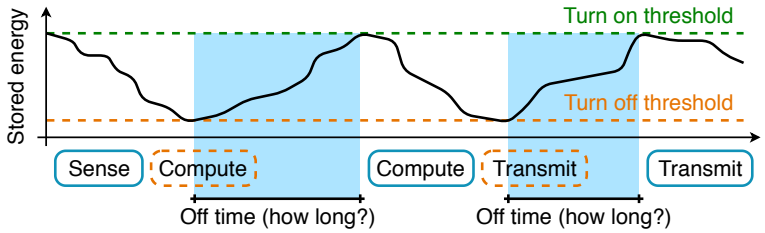


Figure 4.1: Energy-harvesting batteryless sensors operate intermittently, with power failures of unpredictable duration. These off times are hard to measure with existing timekeepers such as real-time clocks.

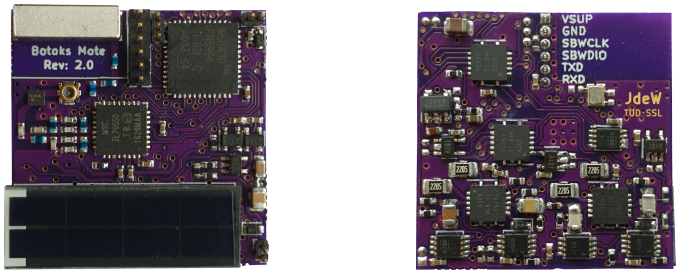


Figure 4.2: Botoks platform. The front (left) contains radio, antenna, solar panel and MCU. On the back side (right) is the Cascaded Hierarchical Remanence Timekeeper. The board dimensions are 1"×1".

Contributions. This chapter seeks to provide *hardware and software kernel support to enable continuous timekeeping* that is resilient against repeated power failures incurred by *batteryless devices operating intermittently* and adaptive to the dynamic constraints of intermittent computing. The *hardware layer* is inspired by the concept of remanence timekeepers [245, 117] 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 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 in 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 batteryless (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. *Batteryless 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, featuring an energy harvester, an CHRT and an ultra-low-power active radio (see Figure 4.2) ready to be used to prototype complete timekeeping batteryless applications.

Table 4.1: 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.

RTC Model	Resolution	Start-up time		Current
		Nominal	Worst	draw
NXP PCF85263A [266]	10 ms	200 ms	2 s	320 nA
Abracon AB18X5 [200]	10 ms	900 ms	21.5 s	55 nA
ST M41T62 [282]	10 ms	<1000 ms	1 s	350 nA
Maxim DS139X [138]	10 ms	<1000 ms	N/A	500 nA

The timekeeping architecture embedded in Botoks[2] is first characterized independently, and then evaluated inside two case-study applications, emphasizing the importance of timestamps and network synchronization.

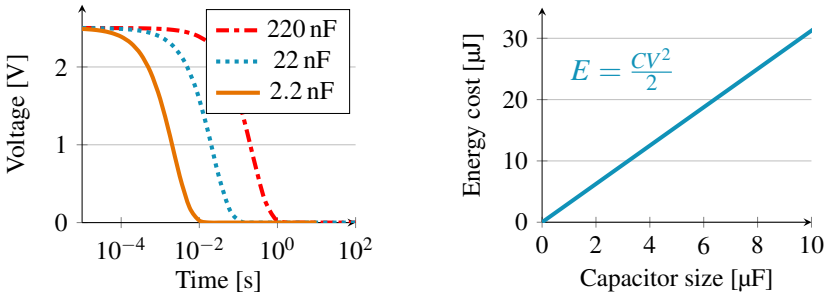
4.2. MOTIVATION

Low-power timekeeping solutions of today 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 [323, 118, 97]. 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 4.1). 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 a killer for intermittently-powered devices, where power-down failures are usually longer than operational time. These devices lose power even tens of times per second [246, 289], meaning that an RTC on a 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 [176], 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.

4.2.1. REMANENCE TIMEKEEPERS

Remanence timekeeper, introduced in Mayfly [116], was the most promising attempt at intermittency-safe, infrastructure-free local timekeeping. 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.

[2] Phonetic transcription of BTKS: Batteryless Timekeeping Sensor.



(a) Off time measurement increases with capacitance.

(b) Larger timekeepers cost more energy.

Figure 4.3: Partial design space for Remanence Timekeepers.

4

Trade-Offs. To use a remanence timekeeper the developer is presented with a complex trade-off space between application timing requirements, resolution, and available energy. 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 a resistor of the same size, which means that longer outages can be timed (Figure 4.3a). However, charging a larger capacitor costs more energy at each reboot and more time to charge (Figure 4.3b), 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 that can satisfy different needs of an application, balancing and anticipating all these trade-offs at design time.

Timekeeping Rigidity. Beyond the physical constraints, application behaviors with time-varying timekeeping requirements mean that a single remanence timekeeper tuned to a *specific outage length* is not useful for outages of a different length. For example, synchronizing radio transmissions needs millisecond-level timekeeping, while a humidity sampling task only needs minutes or hours. A single remanence timekeeper 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 needs are application-dependent and dynamic. Static remanence timekeepers are too rigid.

Cost, Size, Complexity. Despite the large trade-off space, remanence timekeepers offer attractive benefits due to the 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.

Table 4.2: 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.

Platform	Intermittent-safe	Infrastructure-independent	Time-aware
Hamilton [160]	✗	✓	✗
WISP [276]	✓	✗	✗
BLISP [126]	✓	✗	✗
Capybara [51]	✓	✓	✗
Flicker [114]	✓	✓	✗
Botoks	✓	✓	✓

4.3. SYSTEM OVERVIEW

In light of the shortcomings of single remanence timekeepers and RTCs, in this chapter we argue for a different approach where multiple remanence timekeepers of increasing capacitance are chained together. 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 CHRT are summarized below.

Consistent Time Resolution. Compared to single-tier designs, the multi-tier timekeeper does not suffer from loss of resolution as the discharge curve flattens, meaning that high resolution can still be leveraged after relatively 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 time tracking 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 run-time 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. Looking at Table 4.2, compared to existing low-power platforms, 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 the 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 hardware architecture and platform. This allows developers to use the timekeeping functionality without having to delve

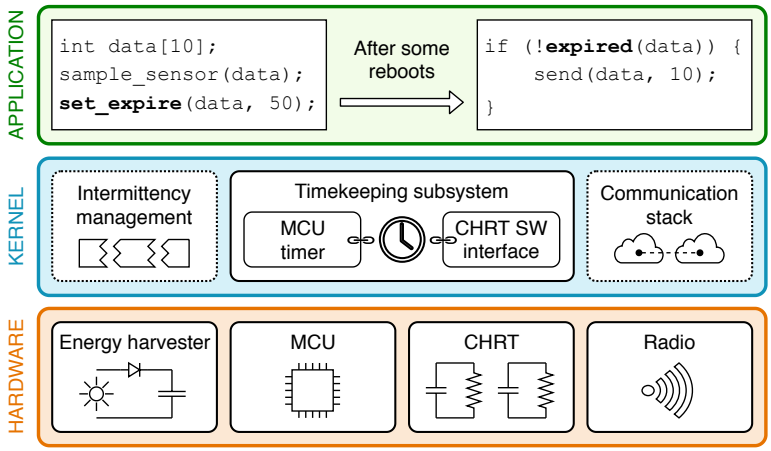


Figure 4.4: Botoks high-level architecture. The runtime kernel keeps track of time using the CHRT. Applications are instrumented with CHRT functions to infer on the elapsed time despite power outages.

into the underlying physics of capacitor discharge. This hardware abstraction software layer allows for integration into any intermittency-management runtime like InK [325] or Chinchilla [190]. In Section 4.4 and Section 4.5 we delve into the design aspects of the timekeeping hardware and its accompanying software support, respectively. Both hardware and software layers are integrated into Botoks, as portrayed in Figure 4.4 and described in Section 4.6, which will be used to evaluate our design (refer to Section 4.7).

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.

4.4. CHRT: HIERARCHICAL TIMEKEEPING

The limitations of ultra-low-power timekeepers for batteryless and intermittently-powered systems listed in Section 4.2 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, 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 (Section 4.5), 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.

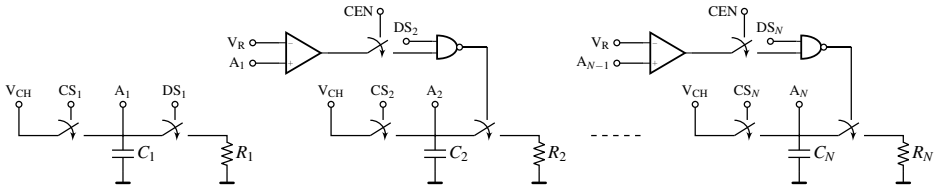


Figure 4.5: Cascaded Hierarchical Remanence Timekeeper. The left-most stage is activated first, while the subsequent stages are activated in a cascaded manner. V_{CH}: regulated output that charges the timekeepers; CS_i: charge signal of tier i; A_i: voltage sampling point of tier i; DS_i: discharge signal of tier i; CEN: cascaded mode enable signal; V_R: comparator reference voltage.

4.4.1. CHRT CIRCUIT

A schematic of the CHRT circuit is shown in Figure 4.5 (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. Using the tiers independently, programmers can map tiers to particular functions based on the timing granularity required.

4.4.2. 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 [116]. 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. Assume a resolution of δt is required, using an N-bit ADC, a resistor R and a capacitor C, and during calibration and usage the capacitor is charged at V_0 (which is also the maximum value the ADC can measure). We want to find the maximum time interval $\Delta t^{\max}$ such that, for all $\Delta t_x < \Delta t^{\max}$ and $\Delta t_y = \Delta t_x + \delta t \leq \Delta t^{\max}$, the difference between the ADC values corresponding to Δt_x and Δt_y is at least K integers. Larger values of K

yield better robustness against noise, but reduce the timekeeping range of a capacitor.

Assume that V_x is the voltage associated to Δt_x, and V_y is associated to Δt_y ($\Delta t_x < \Delta t_y \Rightarrow V_x > V_y$). Then, we want

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

and, by applying the RC circuit discharge model, that is, $t = -RC \ln(V/V_0)$, to (4.1), we obtain

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

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

Number of Tiers. Applications have a variety of accuracy and range requirements. In a critical software section a data sample might expire within a few milliseconds, non-critical data samples within 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, expressed in terms of resolution and maximum time-able interval, (4.2) can be used multiple times to compute the number of CHRT tiers required and their size. We have implemented this heuristic in a script [69] that suggests a CHRT configuration based on the timing requirements[3]. However, we note that for many applications the default configuration used in Botoks is likely to be suitable (see Section 4.6). This configuration allows for timing outages up to 100 s with a resolution up to 1 ms.

4.5. CHRT SOFTWARE LAYER

The CHRT architecture presented in Section 4.4 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 software is responsible for a one-time factory circuit calibration that must be performed before using the CHRT, as in the case of RTCs.

4.5.1. 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

[3] Details on this heuristic are also presented in [63, Section 4.2.1].

recharge the tiers. A non-goal of the HAL is 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 [325] or Chinchilla [190]) 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.

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 N tiers $T_0, T_1, \ldots, T_{N-1}$, chosen as suggested in Section 4.4.2, and that $R_i = \left[t_i^{\min}, t_i^{\max}\right)$ is the optimal timekeeping range of tier T_i, again, determined as given in Section 4.4.2. We call *target tier* T_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 T_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 T_{x+1} (if it exists) in case $t \geq t_x^{\max}$, or T_{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 K_{L} and K_{R} are used to tell the CHRT software layer to charge all tiers in $\left[T_{x-K_{\mathrm{L}}}, T_{x+K_{\mathrm{R}}}\right]$, given that T_x is the current target tier. For instance, if $K_{\mathrm{L}} = K_{\mathrm{R}} = 1$, tiers T_{x-1}, T_x and T_{x+1} would be recharged on reboot, and the discharge would start from tier T_{x-1}, and continue with the larger tiers in a cascaded fashion. The parameters K_{L} and K_{R} control the trade-off between timekeeping robustness and energy consumption, as charging more tiers requires more energy. In particular, K_{R} has a higher impact on energy consumption, as larger tiers consume more energy.

4.5.2. 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-

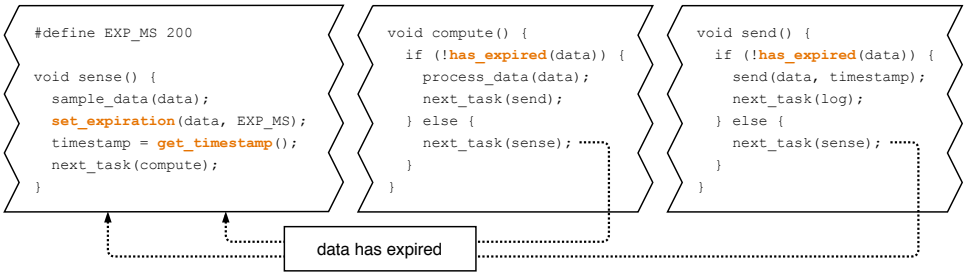


Figure 4.6: Example usage of the high-level CHRT API. The program is a typical sensing routine, and the API is used to specify that the elapsed time between sensing and transmission of some data must not exceed 200 ms of absolute time (on and off), as well as to get a timestamp to be sent together with the data.

level timekeeping tools to be used in real-world batteryless 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 4.6 demonstrates the usage of this high-level API.

Timestamp Generation. `get_timestamp()`: a high-level API function, 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 be used to annotate data.

Expiration Timers. The high-level API exposes two more functions to keep track of aging data, and discard it when it is deemed expired. 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 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 the system time is updated (using `chrt_get_time()`) and the CHRT is recharged (with `chrt_charge()`).

4.5.3. CHRT SOFTWARE CALIBRATION

Ideally, the RC circuit discharge model, $t = -RC \ln(V/V_0)$, could be used to estimate the 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 implemented, to obtain better precision and accuracy of the timekeeper. During 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. This way, an interpolated version of the *RC* circuit discharge model is built and stored in the form of a lookup table. At run-time, the voltage across the target tier is used to look up the table and retrieve the corresponding elapsed time.

4.6. SYSTEM 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. All hardware, software, and tools, as well as documentation for CHRT and Botoks, are open-sourced [69].

4.6.1. CHRT PLATFORM

The four-tier CHRT was implemented (1) as a *system peripheral* integrated into Botoks, built with off-the-shelf SMD components (Figure 4.2), (2) as a *stand-alone* development board, featuring the same components as on Botoks, and (3) as an *integrated* version.

CHRT Tier Settings. Following the practical guidelines given in Section 4.4.2, 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 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. As for the capacitors, we mixed different sizes to obtain a variety of resolutions and timekeeping ranges, to cover the needs of various applications. Precisely, we chose to cascade four tiers of increasing size: 2 nF, 22 nF, 220 nF and 2200 nF. The best 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 time-able interval is 100 s, measurable using the largest tier, for which a resolution of 1 s is guaranteed.

CHRT with SMD Components. We built the CHRT with off-the-shelf components on a custom PCB as a proof of concept. Other than the four RC circuits, the CHRT includes other important components. Each capacitor is readable via a trace to an ADC channel on the MCU. Ultra-low-power D-type Flip Flops (SN74AUP2G79 [128]) protect the input signals after the MCU dies. TS3A4751 [130] switches are used to gate power to the capacitors. For the cascaded effect, TLV3691 [129] comparators are set to enable the discharge of a tier when the voltage on the previous tier drops below 0.25 V. The stand-alone CHRT includes jumpers to enable or disable individual tiers.

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 [286], and simulated the circuit with Cadence Virtuoso [37] 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 batteryless enabled microcontrollers for low cost instead of expensive crystal centered RTCs. We report the power consumption numbers in Section 4.7.3.

4.6.2. BOTOKS PLATFORM

Our batteryless 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 [132], with 256 kB FRAM and 8 kB 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 [127] 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 [54] chip and a monopole antenna. The complete fabricated device is shown in Figure 4.2. We expect this platform to enable other researchers and practitioners interested in battery-free devices.

4.6.3. SOFTWARE

The code of the CHRT software layer described in Section 4.5 is split into two subcomponents, 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 [54] radio transceiver. All the code is written in C and can be compiled with the MSP430 GCC compiler [131].

4.7. 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.

4.7.1. 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 4.7.4 and Section 4.7.5. To sample digital and analog traces, a Saleae Logic Pro 16 logic analyzer [255] was used. Finally, for power consumption measurements we used the STM32 Power Shield [281].

4.7.2. 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.

The second application is a message 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.

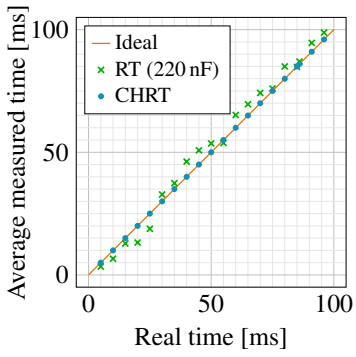
4.7.3. CHRT MICROBENCHMARK

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).

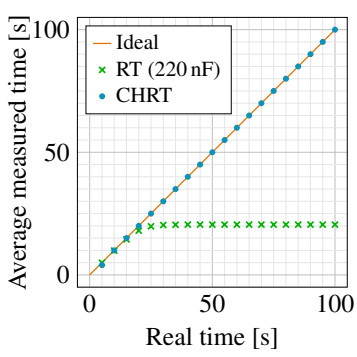
CHRT Accuracy. Figure 4.7 presents the accuracy of the smallest tier and the largest tier of Botoks's on-board CHRT. Figures 4.7a and 4.7b show the average reported time of the two tiers, for their optimal timing ranges, where every data point is the average of 10 measurements. The accuracy of a single-tier remanence timekeeper (of 220 nF) is also presented, to show the need for flexible 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.

Error Distribution. The same raw measurements were used to generate Figures 4.7c and 4.7d, 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.

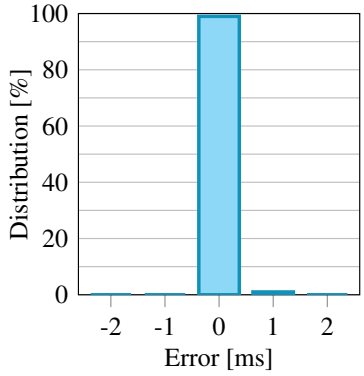
Energy Consumption. The static current consumption of the four-tier CHRT amounts to 0.958 µA for the SMD version (the one embedded in Botoks), and to 37.335 nA for the



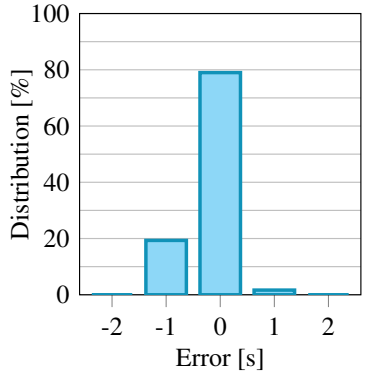
(a) Tier 0 (2.2 nF) average measured time.



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



(c) Tier 0 (2.2 nF) error.



(d) Tier 3 (2200 nF) error.

Figure 4.7: CHRT average measured time (top) and error distribution (bottom) 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).

integrated version[4], at 2.2 V. We also measured the energy required to charge the tiers at each reboot, as reported in Table 4.3. In Figure 4.8 the energy required to measure time is reported over the whole time measurement range. Not only CHRT 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 4.3. Even when all the four tiers have to be recharged, the initialization time of the CHRT (≈ 5 ms) is orders of magnitude smaller than for any available RTC (refer to Table 4.1).

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.

[4]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.

Table 4.3: CHRT dynamic energy consumption, i.e. energy to charge each tier, and charging times. The CHRT has much lower lower start-up time than any state-of-the-art RTC.

Timekeeper	Charge energy (µJ) Theoretical	Measured	Start-up time (ms)
CHRT Tier 0	0.00529	0.010†	0.3114
CHRT Tier 1	0.05819	0.110†	0.5053
CHRT Tier 2	0.58190	1.295†	0.913
CHRT Tier 3	5.81900	5.434	3.125
CHRT Total	6.46438	6.636	4.893

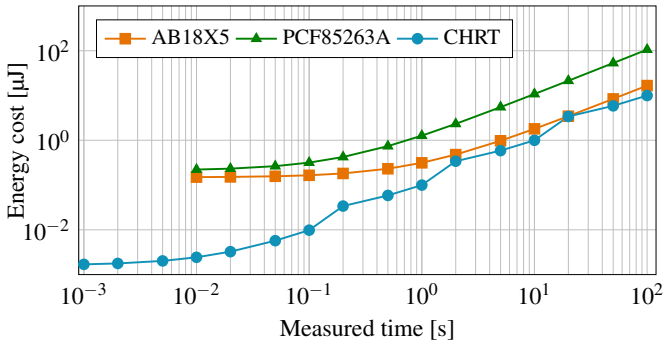


Figure 4.8: Comparison between CHRT and two RTCs from Table 4.1 of the total energy required to time a period. The CHRT allows for millisecond timing whilst requiring less or similar energy compared to the lowest power RTC.

4.7.4. 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 [247], 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 base station, was compared to the ground truth, measured with a logic analyzer connected to Botoks's power pin. As shown in Figure 4.9, 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 $\mathrm{km\,h^{-1}}$.

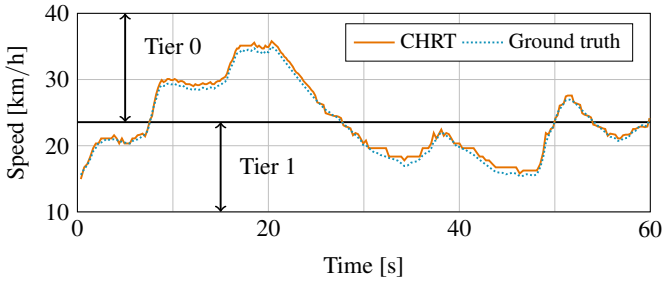


Figure 4.9: Calculated speed of the bicycle, in $\mathrm{km\,h^{-1}}$, over a period of 60 s. The result obtained with Botoks and its embedded CHRT is compared to the ground truth. As shown, the two smallest tiers of the CHRT are dynamically used through the experiment.

4.7.5. 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 [240] and harvested by a Powercast receiver [241] 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-B packets contained preamble, average on time measured with the CHRT, a dummy payload and 16-bit CRC, and was sent at a data rate of 200 kbit/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[5]. Specifically, the receiver uses the average transmission period contained in each packet to alter its listening schedule and align to the transmitter.

Packet reception rates were measured for the synchronized CHRT-powered protocol and for the non-synchronized baseline, for different energy harvesting conditions, as summarized in Table 4.4. Each experiment was run for 10 min.

Outcome. We note that this is the first successful case of two intermittently powered devices with active radios passing messages consistently. Figure 4.10 shows the percentage of received packets (the complementary of lost packets). As it stands out, 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

[5]Details on this network synchronization protocol are also presented in [76].

Table 4.4: Description of scenarios described in Figure 4.10. 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.

	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

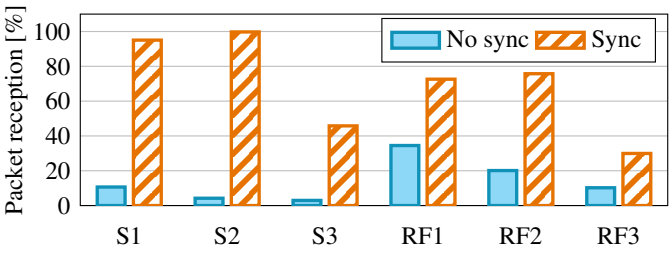


Figure 4.10: Percentage of received packets for a point-to-point link between two Botoks devices, measured in six different energy harvesting conditions described in Table 4.4. The percentage of packets received using the CHRT-powered synchronization algorithm is compared to the baseline case—with no 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.

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.

4.8. 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.

Batteryless Systems and Sensing. Batteries [232] 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 4.2. Botoks does not require central coordination and energy provision point, contrary to e.g. backscatter-based nodes [276] or visible light nodes [118].

Batteryless Timekeeping. The foundational work is [117] where two timekeeping techniques, *TARDIS* and *CusTARD*, were proposed to keep track of time across power failures in batteryless platforms. Both approaches exploit the decay of charge in physical components, SRAM and capacitor respectively, to generate a continuous notion of time. TARDIS is difficult to handle, has high memory overhead, and yields poor accuracy. CusTARD, that is also a foundation of Mayfly [116], has all the limitations of capacitive timekeepers listed

in Section 4.4. A similar discussion on Mayfly's timekeeping limitations was presented in [77]. As only very few of the checkpoint- and task-based programming models support time-sensitive data processing, we note that as the CHRT can be integrated into *any* runtime: task- or checkpoint-based. Enabling time-sensitive intermittent computation and sensing using the CHRT language constructs.

Energy Harvesting Sensor Networks. Batteryless sensors obtain energy for operation from the ambient. Since energy harvesting patterns arriving at these wireless sensor nodes fluctuate in time and are not easily predictable [183, 271], asynchronous network protocols, such as [213], are used in coordinating communication—proposed primarily for battery-powered wireless sensor networks. Another approach to deal with unpredictable energy harvesting patterns is to keep all nodes in the so-called *energy neutral operation* mode [86, 168], meaning that the consumed energy is always less than the harvested energy (thus power never fails). Our proposal diverges from aforementioned prior works targeted for energy harvesting systems. We propose a *synchronized* duty-cycling scheme based on batteryless timekeeping, which aligns nodes' schedules *across power failures*, similar to [98] but operating intermittently.

Network Time Synchronization. IoT nodes' clocks are generally sourced by cheap oscillators that are prone to significant, unpredictable instabilities. Due to these drifts, the local clocks of each node diverges over time. Consequently, it is mandatory to perform periodic time synchronization to ensure that nodes are able to acquire a common notion of time and perform coordinated actions [187]. Naturally, there is a considerable amount of work dedicated to wireless IoT sensors' time synchronization, such as [170, 180, 324]. The goal of these works is to build a network-wide notion of time. Unfortunately, all these works do not consider very frequent power failures and, in turn, the loss of synchronization state—a common phenomenon among batteryless platforms.

4.9. DISCUSSION AND FUTURE WORK

We discuss future directions enabled by Botoks and CHRT.

General Application. While the CHRT concept is particularly useful for intermittent computing, where power failures and energy constraints force designers to rethink timekeeping; the CHRT can be applicable to all embedded systems that need timekeeping through power failures. Using our CHRT language constructs, time-sensitive intermittent computation and sensing can be integrated into *any* intermittent computing runtime: task- or checkpoint-based.

Tool Support for CHRT. Beyond intermittently-powered networking, there is a dearth of tools for intermittent computing past EDB [49] and Ekho [112]. 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.

Complex Intermittently-powered Networks. Further exploration is needed on how to expand point-to-point Botoks link into a *full-fledged network*. 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.

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.

4.10. CONCLUSIONS

We have presented a new battery-less timekeeping mechanism enabling reliable battery-less sensing and computation, the *Cascaded Hierarchical Remanence Timekeeper* (CHRT), and presented two application instantiations useful for intermittent computing: time-critical sensing and intermittent communication, addressing the timekeeping challenge. 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.

CHAPTER 5
BATTERY-FREE WIRELESS NETWORKING

This chapter is based on:
Jasper de Winkel, Haozhe Tang, and Przemysław Pawełczak (2022).
Intermittently-Powered Bluetooth That Works
Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services. ACM, Portland, OR, USA, 287–301.

5.1. INTRODUCTION

In this chapter, we address the most prominent challenge with battery-free devices that undergirds practical deployments, that is **wireless networking**. As our demonstration of basic communication with a battery-free device, as shown in Chapter 4, does not integrate seamlessly into widely adopted wireless networks, it does not fully address the networking challenge. In this chapter, we set out to provide a generic architecture enabling wireless communication on battery-free devices compatible with pre-existing wireless networking protocols. Using this new architecture, we develop a battery-free smartwatch as shown in Figure 5.1, demonstrating the feasibility of connected interactive battery-free IoT devices.

Problem Statement. An IoT sensor requires a wireless link to communicate. This link needs to be ❶ low-power—to *operate as long as possible* on a single energy charge, ❷ belonging to one of the mainstream wireless standards—to be *backward compatible* with already deployed networks, ❸ independent from specific infrastructure, such as RF carrier wave generators to backscatter on—to be *flexibly deployable*, and ❹ bi-directional, as opposed to backscatter IoT tag-to-infrastructure links only—to *enable remote maintenance* and firmware updates [13, 25] as already-deployed IoT devices are severely limited without the ability to reconfigure it wirelessly. To address requirements ❶ and ❷ plenty of research has been dedicated to enabling ultra-low-power communication for popular wireless communication standards—leading to battery-free operation—through the backscatter principle. Recent examples of such wireless standard-compliant backscatter-based transmitters include LoRa [151], BLE [333] and Long Term Evolution (LTE) [45]. These wireless systems however do not address requirement ❸. A related approach based on a wake-up radio [239] is also not viable for the same reason. Therefore the only solution to battery-free IoT that addresses requirement ❸ is an ultra-low-power active radio. However, intermittent operation of a battery-free active radio, i.e., non-backscatter, IoT node implies that the active radio communication also becomes intermittent. Unless steps are taken to sustain the protocol state despite a power interrupt, even a single

Figure 5.1: First battery-free open-source [73] smartwatch with FreeBie: intermittently-powered bi-directional BLE link.

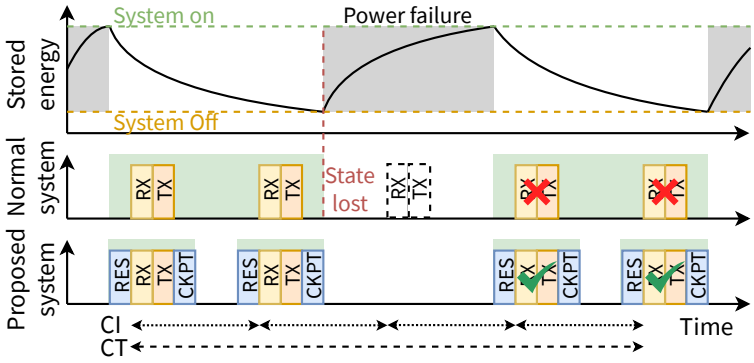


Figure 5.2: Intermittently-powered device operation. In a non-protected system even a single power failure causes the network state to be lost, requiring unnecessary new handshakes. In our proposed system the network state is stored and restored to/from non-volatile memory enabling to sustain connections. CI: connection interval, CT: connection timeout, RES: state restore, CKPT: state checkpoint, RX/TX: reception/transmission. ▮ region denotes the period of devices on time.

one will cause a connection drop, see Figure 5.2, forcing connection re-establishment which itself consumes time and energy of a battery-free system. Therefore, a design of a truly bi-directional (connection-based) wireless active link for a battery-free IoT sensor—addressing requirement ❹—needs to sustain already established connections despite frequent power interrupts. Unfortunately, all battery-free intermittently-powered state-of-the-art active radio platforms available are connection-less, or require the system to reconnect after each power failure. This includes broadcast-only (best-effort) BLE communication [97]. Simply, classical techniques to prolong IoT life based on duty cycling do not apply: intermittent power supply takes away the guarantee that the energy will be available at the scheduled wake-up times of the device [97, Section 7.1]. Custom protocol approaches such as [98] do not satisfy requirement ❷, as they are not backwards compatible with mainstream wireless standards. All this results in wireless communication being one of the unsolved challenges in the intermittent computing domain [275, Section 4(b)]. This challenge has not been tackled yet due to complexity of the problem: system designer needs to simultaneously consider network protocol specification constraints, application needs, energy demands and energy use. Also, the most common way of dealing with power interrupts: *checkpoining* system state and restoring it as the energy returns [184, Section 3.2], has never been applied to wireless network protocols. Checkpoints restoration have usually random duration and have to be placed at a precise point in time not to break the protocol timing. Therefore, a battery-free IoT that is useful, i.e., that addresses requirements ❶, ❷, ❸, and ❹, has yet to be achieved.

Contributions. Addressing the above problem we provide the following contributions:

***Contribution 1:* New intermittently-powered wireless systems architecture.** Our core novelty is a *new battery-free networked IoT device state checkpointing system operating on a process level.* Each core class of IoT device processes—application, network and Operating System (OS)—are checkpointed and stored in non-volatile memory individu-

ally. Checkpoints are triggered by the process scheduler based on real-time requirements of incoming processes. By triggering checkpoints in the scheduler we protect the network protocol state timing from unnecessary in-between checkpoints. This checkpointing approach enables keeping track of network protocol state in-between power interrupts and resuming already established communication when energy conditions allow. In addition, our device architecture allows the MCU to switch-off whenever possible and seamlessly resume, saving power when compared to sleeping. Both architectural novelties give rise to a *new peripheral abstraction layer*: the original network software stack never sees that the radio is powered off even when the storage capacitor is fully depleted.

Contribution 2: **Implementation of bi-directional active wireless link powered intermittently.** We pick one of the most ubiquitous IoT wireless system: Bluetooth, and in particular its low energy configuration BLE [32], and build a *BLE system powered intermittently*, called FreeBie, with its hardware and software released as open-source [73]. With our architecture, a battery-free BLE node can communicate bi-directionally with any BLE host and can sustain an already established bi-directional BLE link—even after multiple power outages at the battery-free BLE node.[1]

Contribution 3: **New battery-free IoT applications.** Our novel architecture enables never before seen IoT applications, in particular BLE *firmware updates* performed battery-free on an intermittent power and a *fully-functional battery-free smartwatch*, shown in Figure 5.1. Our architecture provides a foundation to connect and communicate bi-directionally for the next generation of battery-free IoT devices.

5.2. BACKGROUND, CHALLENGES AND KEY INSIGHTS

Taking a recent example, long-term experiments with a commercial-grade battery-free BLE node of [59] demonstrated that time of the day, orientation, and deployment location can affect the duration of continuous operation: from almost constant operation to few transmission activities throughout the day only [159]. Therefore, necessary system support for intermittently-powered devices is needed that takes care of [184] (i) *control flow*—to guarantee that the device will start from the state right before the last power failure, (ii) *data consistency*—to guarantee that the system will restore correctly from power failure, (iii) *environmental consistency*—to guarantee that the time-sensitive data will be handled correctly when data becomes outdated after restoring from power failure, (iv) *concurrency*—to enable execution of multiple active applications, and (v) ***undisrupted communication***—to enable wireless communication between (intermittently-powered) devices and guarantee synchronisation despite power interrupts.

While there are plenty of frameworks available that aim at attaining points (i)–(iv), see Table 5.1 and Section 5.7 later on, sustaining a communication of a radio link powered by an intermittent source, i.e., attaining point (v), is far from being solved.

Infeasible solutions for 'intermittent' communication. Increasing capacitor size or energy harvesting efficiency, is a typical solution to improve battery-free systems. Sadly, a large capacitor/energy harvester increases the device's size and increases charge times. So all active radio intermittently-powered devices trade-off capacitor and/or harvester

[1] A comparison of battery-free BLE platforms is given in Table 5.5.

Table 5.1: Comparison of relevant existing software support systems for intermittently-powered devices. Colour scheme: denotes non-desired or limiting features from the perspective of wireless communication protocol and denotes the desired features.

	InK [325]	**TICS** [166]	**Coala** [193]	**Capy-bara** [51]	**MPatch** [66]	**Empire** [3]	***This work***
Checkpoint trigger	Task end	In-code check-point	Task end	Task end	Voltage level	Voltage level	Process end
Checkpoint type	Task	Check-point	Task	Task	Check-point	Task	Process
Requires code instrumentation	Yes	No	Yes	Yes	No	Yes	No
Time-deterministic restoration	No	No	No	No	No	No	Yes
ARM-based processor architec-ture	No	No	No	Yes	Yes	No	Yes
Dynamic memory allocation	No	No	No	No	No	No	Yes
Interrupt support	Yes	Yes	No	No	Yes	No	Yes
Peripheral support	No	No	No	Yes	No	Yes	Yes
Preemptive scheduling	Yes	No	No	No	No	No	Yes

size for communication functionality, sending connection-less data, such as beacons in case of BLE, e.g. [147, 273]. Simply, when the device powers off mid-transmission, the device will restart and re-send the beacon again. This however is not feasible for connection/handshake-oriented protocols, especially if they require strict timing to establish and sustain a connection. Lastly, reconnection costs packet transmissions. For example, for our smartwatch's BLE link each reconnection would **require about 70 packets to be sent** by the BLE client, taking more than 43 s to reconnect at low (200 lx) ambient light. These packets include connection establishment, negotiation of connection parameters, service discovery and notification configuration. Apart from the above-mentioned connection re-establishment overhead, if the BLE device needs to re-establish the already-existing connection, one would have to build much more complex application. Such application would have to take care of storing authentication information, tracking network status, or transfer progress information—all causing processing overhead, i.e. taking extra time before the existing connection is re-established.

Other techniques, such as 'classical' duty cycling [40] and transmission power control [88], are also infeasible. Duty cycling is problematic, as the main capacitor may get depleted within the sleep period resetting connection state, while duty cycling itself consumes energy during sleep, e.g. for the NRF52840 BLE module sleep current is 3.16 µA at 3 V [220]. Needless to say, it is impossible to turn the system off completely and then wake up: some components like the RTC need to remain powered to wake up the system from sleep. In the case of transmission power control, additional coordination between the sender and receiver would be required to avoid dropping packets and possibly the connection when adjusting transmission power. When considering adaptation of transmitted packet lengths: reducing packet length would not scale linearly with energy expenditure, as overhead such as the crystal ramp-up time and pre- and post-processing remain constant [221]. Thus, the goal is to *create a framework that enables unobstructed communication on intermittently-powered budget.*

Table 5.2: C/C++ lines of code of different network protocol implementations. Compared to the orders of magnitude smaller codebase of software support systems for intermittent operation, e.g. [193, Table 3], the cognitive burden to analyze and instrument protocol code is unprecedented.

Protocol type	Implementation	Lines of code[1]
Bluetooth	Packetcraft [16]	397 200
TCP/IP	LWIP [91]	88 100
Thread	OpenThread [104]	250 500

[1] Measured with cloc [61] (rounded to 100 lines).

Framework Requirement 1: **Time-aware Checkpoints and Real-time Restoration.** We postulate that if the duration of power failures is within the allowed connection timeout (see again Figure 5.2), the transmitter and the receiver should resume the connection without the need to restart. This resumption must be supported by a framework responsible for copying of the protocol state, i.e., volatile MCU registers and volatile memory to a non-volatile memory before the power interrupt, and restoring the last saved state back to the respective volatile registers.

There exist two classes of frameworks that optimise the amount of memory to store and resume: task-based, e.g. [193] and checkpoint-based, e.g. [166]. A task is a section of code with defined input and output of non-volatile variables. Tasks are connected through these variables to form a state machine. Sadly, automatic/compiler-based transformation of a wireless protocol implementation code into tasks is not feasible due to the large and complex codebase, see Table 5.2. This means that the developer would have to do this by hand—*a daunting task to achieve* [166, Section 5.4]. A conceptual counterpart to a task is a checkpoint, i.e., a function inserted manually or automatically at compile time (to the original codebase) that stores the program's state until that checkpoint. Sadly, both tasks and checkpoints introduce a computation penalty—the store and restore operation. In other words, *checkpoints or tasks will break the protocol's timing* nullifying their use for wireless networking. Fortunately, state saving and restoration can be triggered not only by the end of a task or a checkpoint, but also by the internal timer or the capacitor voltage monitor. However, the timer would have to follow the protocol state timing *precisely*, while the voltage-based trigger does not follow any timing. Finally, the restoration time from the checkpoint must be constant and time-bounded. Otherwise, the real-time requirement of the wireless protocol state machine will be violated.

To summarize, **the key challenge** is that existing checkpoint and restoration methods, see Table 5.1, are not suitable for wireless networking support. **Our key insight** to solve the challenge is that *the system state checkpoint can be triggered by the end of the wireless protocol process*, i.e., the only atomic operation that must be executed without interruptions. This way no code instrumentation, fixed checkpoint timers, and voltage monitors are needed.

Framework Requirement 2: **Virtualisation of time and peripherals.** Time and peripheral state would be disrupted by intermittent operation. Thus, dedicated software abstraction layer to mask (*virtualize*) time and peripherals to network protocol is needed. **The key challenge** is that, referring again to Table 5.1, no software framework is able to support peripherals and time (required for time-deterministic restoration).

Our key insight to solve the challenge is to maintain a continuous notion of time despite intermittent operation and to develop a software abstraction layer that offsets on-board peripherals such as timers according to the continuous notion of time, masking the effects of intermittency. The continuous notion of time can be enabled by both a CHRT (Chapter 4) or by using a over provisioned RTC that is powered for as long as possible. The current draw of modern low-power timekeepers is in the order of nA [9]. As the sleep mode consumption reaches µA levels, only keeping the timekeeper on is *about ten times more energy efficient* than sleeping.

Framework Requirement 3: **Dynamic handling of network connections.** A framework must adapt connection parameters to available harvested energy. The framework must prioritize an already-established connection or focus on sustaining on-device computation and sensing. The system must support preemptive scheduling (to prioritise network processes over application or OS processes). **The key challenge** is that these features are also not supported by existing systems except for InK [325] (although InK would require manual code transformation), refer to Table 5.1. **Our key insight** to solve the challenge is that network protocols allow adjustment of the Connection Interval (CI), this and other parameters can be used as a foundation for connection adaptation.

5.3. INTERMITTENTLY-POWERED WIRELESS SYSTEM

Driven by Framework Requirements 1, 2 and 3, we propose an *architecture that sustains wireless protocol communication for intermittently-powered devices*, see Figure 5.3.

5.3.1. TARGET NETWORK AND DEVICE ARCHITECTURE

Network Topology and Device Capabilities. We consider a star network topology following a Connection Interval (CI)-oriented Connection Timeout (CT)-driven wireless communication protocol of choice, as exemplified in Figure 5.2. *The host is tethered or battery-based.* The end device, on the contrary, is battery-free and intermittently-powered (by energy harvester). We assume that both devices do not share a common signal that can be used to synchronise them, as in e.g., [97]. As with most network protocols, the end device has to announce its presence to a host for a connection to be established.

End Device Hardware. We propose a battery-free end device logically separated into two power domains, see Figure 5.3: (i) *processor (MCU) power domain* and (ii) *"always-on" ultra-low-power domain*—charging on-board capacitors through onboard solar panels, through which both domains are powered. The multi-power domain architecture allows for either a RTC (through overprovisioning of power) or the CHRT (Chapter 4) in order to keep a continuous notion of time. The (external) timekeeper is able to switch the processor power domain off completely, as proposed in [153]. In this chapter, we specifically chose to use a RTC to show that even with only typical IoT device components, that intermittent operation whilst maintaining wireless communication is possible and can even save power. This choice results in additional overhead due the limited resolution and the continuous power requirements of RTCs. An extended CHRT with the ability to turn on the processor power domain serves a alternative with a lower start-up time.

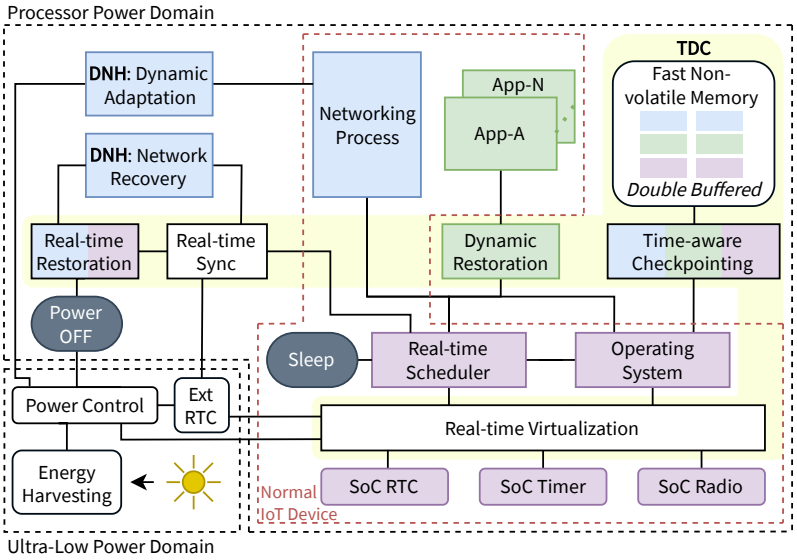


Figure 5.3: Proposed architecture for intermittently-powered battery-free wireless communication systems.

5.3.2. SYSTEM COMPONENTS

TIME-AWARE CHECKPOINTS AND REAL-TIME RESTORATION

We propose *Time-Deterministic Checkpoint (TDC)*: a time-aware checkpoints and real-time restoration system for intermittently-powered wireless networking protocols. TDC, addressing Framework Requirement 1, is built as follows.

Process as Atomic Data Structure Checkpoint. We categorize processes in three groups: (i) *network*, (ii) *OS* and (iii) *application* (with one process per concurrently-running application). Implementation-wise, a process is a hand-picked file/directory part of a source code. Each process can be classified as requiring real-time operation or not. Per default the processes network and OS are real-time processes, the application process(es) can be real-time or non-real-time.

Process Separation and Memory Structure. The memory separation for all processes is performed by the linker at the code compilation stage. The developer splits the source code of a complete system into code directories—one per process. Static memory declared in the source files of each directory is allocated in separate regions as shown in Figure 5.4. The registers, stack and heap are included in the OS process checkpoint.

We assume that the memory from a non-restored process cannot be read or written to. This can be enforced by a Memory Protection Unit (MPU), preventing writes and reads to unrestored memory. Communication between processes occurs through the OS using Inter-Process Communication (IPC) with dynamically allocated messages. Finally, we note that the developer can specify variables (such as a buffer for logging) that do not need to be checkpointed and restored. For this case, a dedicated region in volatile memory is allocated; see Figure 5.4.

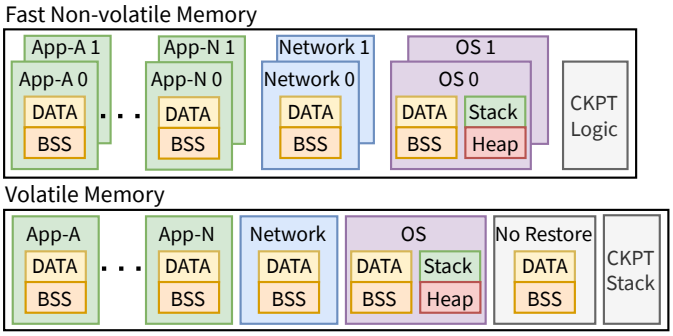


Figure 5.4: Memory map of intermittently-powered wireless networking device. Numbers 0 and 1 for App-*x*, Network and OS process denote buffer numbers of double buffered memory. Colours match respective processes in Figure 5.3.

Process Checkpoint Scheduling. In modern network protocol stacks, each networking process, at the end of its execution, provides information to the device OS when the next networking process, e.g., beginning of a next connection interval as shown in Figure 5.2, occurs. Therefore the core component of our architecture is a process scheduler handled by the battery-free device's OS. The scheduler using the virtualisation layer (defined in Section 5.3.2) decides on the next moment in time when the battery-free device powers on again. The power-on moment is determined by the process with the earliest deadline in the scheduler's queue. If more than one process has the same deadline, than these are served as first-come first-served. Each running application shares the same priority and is queued through a scheduling queue, while the network process (through an interrupt) can preempt the application process residing in the queue's head.

Considering what processes are at the top positions of the scheduler queue pending execution three combinations of processes that occur during a power-on cycle exist. These are: (i) application only, (ii) network only, and (iii) combined application and network. In case (i) and (ii) respectively, the network process or application process does not have to be checkpointed and restored. Case (iii) occurs when the application process is scheduled for execution close to a network process, or when the network process triggers the application to run. In this case, the application is dynamically restored by the scheduler. When there is no process awaiting execution, the MCU is placed either into sleep mode or the processor power domain is switched off. The decision of whether to switch off the processor power domain or to sleep is based on the duration of the checkpoint and restoration process. In our architecture, we switch off the processor power domain if the next process event is scheduled to start later than T_{minOff}, i.e., as a minimum the combined time of checkpoint and restoration. Otherwise, the MCU is set into sleep mode. Depending on the implementation T_{minOff} can be set to the break-even point between additional energy cost of checkpointing and restoring and the power saved by turning the processor domain off. Then, the processor domain only switches off when it is energy-wise beneficial to do so, at the cost of less frequent checkpoints.

Prior to switching off, the state of the system has to be checkpointed. First, the next power-on time is determined—that particular step is performed by the time and peripher-

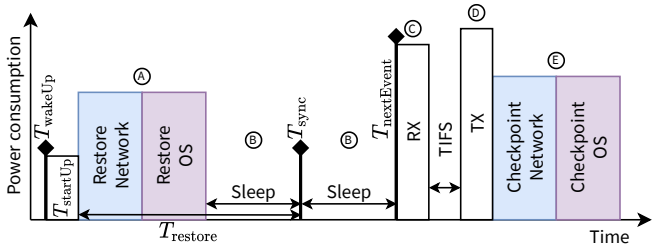


Figure 5.5: System events in one network-only cycle (power consumption levels marked as Ⓐ–Ⓔ); TIFS: Time Inter-Frame Space, T_{sync}: RTC time synchronisation moment.

als virtualisation layer described in Section 5.3.2. Next, based on the scheduler queue, the real-time processes that need to be restored during the next cycle are determined. This information is stored in the next OS checkpoint. Then, any process combination—either (i), (ii) or (iii)—that has been executed during the current power-on cycle is checkpointed, followed by the OS checkpoint. Finally, the device is switched off. An illustration of a power cycle consisting of only the networking process is presented in Figure 5.5.

Process Checkpoining. Application and network process checkpoints are always committed with an OS process checkpoint. This protects from a situation where memory can be dynamically allocated within an application or network process and then lost due to an incomplete OS checkpoint. In order to make the system incorruptible each checkpoint is double-buffered, with two dedicated memory regions allocated for each process, as shown in Figure 5.4. The OS process checkpoint must be restored after the scheduled wake-up from power-off as it includes the processor registers, stack, heap, scheduler, OS functionality (such as timers, queues and IPC), and the peripheral state.

Process Restoration. Processes can be restored either as non-real-time or real-time. Non-real-time processes are loaded dynamically prior to execution by the scheduler. For non-real-time checkpoints the restoration time is neither monitored nor compensated.

Unlike non-real-time processes, real-time processes are restored in advance prior to the scheduler resuming operation. As process checkpoint sizes may vary, we add a margin on top of the processes restore time. The allocated time for restoration, including the margin, is denoted as T_{restore}.

Upon resuming from the power-off state, the system begins process restoration, as shown in Figure 5.6. With a checkpoint present, the OS process is restored together with the network processes and real-time application processes as determined by the scheduler in advance. Once the virtualisation layer compensates for the power-off time the system synchronises with the external time source (refer to Section 5.3.2) and the device resumes operation as normal, i.e., moving to sleep mode and then executing the networking process or other application processes. If the device is unable to power back up at the desired time after switching off, or if the time synchronisation was unsuccessful, a recovery process is instantiated driven by *Dynamic Network Handling (DNH)*, as explained in Section 5.3.2. During the first boot, the system is synchronised to the external time source and starts operation.

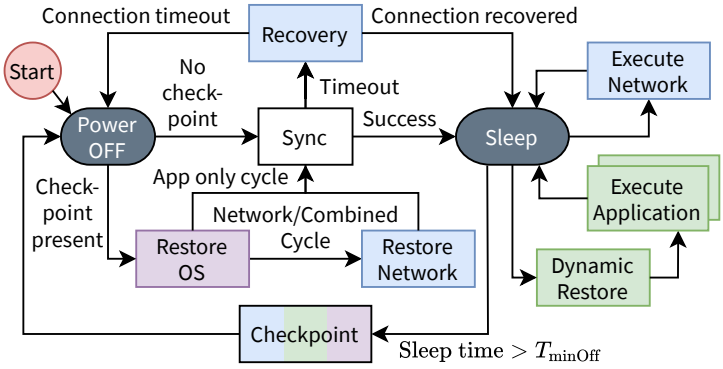


Figure 5.6: Restoration of a process after power off.

VIRTUALISATION OF TIME AND PERIPHERALS

We propose a time and peripheral virtualisation layer, addressing Framework Requirement 2. The virtualization layer is placed logically between the peripherals and the pre-existing network software stack (see Figure 5.3).

Our architecture needs a method to synchronise to external RTC time for our system to deterministically execute real-time processes. This synchronisation step needs to occur after real-time process restoration and prior to the scheduler resuming operation and is performed as follows. When power is reapplied to the processor power domain, the MCU starts up and the boot time of the MCU, denoted as T_{startUp}, is considered consistent. Next, the real-time processes are restored, as described in Section 5.3.2 (*Process Restoration* paragraph). Finally the system awaits a synchronisation pulse at T_{sync}, where $T_{\text{sync}} \leq T_{\text{nextEvent}}$ and $T_{\text{nextEvent}}$ is the starting time of the upcoming process event. T_{sync} is the point in time that synchronises the system to the external time and marks the resumption of real-time operation and scheduling. T_{sync} should be as close as possible to $T_{\text{nextEvent}}$ as allowed by the external RTC resolution to avoid overhead. The next wake-up time is given as $T_{\text{wakeUp}} = T_{\text{sync}} - T_{\text{startUp}} - T_{\text{restore}}$. Since T_{sync} is known prior to turning off, this value is stored as part of the OS checkpoint and used as a reference starting value for the MCU timing peripherals during the next power-on cycle.

DYNAMIC HANDLING OF NETWORK CONNECTIONS

Addressing Framework Requirement 3, we introduce DNH—a final component of our architecture. DNH is responsible for: (i) network recovery and (ii) dynamic network adaptation. *Network recovery* is needed when the device does not turn off according to the schedule but turns off unexpectedly due to a power failure. As most wireless network protocols operate based on Connection Timeouts (CT), when a connection has been established but no packets are received within the CT window, the connection is dropped (see Figure 5.2). In our architecture, if the device after a power failure can harvest enough energy to turn on before CT is exceeded, connection recovery is executed when the device powers back up before resuming operation. That is, missed connection events are skipped and the network process is scheduled for the next connection event. *Dynamic*

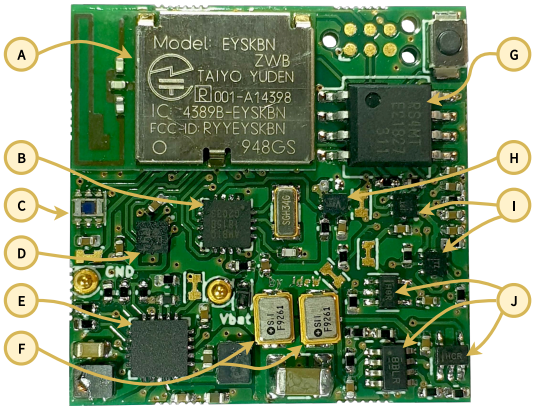


Figure 5.7: FreeBie mote. A total size is 1"×1". Components marked as Ⓐ–Ⓙ are explained in Section 5.4.

network adaptation further improves the performance of our system, by monitoring the available amount of energy. The Connection Interval (CI) is decreased in the case of abundant energy and increased when little energy is available. This method allows the system to adapt to changing energy conditions whilst keeping the connection alive and increases responsiveness in the case of abundant energy.

5.4. SYSTEM IMPLEMENTATION: FREEBIE

We proceed with the implementation. As a case study we select BLE and denote its intermittently-powered version as FreeBie.

5.4.1. FREEBIE HARDWARE

A fabricated FreeBie is shown in Figure 5.7, with hardware and software open-sourced [73]. Its main blocks are as follows.

Wireless Connectivity and Storage. FreeBie is built with a wireless module [56] containing a nRF52840 BLE ARM-based MCU [220] (Ⓐ in Figure 5.7). To store the state of the system in-between power failures, MB85RS4MT fast non-volatile Ferroelectric Random Access Memory (FRAM) [92] (Ⓖ in Figure 5.7) is chosen.

Timekeeping. The AM1815 RTC [9] is chosen (Ⓑ in Figure 5.7) for its 10 ms resolution and low power consumption, i.e., 55 nA at 3 V. We did not use hardware timer like the TPL5111 [298] used by [141] due to their inability to sustain the clock accuracy of the BLE specification [32, CS 5.3].

On-board Sensors. FreeBie contains two external sensors: an OPT3004 luminosity sensor [297] (Ⓒ in Figure 5.7) and a BMA400 accelerometer [33] (Ⓓ in Figure 5.7). Both sensors are powered through the MCU only when required. These sensors are included in FreeBie to enable the community to build new applications on top of the FreeBie mote.

Energy Management. FreeBie is solely powered by harvested solar energy using a BQ25570 energy harvester [299] (Ⓔ in Figure 5.7). Its boost converter boosts the voltage

generated by two EXL2-1V50 solar panels [179], as seen in Figure 5.1. The harvested energy is stored in two parallel 7.5 mF capacitors [264] (Ⓕ in Figure 5.7). The chosen storage size is dictated by the compatibility with an Android [103] OS, used as one of the BLE hosts (see Section 5.5). Android initialises a BLE connection with a CI of 45 ms. The chosen storage must sustain this CI until new connection parameters can be applied.

The processor domain is powered by energy harvester's internal buck converter configured to generate an 1.8 V supply. This output is switched on when the voltage of the storage capacitors reaches 2.6 V and is switched off when it drops below 2.05 V. With the external power switch [309] (Ⓗ in Figure 5.7), the external RTC is able to switch off/on power to the processor power domain. In order to protect the MCU while it is off, logic/switches prevent always-on signals from reaching the processor (Ⓙ in Figure 5.7).

Display. We added the option to connect a display to the FreeBie mote. Specifically, a Sharp 1.28" LCD-TFT [269], connected through SPI to the MCU (both connectors for the solar panels and display are present at the back of FreeBie). The display is used in a smartwatch application and is powered from the main energy storage. Power to the display is controlled by the MCU and the state is maintained using a SN74AUP2G79 flip-flop [128] allowing the display to stay on when the MCU is off (Ⓘ in Figure 5.7).

5.4.2. FREEBIE SOFTWARE

The Packetcraft BLE stack [231] was chosen as the basis to implement FreeBie's system architecture. Packetcraft implements all the required Bluetooth standard layers—from layers that configure the MCU's registers to high level layers (such as Attribute Protocol (ATT) profiles).[2]

With our architecture only relatively simple modifications are required to run the BLE networking stack intermittently. First, the code source files needs to be separated into application, network and OS processes, allowing for easy checkpointing and restoration. Second, the scheduler needs to be modified to allow the system to switch off when idle and compensate for the power-off time of the device upon restoration. Finally, dynamic connection handling is build on top of standard BLE features. The implementation is described in detail below.

TIME-AWARE CHECKPOINTS/REAL-TIME RESTORATION

Process Separation Implementation. Process separation described in Section 5.3.2 is implemented as follows. Thanks to Packetcraft's logical separation between OS layer (called Wireless Software Foundation (WSF) with its underlying components—Platform Abstraction Layer (PAL) and the MCU peripheral drivers) and networking layer, manual source file separation is straightforward—WSF together with its underlying components form the OS process source files. The rest of Packetcraft source files are considered to belong to the network process. Applications are considered as a separate third group of process source files. With the separation of the code into processes, the network, application and OS volatile memory is split per process. These processes encompass all

[2] We are aware of other open-source implementations of BLE, namely Apache Nimble [11], and Zephyr [330]. From these implementations only Packetcraft supports BLE 5.2, can be deployed on Nordic nRF52 series MCUs, and can be built with the standard GCC toolchain [15].

volatile memory associated with the process except for dynamically allocated memory that is contained within the OS process.

Process Checkpoint Scheduling Implementation. The proposed scheduler, introduced in Section 5.3.2 (*Process Checkpoint Scheduling* paragraph), is implemented as follows, taking the WSF scheduler as basis. Normally when the OS scheduler is idle, the function `PalSysSleep()` is called and the system sleeps until the next process event. This function is extended to allow the system to checkpoint and turn off per the proposed scheduler criteria. In our implementation $T_{\text{minOff}} = 20$ ms is experimentally determined, and the type of next on power cycle is determined through the scheduling queues and MCU's RTC compare registers. When the system is able to turn off, first the next power-on time is determined by the virtualisation layer (its implementation will be described in Section 5.4.2). Then, real-time processes to be executed are marked for restoration in the next OS checkpoint. Finally, the currently restored/active processes are checkpointed, followed by the OS checkpoint, after which the processor power domain is switched-off through the virtualisation layer.

Secondly, we introduce a major change to the OS scheduler — the notion of *restored* and *non-restored* processes. Non-real-time processes are scheduled in a different queue than real-time processes, and each process possesses a state variable that indicates if the process has been restored or not. When a non-real-time process has not been restored prior to execution, the process is first restored, then marked as restored and finally executed. If the process has already been restored, the process is simply executed. Since all real-time processes for the next power-on cycle are identified in advance (and restored prior to the scheduler resuming operation), they are not tracked during operation since these processes (due to their real-time requirements) cannot be loaded on demand.

Process Checkpointing Implementation. The checkpointing framework checkpoints the uniquely defined memory sections of each process in volatile SRAM and stores the checkpoints in external FRAM. For the OS checkpoint, the stack size is determined using the stack pointer register and the heap size is determined by tracking the total size of the dynamically allocated memory, only the utilised portions of the reserved space for the stack and heap are checkpointed. As described in Section 5.3.2 (*Process Restoration* paragraph), due to T_{restore}, variations in checkpoint size and thus restoration time of the real-time checkpoints are compensated for.

Process Restoration Implementation. When the MCU powers up, first the external RTC is configured for the synchronisation point as defined in Section 5.4.2. The time of this point has been pre-selected before the MCU switches off as defined in Section 5.3.2. Next, the relevant process checkpoints are restored from FRAM to SRAM. Directly reading from external memory is not possible since it is slower than reading from internal SRAM and thus would influence the timing of the BLE stack. The restore time T_{restore} is set to 10 ms.

VIRTUALISATION OF TIME AND PERIPHERALS

The real-time virtualisation presented in Section 5.3.2 is implemented as follows. Due to the choice of external RTC, we are limited by a resolution of 10 ms. Since 10 ms is not an integer multiple of 32.678 kHz tics, i.e., the frequency of our external RTC, the 10 ms tics source induces additional jitter (in addition to the jitter of the crystal itself).

Due to our strict timekeeping requirements (500 ppm) this is not acceptable. Hence for synchronisation of the system to the external RTC time we synchronise to 250 ms intervals as synchronisation points, as this is the smallest integer multiple of 32.678 kHz tics possible with 10 ms resolution. We note that although the resolution of the external RTC is only 10 ms, on integer multiples, such as 250 ms, the timing is mainly only influenced by the jitter of the crystal itself.

The synchronisation point itself is a hardware signal sent from the external RTC to the MCU that, in turn, instantly starts the enabled MCU's timing peripherals such as the on-board RTC. Since the synchronisation point is a known moment in time, reads and/or writes to the on-board MCU's timing peripherals such as the RTC (as it starts from zero, and the state is lost during power-off) are compensated through the virtualisation layer by applying T_{sync} as an offset. Any read to the RTC time register through the virtualized peripheral API returns the compensated time instead of the raw time. Hereby the effects of intermittency are masked to the running processes.

Using the definition in Section 5.3.2, the start-up time is computed relative to the nearest synchronisation point. The value of $T_{\text{startUp}} = 10$ ms is experimentally found. By setting the alarm in the external RTC to T_{wakeUp} prior to switching off, the system will turn on at the alarm, as the external RTC can control the processor domain power via the external power switch. Finally, after the wake-up alarm is set, using the external RTC, the processor power domain is switched off.

DYNAMIC HANDLING OF NETWORK CONNECTIONS

The design presented in Section 5.3.2 is implemented as follows. In the Bluetooth protocol, the host dictates the initial connection parameters. For intermittently-powered devices these parameters, depending on the available energy, might not be suitable. Hence after a connection establishment we automatically request favourable connection parameters corresponding to the energy available in the system. If the BLE host forces a connection update, we will immediately request new connection parameters if the ones chosen by the BLE host are unsuitable.

During the restoration process we sample the voltage of the storage capacitors to determine how much energy is available to FreeBie. For simplicity, we define quantized energy levels as given in Table 5.3. According to the Bluetooth Core Specification [32, CS 5.3 (page 2255)] (i) CI shall be a multiple of 1.25 ms in the range 7.5 ms to 4 s and (ii) Supervision Timeout (ST) (referred previously as CT) shall be a multiple of 10 ms in the range 100 ms to 32 s and it shall be larger than (1 + SL) × CI × 2, where Slave Latency (SL) specifies how many connection events may be skipped by the end device, i.e., with CI of 4 s and SL of 3 allows FreeBie to stay off for almost 16 seconds). At the low energy level we use the maximal allowed parameters to let FreeBie stay powered off as long as possible. As more energy becomes available, FreeBie harvests more energy so we increase CI and decrease SL accordingly. Table 5.3 lists the requested connection parameters corresponding to these energy levels. We note that since connection update requests are not instantly applied, if the update is granted—they are applied at a later (specified by the host) connection event.

If during the restoration no synchronisation pulse is received at the expected time, recovery is executed. After reading and synchronising to external time, the decision is

Table 5.3: Requested FreeBie connection parameters. CI: Connection Interval, SL: Slave Latency, ST: Supervision Timeout.

Energy level	Luminosity	CI	SL	ST
Very low (dark)/low (dimmed)	200 lx/300 lx	4 s[1]	3	32 s
Medium (bright)	600 lx	2 s	1	32 s
High (overcast)	10 klx	1 s	0	32 s

[1] Real value is 3.99875 s as reference BLE stack [216] forbids a 4 s CI.

made based on the current connection settings if the connection is recoverable. If this is the case, the next connection event is scheduled and the state of the network stack is passed on to the future state. The network stack reattempts transmission of lost packets scheduled during the power failure. During the restore, prior to synchronisation, all real-time processes are restored.

5.4.3. FREEBIE APPLICATIONS

Benefiting Applications. Our architecture is of most benefit to ultra low power systems requiring bi-directional communication. We note that most devices require some form of connectivity not only to send data (like sensor samples), but also for configuration and firmware updates. Our architecture enables bi-directional communication on these severely energy-restricted devices and allows the MCU to switch off completely during idle periods, further reducing power consumption. Examples of such devices include hybrid (classical/smart) watches, IoT devices or on-body sensors—all operating on harvested energy. We chose two applications demonstrating our architecture capabilities: (i) a smartwatch using multiple BLE services to interact with the host; and (ii) firmware updates. Specifically, in the case of firmware updates, as the end device must request and receive firmware fragments from the host while the host transmits the firmware fragments and waits for reception confirmation—the firmware update is a stress-test of bi-directional BLE communication.

Battery-Free Smartwatch. We have developed a *battery-free smartwatch* based on two BLE services, (i) the Current Time Service (CTS) [31] and (ii) the Alert Notification Service (ANS) [30], operating on top of FreeBie hardware.

The smartwatch, see Figure 5.1, works as the ATT client [32, CS 5.3] of those two services. For the BLE host we have developed an application for the Android 11 OS [103] working as the ATT server of those two services. Once a connection is established, service discovery is executed to find the CTS and the ANS on the BLE host. Once successful, the smartwatch enables all notifications of both services. Then, the BLE host sends the current time to the smartwatch, which is later on updated independently of the host through the application process that runs every minute to increment time. In addition, the BLE host sends unread email notifications to the smartwatch which activates its on-screen email icon.

Firmware Update. We have also implemented the first *battery-free active-radio over-the-air firmware update.* Although successful demonstrations of battery-free over-the-air

reprogramming were done for backscatter-based nodes [289, 2], battery-free active-radio firmware updates (to the best of our knowledge) have never been demonstrated.[3]

We designed a custom BLE service for our firmware update application where FreeBie works as the ATT server of that service. Once connected, the BLE host should first send the firmware length and then initiate the update. Once initiated, FreeBie starts requesting a firmware section by sending the index of the section. Once FreeBie receives the requested firmware section, it writes this section to the corresponding address in the inactive firmware region. This process repeats until FreeBie obtains the complete firmware.

5.5. FREEBIE EVALUATION

5.5.1. EVALUATION SETUP

We evaluated both FreeBie applications at four light conditions in a controlled environment, 200 lx, 300 lx, 600 lx, 10 klx representing, respectively, (i) dark indoor light, (ii) dimmed indoor light, (iii) bright indoor light, and (iv) overcast day.

BLE Host Hardware and Software. For smartwatch, an Android application was built running on Google Pixel 3a [102] with Android 11 OS [103] acting as BLE host. For firmware update, a nRF52840 [218] development kit using SoftDevice [216] acts as BLE host. FreeBie is compared to Packetcraft [231] and SoftDevice [216]; comparison with checkpoint-based systems, e.g. [166], is impossible as *they do not work*, see Section 5.2 and Table 5.1.

BLE Advertising and Connection Parameters. The advertising interval of FreeBie is fixed at 2 s. For the hosts, per default Android 11 starts with a Connection Interval (CI) of 45 ms and 5 s Supervision Timeout (ST). For the firmware update host, we selected an initial 2 s CI and 32 s ST.

Controlled Test Environment. We put FreeBie at the bottom of a closed light box. A wirelessly-controlled LED bulb [163] was installed at the top of the box to create repeatable and controlled light source. A luminosity meter [302] was placed next to FreeBie mote to measure the exact luminosity projecting onto FreeBie's solar panels.

Long-term Evaluation. For a day-long evaluation we have collected luminosity values from a modern commercial smartwatch [82] worn on a wrist. Data was collected when the user[4] was performing (mostly outdoor) daily activities. The luminosity trace was then recreated in the controlled test environment described above.

Power Consumption Measurements. Connection event and sleep power consumption for Packetcraft [231], SoftDevice [216] and FreeBie's were measured with the X-NUCLEO-LPM01A [283] power consumption measurement board. FreeBie's power consumption whilst the processor domain is off was measured with the Keithley 2450 SMU [154]. Power consumption of Packetcraft and SoftDevice is measured on the NRF52840 development kit [218]; FreeBie power measurements are measured on the FreeBie mote. Further details of the evaluation setup are given in the artifact [73].

[3] The case study aims not to create a novel firmware update application [13, 25] but to evaluate FreeBie.

[4] The data collection was approved by the human ethics committee of the institution with which the authors of this study are affiliated with.

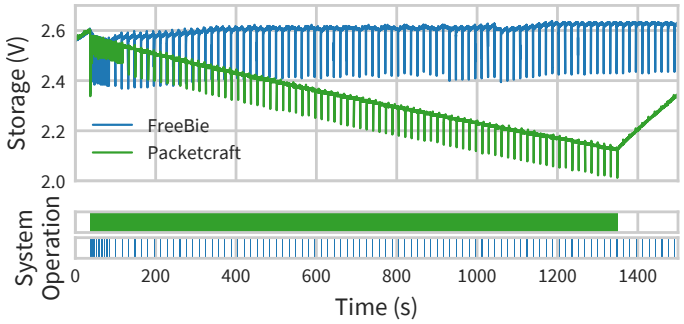


Figure 5.8: Example BLE connection retention on FreeBie hardware at 200 lx compared against Packetcraft [231]. The system operation bars (bar colour matches system on the 'Storage' plot) indicate when the system is on.

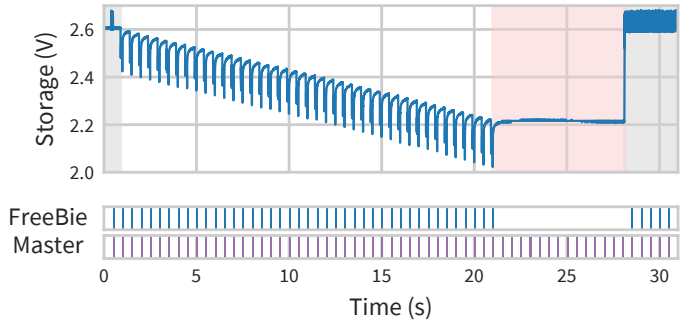


Figure 5.9: FreeBie power failure recovery. Despite missing several BLE packets FreeBie can recover the connection. : BLE network activity by the host, : FreeBie network activity, : FreeBie is actively powered, : FreeBie power failure.

5.5.2. FREEBIE EVALUATION RESULTS

BLE Connection Retention. First, to demonstrate that our system can sustain a BLE connection at intermittent power, we run a basic ≈30 min-long BLE connection. The result is presented in Figure 5.8. We clearly see that our system consumes less power operating intermittently and maintains the connection, while the default Packetcraft network stack [231] powers off below ≈2 V and never resumes the connection.

BLE Connection Recovery. To show that FreeBie can recover from power failures, we powered FreeBie from a stable power supply during a BLE connection. Then we turn off the power supply until FreeBie runs out of power. Then the power supply is resumed again, triggering the connection recovery. A snapshot of this process is presented in Figure 5.9. We clearly see that FreeBie can recover before the 32 s ST is reached.

Checkpoint and Restoration Overhead. First we measured code size for both FreeBie applications, split per process. The results are presented in Table 5.4. Our firmware application is small, therefore little is gained during a network-only power cycle by not restoring the application. During an application-only cycle, however, where the network process does not have to be restored, on average restoration is 21.30% faster compared to

Table 5.4: Code size of the FreeBie applications (in bytes).

	Smartwatch			Firmware update		
Process	**Data**	**BSS**	**Text**	**Data**	**BSS**	**Text**
Application	0	2368		0	12	
Network	376	5668		320	5168	
OS	292	2392		244	2452	
Total	668	10428	230120	564	7632	204797

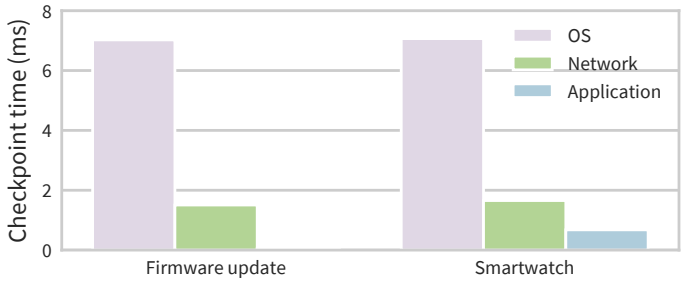


Figure 5.10: Average checkpoint time of each process for the two considered applications.

a naive checkpointing implementation where everything is restored. Due to the reduced overhead, with a more significant application—such as the smartwatch—not only the application-only cycles are 21.39% faster, but also the network-only cycles (by 7.76%) compared to the naive implementation. The average checkpointing times are depicted in Figure 5.10.

Smartwatch Evaluation. Figure 5.11 shows a 2.5 min long trace of smartwatch operation at each luminosity (except 200 lx, as the screen consumed to much power) from the initial connection establishment. Note that the operating times of FreeBie are very short, shown as a very thin green bar. Nonetheless we see that at each luminosity, FreeBie works despite long power-off intervals. The more energy is available—the smaller the off intervals—the bigger the responsiveness. Zooming in, the execution starts when the storage capacitor voltage reaches 2.6 V. When the system switches on, the voltage drops sharply due to the inrush current and the workload of initialisation but recovers afterwards.

After one round of advertising, FreeBie is connected with the Android BLE host. Note that since the connection was established, FreeBie was turned on continuously for a relatively long time (see 'On/Off' plots underneath the storage plot) and the voltage also dropped significantly. This is caused by Android's initial connection parameters preventing FreeBie from turning off. When the requested connection parameters are applied, FreeBie can start operating intermittently and turn off. For both 300 lx and 600 lx after 50 s and 25 s respectively, when all ATT services were configured and both BLE peripheral and BLE host started sending empty packets, SL is applied which further increases the off time.

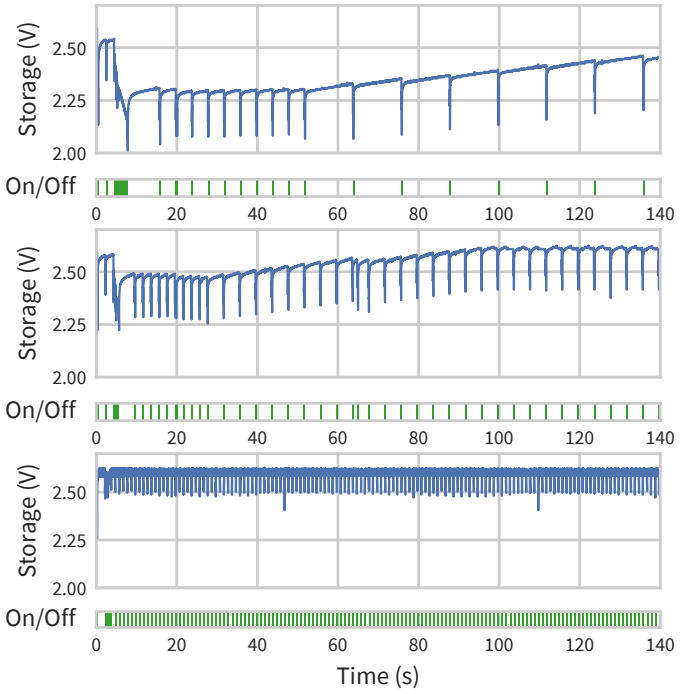


Figure 5.11: FreeBie smartwatch operation at three luminosities (top to bottom figure: 300 lx, 600 lx, 10 klx). Connection parameters for each luminosity are given in Table 5.3.

Firmware Update Evaluation. Figure 5.12 shows the execution of firmware update at 600 lx. Comparing this figure with Figure 5.11 (center) (smartwatch evaluation at the same luminosity) FreeBie starts more favourably due to the different initial connection parameters set by the host (as defined in Section 5.5.1), hence it can keep a higher storage voltage from the start. As expected, in an *application-only cycle* (A), no network process is restored or checkpointed. In *network-only cycle* (B) no application process is executed nor restored. In the *combined cycle* (C) after the network process execution, the system detects the application process pending execution in the near future. Hence the system sleeps until the scheduled time of the application execution, then dynamically loads the application and executes it, after which no processes are scheduled in the near future so the system checkpoints and turns off.

Power Consumption. We characterise FreeBie network-only cycles and compare the power consumption of FreeBie to (i) Packetcraft with low-frequency clock enabled and logging disabled, and to (ii) the proprietary Nordic Semiconductor's BLE stack, i.e., SoftDevice [216], at one CI value and four values of SL, i.e., 0, 1, 2, and 3. The results are presented in Figure 5.13.

Thanks to FreeBie's low power consumption when the processor domain is switched off (0.8352 µW) it is **9.5 times more efficient** compared to sleep mode (8.0172 µW with SoftDevice). FreeBie benefits from long connection intervals. Compared to Packetcraft,

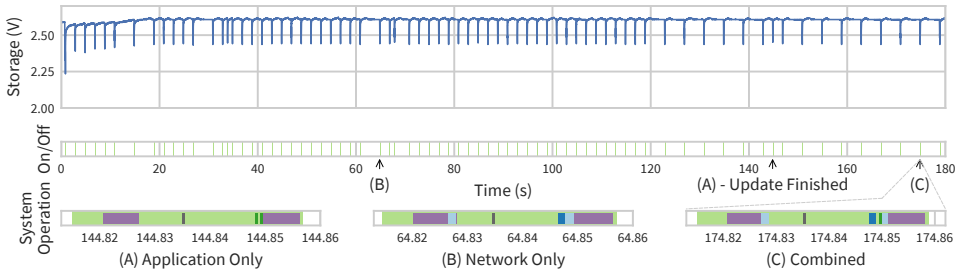


Figure 5.12: Evaluation of firmware update at 600 lx. Colour scheme: OS restore and checkpoint; on time; application restore and checkpoint; network restore and checkpoint; synchronisation point; network activity.

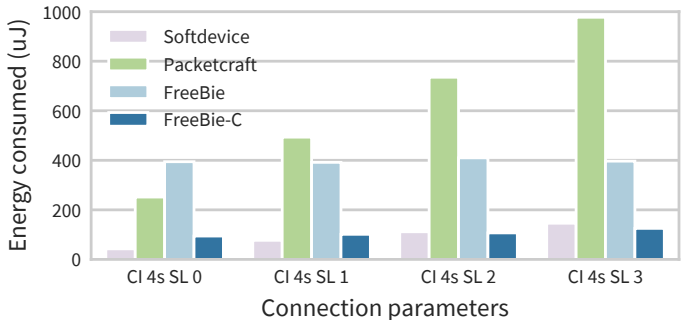


Figure 5.13: Consumed energy during one Connection Interval (CI), as shown in Figure 5.5, for four different Slave Latency (SL) values. FreeBie is compared against Packetcraft [231], SoftDevice [216] and a modified version of FreeBie (FreeBie-C) with external memory overhead excluded.

with a SL of 0 FreeBie's overhead of additional power consumption due to checkpointing and restoration is larger than the power saved by switching the MCU off. However, as the SL increases FreeBie starts to outperform Packetcraft. At a SL of 1 FreeBie already consumes less power compared to Packetcraft. Already with a SL of 3 *we are 2.46 times more energy efficient than the default Packetcraft stack.*

On the other hand, due to FRAM store and restore overhead and FreeBie's requirement to synchronise with the external RTC, FreeBie is not able to compete with SoftDevice's power consumption. Simply, utilising external FRAM consumes large amounts of power and is also slower than even a lower-clocked MCU with on-chip FRAM [296]. Ideally a MCU with on-board FRAM or MRAM and a ultra-low power RTC, e.g. the upcoming Ambiq Apollo 4 Blue [8], would remove this overhead almost completely. Therefore to make this comparison we have removed the external memory overhead from FreeBie traces denoting it as FreeBie-C, outperforming SoftDevice starting at a SL of 2.

Finally, we report the power consumption at each part of the network cycle, including checkpointing and restoration, as shown in Figure 5.5: FRAM read (restore checkpoint) (Ⓐ) is 10.26 mW, MCU sleep current (Ⓑ) is 2.41 mW, RX current (Ⓒ) is 18.86 mW, TX current (Ⓓ) is 19.35 mW, and FRAM write (checkpoint) (Ⓔ) is 12.03 mW.

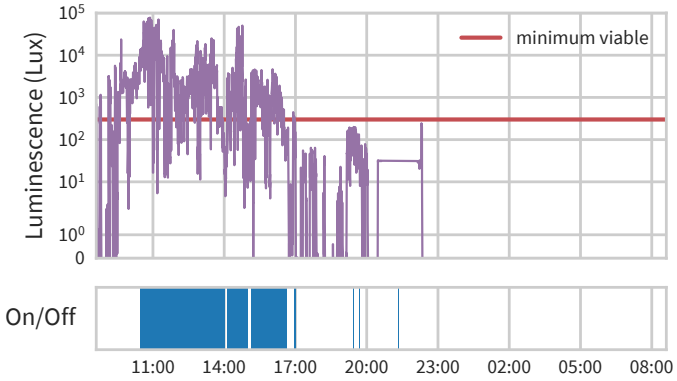


Figure 5.14: 24 hour-long operation of the FreeBie smartwatch.

Long-term Execution. Figure 5.14 shows 24-hour operation of the FreeBie smartwatch. We see that FreeBie is able to sustain a connection despite power interrupts for extended period of time (see 'On/Off' trace representing FreeBie activity, in particular between 11:00 and 17:00). Moreover, FreeBie is able to sustain a connection in varying energy availability: during the whole experiment the BLE link only had to reconnect seven times. If FreeBie receives more than 300 lx (minimum viable luminosity with the FreeBie LCD powered on) FreeBie is almost always on, only disconnecting when insufficient energy is provided for extended time. Note, if a full day operation of FreeBie is required then increasing the surface area of the solar panels would decrease the minimum viable luminosity threshold, see Figure 5.14 (top)—increasing the on time of the smartwatch.

5.6. DISCUSSION AND FUTURE WORK

Hardware Improvements. As shown in Figure 5.13, external FRAM and RTC limit the benefits of our architecture (both in terms of price, size and energy consumption). The next step is a FreeBie version build with next-generation System on Chip (SoC) such as [8] with on-chip MRAM, reducing FreeBie cost/size. More energy-efficient harvesters, such as [222], as part of a complete SoC would make FreeBie not only more efficient but also *potentially cheaper than battery-based systems.*

Battery-free Host. In our architecture only the end device is battery-free and intermittently-powered. The next research goal is an intermittently-powered host. The main research challenge would be integrating a synchronisation mechanism such as [97] into a fully battery-free architecture and efficient wake-up scheduling for end devices.

Delay-tolerant Networks. One might propose a delay-tolerant network as a solution to the wireless link intermittency problem [174, 143]. Sadly, considering point-to-point links, protocols such as BLE have strict timing requirements and do not allow any delay beyond what is specified by the standard.

Table 5.5: Comparison of relevant state-of-the-art BLE platforms.

	[59]	[314]	[181]	[273]	[84]	[51]	[334]	**FreeBie**[2]
Size	25 (⌀) × 5.5 mm	4 × 4 cm	20 mm (⌀) [7]	35 × 53 mm	1 × ≈1.5 cm[4]	≈70 × 55 mm [5]	—	2.54 × 2.54 mm
Capacitor size	0.2 F	Unknown	50 mF [8]	200 μF	N/A[3]	1 F [6]	N/A	15 mF
Radio chipset	CYBLE [60]	X-less radio	nRF51822 [219]	CC2650 [294]	Custom	CC2650 [294]	Custom	nRF52840 [220]
Minimum luminosity	100 lx	N/A	N/A	150 lx	N/A	Not reported	N/A	200 lx
Backscatter-based	No	No	No	No	Yes	No	Yes	No
Battery-free	Yes[1]	Yes	Yes	Yes	Yes[3]	Yes	No [3]	Yes
Intermittent	No[1]	Yes	No	No	No	Yes	No	Yes
Advertising only	No	No	No	No	Yes	—	Yes	No
Resumes connections	No	No	No	No	No	No	No	Yes

[1] Supercapacitor; [2] This work; [3] Actual implementation was continuously powered;
[4] Modulator only, logic driven by signal generator; [5] Size inferred from [51, Figure 7]: comparable in size to Arduino Uno board; [6] Maximum value of capacitor bank taken; [7] Excluding Powercast receiver [241];
[8] Unreported in the paper; a smallest value from Powercast receiver assumed.

5.7. RELATED WORK

Semi Battery-Free Wireless Networking. Augmenting battery-based IoT with backscatter tags was advocated in [237]. Such systems include [126, 249, 122, 94]. All these networks still need (i) a battery and (ii) a carrier generation source. A separate class of nodes are based on wake-up radios [239]. Wake-up radios still consume power when listening (which increases with receiver sensitivity [239, Figure 12]) and require the same infrastructure investment as backscatter-based radios. An example of battery-free sensors based on proprietary low-power wake-up radio technology is [85].

Battery-Free Bluetooth. All of the battery-free BLE nodes we are aware of do not operate intermittently when considering the BLE protocol itself. In each of such nodes one connection-less beacon transmission can be sent within a single capacitor charge from harvested energy; the recent examples of non-backscatter versions of such a systems are [39, 101, 90, 258, 147, 291] (academia) and [83, 59] (industry). Another (but less popular) approach for battery-free BLE is based on providing power wirelessly to the BLE nodes [181]. Except for our work no studies on intermittenly-powered BLE are presented beyond general calls for such system. A battery-free BLE node of similar hardware architecture to ours, i.e., an RTC-driven MCU with external FRAM, has been presented in [273]. The fundamental difference, however, is that [273] does not allow for state retention of the intermediate state of the BLE protocol (and other applications). Commercial implementations of battery-free BLE include [314]. Refer to Table 5.5 where a comparison of battery-free BLE platforms is given.

Battery-Free Wireless Networking. Battery-free networks include backscatter-based LoRa [151] and LTE [45]. An alternative approach focuses on active radios and includes [3,

97, 98, 250]. Yet another approach is to perform transformations of already existing protocols to have them failure-resilient [248]. Therein however it was assumed that a node with a power-off had its all memory flushed and needs to initialise from zero. The mathematical analysis of the channel capacity of a intermittent communication point-to-point link is given in [158]. Another way of providing energy to battery-free systems is based on wireless power transfer, recent examples include [204, 140].

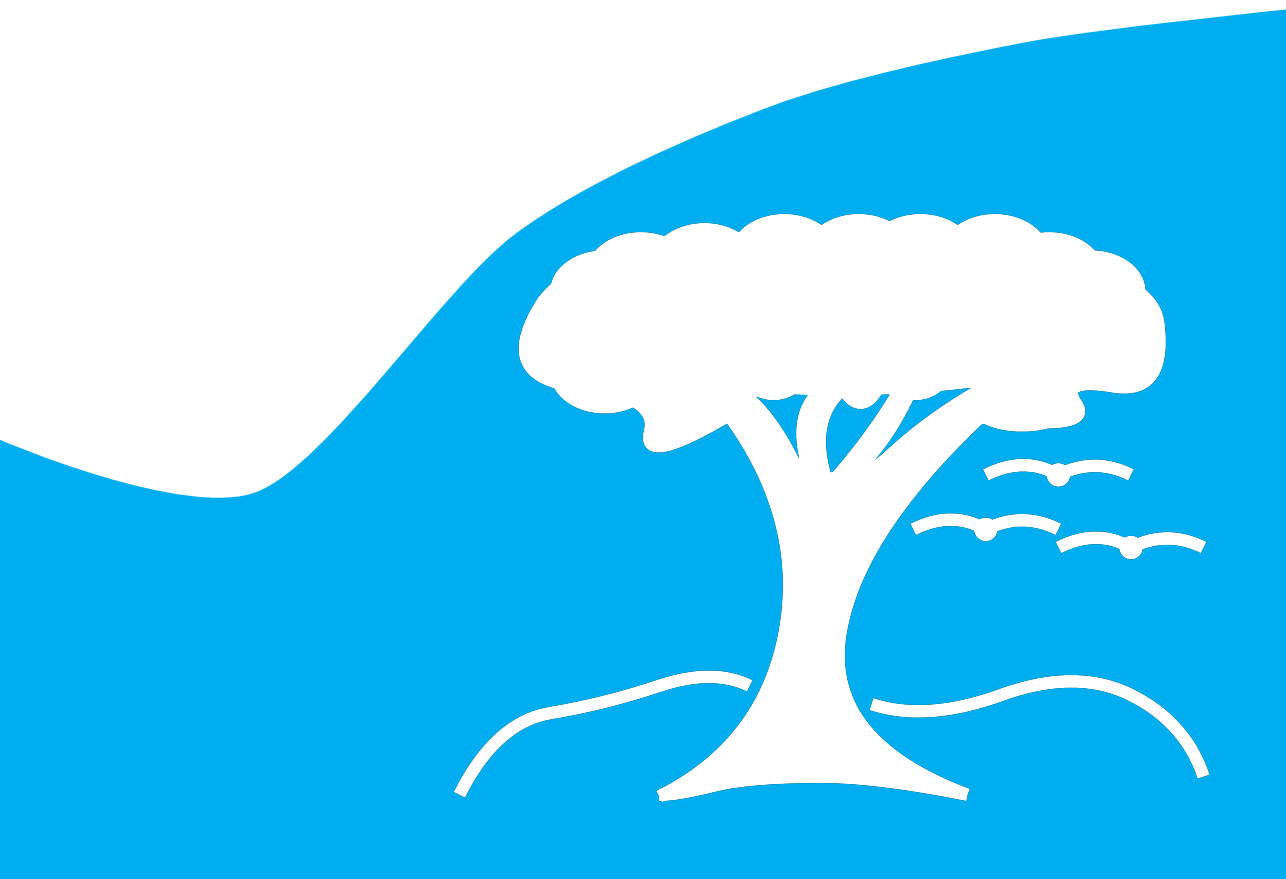
Initial studies of duty-cycled bi-directional communication on intermittent power for IEEE 802.15.4-compliant (i.e., non-Bluetooth) CC2420 radio [293] has been proposed in [322]. The same work also proposed to use low-power timing circuit to wake up system to exchange data with a neighbour [322, Figure 3]. Idea of custom protocol state preservation in FRAM has been presented in [38].

Intermittently-Powered Systems Software Frameworks. Software supporting intermittently-powered operation have already been comparatively presented in Table 5.1. Naturally, the list of such systems is only partial and we refer to extensive surveys presented in [21, Table 1], [325, Table 1].

5.8. CONCLUSIONS

We presented a new architecture enabling battery-free operation of a wireless communication protocol. Using this architecture and our real-time checkpointing mechanism we are able to sustain already established wireless connections despite power interruptions. The proposed architecture was used in developing FreeBie: the first truly intermittently-powered active Bluetooth Low Energy (BLE) system that is not based on connection-less transmissions, demonstrating sustained bi-directional communication and thus addressing the wireless networking challenge. The strength of our architecture is articulated by FreeBie consuming at least 9.5 times less power during device inactivity periods by switching-off when compared to sleep mode.

CHAPTER 6
CONCLUSION



The projected billions and trillions of IoT devices can form a significant burden to society in terms of e-waste. As most IoT devices are currently powered by batteries, these batteries need to be monitored and eventually properly recycled. Hence, the potential impact of battery-free devices on society is significant, especially since batteries are often the part with the lowest lifetime of any IoT device. Replacing the battery with energy harvested from ambient sources forms a more sustainable alternative, and can potentially lead to perpetual operation.

In this dissertation, we have addressed key challenges hindering adoption of battery-free devices such as: the design of interactive systems, debugging and testing of these systems, timekeeping and infrastructure-less wireless networking. We demonstrated the feasibility of battery-free devices through practical applications such as the battery-free smartwatch (Chapter 5) and the battery-free game console (Chapter 2). Throughout this dissertation we develop the required physical hardware and software to evaluate the performance of our contributions in practical **real-world** demonstrations. Next, we address the different challenges presented in this dissertation individually before addressing the overarching research question.

► **Interactive devices.** Human interaction with battery-free devices has largely remained an unexplored subject. Key questions such as are these user-facing systems even feasible or how to mitigate the effects of intermittency in user interaction still need to be addressed. In Chapter 2, we introduced the first intermittently-powered battery-free mobile gaming platform. Powered by ambient solar energy and by the interaction with the user itself using our multimodal energy harvesting mechanism. Beyond a fun toy, it demonstrates that battery-free interactive systems are possible. Due to intermittency, user input is ignored when the system is off, and the system can feel unresponsive. Hence fast start-up times and minimal overhead are crucial to pose a minimal burden to the user. To maintain progress efficiently despite intermittent power, we developed a new kind of checkpointing mechanism named MPatch. MPatch only stores the modified memory regions in between checkpoints, allowing for efficient checkpoints. We achieve this by tracking the modified memory regions using the MCUs onboard MPU. Checkpoints are triggered in a Just-In-Time fashion when the stored energy reaches a low threshold, and then checkpoints occur periodically. A double-buffered approach ensures that the system can always recover its state when power is lost, even if a checkpoint fails. Unlike traditional game saves, the entire system state is stored before a power failure, allowing the system to be restored to an identical state when power resumes.

► **Debugging and testing.** Our debugger DIPS as presented in Chapter 3 enables debugging of intermittent devices just like any normal embedded system. The debugger not only features standard hardware debugging functions such as breakpoints but can also emulate power traces to the Device Under Test (DUT). Unlike traditional debuggers, during regular operation, we automatically reconnect and restore debugging features such as breakpoints when power is re-established on the intermittent device. This mechanism allows for continuous debugging sessions in similar fashion to continuously-powered embedded systems. In detached debugging mode, the system will only connect when prompted, intended for when minimal interference by the debugger is desired. The

emulator not only has the ability to power the DUT through modes such as a square wave but also has the ability to simulate common intermittent device architectures featuring a boost-buck converter using its onboard measurement capability. On top of this platform, we build a testing framework. The framework is capable of automatically testing for common intermittent device problems such as lack of forward progress, memory consistency and peripheral restoration issues. Using the testing framework, we found bugs in state-of-the-art intermittent systems. In addition, user study participants found DIPS *easier to use, more intuitive and more familiar* when compared to the state-of-the-art. Demonstrating both DIPS' ease of use and the need for its debugging and testing functionality.

► **Keeping track of time.** Traditionally simple concepts such as keeping track of time pose challenges on intermittent systems. As timekeeping is a primitive to many IoT device functionalities including networking and scheduling, it is of crucial importance in any system. In Chapter 4 we presented a new kind of timekeeping mechanism, the Cascaded Hierarchical Remanence Timekeeper (CHRT). The mechanism utilizes cascaded RC circuits. These RC circuits can be quickly charged, reducing the start-up time compared to Real Time Clocks. Configurations with smaller capacitance allow for accurate short-term timekeeping with a granularity of milliseconds. Larger RC combinations keep track of time for longer durations at reduced accuracy. The mechanisms cascaded architecture automatically activates the next RC circuit when the active RC timekeeping circuit runs out, allowing the system to keep a continuous notion of time—enabling dynamic short-term and longer-term timekeeping requirements. Using the new timekeeper, we demonstrated a point-to-point wireless communication link between two intermittently powered nodes.

► **Wireless networking.** Integration into pre-existing wireless networks is crucial for battery-free devices to form a viable alternative to their battery-powered counterparts. In Chapter 5 we presented a new architecture enabling battery-free operation of wireless communication protocols, allowing the MCU to switch-off and seamlessly resume any connection when restored despite intermittent power. Central to the architecture is a real-time checkpointing mechanism that separates processes into individual checkpoints. Improving restoration and checkpointing times by avoiding restoration and checkpointing of unused process dependencies. The mechanism extends the real-time operating system scheduler, allowing it to checkpoint and switch the MCU off when idle. Due to this scheduler extension and our process based approach, pre-existing networking software stacks can easily be adapted to intermittent devices with little modification. We have demonstrated this by developing a battery-free smartwatch that can maintain a BLE connection with a smartphone despite just being powered by only solar energy. Not only does our mechanism maintain bi-directional BLE connections but it also saves power as idle power consumption is drastically reduced by switching off. Our choice for BLE is motivated by the fact that it is one of the most widely adopted low-power networking technologies. However, our architecture and mechanisms can also be applied to other wireless networks. Offering a lower power consumption and the benefits of using active radios such as longer range and infrastructure-less operation to battery-free devices.

RESEARCH QUESTION

Having addressed these challenges leads us back to the central research question:

What mechanisms must be developed and deployed in battery-free networked systems to enable connected and interactive IoT applications?

In this dissertation, we have shown that battery-free devices form a viable counterpart to their battery-based IoT devices. We have shown demonstrations of battery-free speed monitoring on a bike, interactive gaming consoles, and even a Bluetooth smartwatch. Through novel mechanisms, we have enabled the development and deployment of these devices, addressing critical challenges of intermittently-powered systems such as interactivity, debugging, timekeeping and networking.

The fundamental mechanisms that enabled these interactive and connected applications include: ❶ Our multimodal interactive energy harvesting and efficient checkpointing mechanisms, which enabled the development of the first intermittent gaming console. ❷ The debugging and testing mechanisms that allow for convenient debugging and testing of intermittent systems. ❸ Our timekeeping mechanisms which form an alternative to RTCs with faster start-up times and low power consumption, measuring time when the MCU is off. Finally, ❹ the real-time checkpointing mechanisms that save power and enable wireless connectivity within widely adopted networks such as BLE on intermittently-powered devices. Not only can the individual mechanisms be used on their own, but they can also be combined, allowing developers to choose mechanisms based on the application's requirements.

Our mechanisms not only enable the applications demonstrated in this dissertation but can be applied to turn a wide range of currently battery-powered IoT devices battery-free. Our ❶ interactive and ❹ connectivity mechanisms enable connected and interactive applications such as smart doorbells and scheduling panels for meeting rooms. ❹ Infrastructure-less connectivity and ❸ accurate timing are essential for easily deployable IoT sensing applications. Examples of these sensing applications include occupancy sensors, industrial machine fault monitors, asset trackers and crop monitors. However, all battery-free applications need to be ❷ debugged and tested during development.

We note that all of the demonstrations presented in this dissertation are based on **real-world** measurements of physically deployed devices and that everything required to reproduce the results presented in this dissertation is available to the community through artifacts [70, 71, 75, 74]. By improving the feasibility of battery-free devices, we hope to reduce the environmental impact of the future billions of IoT devices.

FUTURE WORK

In this section, we discuss future work in the context of the challenges. More specific discussion can be found in each of the corresponding chapters of this dissertation.

► **Interactive devices.** We see our work in Chapter 2 as a starting point for future research into interactive battery-free devices. In essence, interactive battery-free devices are challenging due to the intermittent nature of these systems, as they require (sometimes) continuous user interaction. In a society where everything executes and responds immediately and any delay between action and reaction is generally received negatively,

this is very noticeable with screen-based devices where a direct response is expected. In a game button presses should result in action immediately. Different applications could result in drastically different user experiences. Nudging the user to generate more energy by e.g. pressing buttons, could also result in a better user experience. Especially when incentivized with rewards, in the context of gaming this could be a power-up or extra lives. Hence, nudging techniques, interactive system architectures, suitable applications and user acceptance of intermittency all warrant further research.

► **Debugging and testing.** Testing and debugging of intermittent systems can be improved by expansion of our debugging/testing platform (Chapter 3). Even though our platform is able to detect when no forward progress is being made, it currently only prompts the user that an issue has occurred for the user to investigate. Further testing techniques common in software development can now be applied to intermittently-powered systems, e.g. using fuzzing to test for unintended behaviour. As intermittently-powered systems are closely tied to the power supply with JIT based systems, fuzzing techniques could also be used to modify the input power to the system itself.

► **Keeping track of time.** The topic of keeping track of time on intermittent systems also warrants additional research beyond integration of the Cascaded Hierarchical Remanence Timekeeper architecture (Chapter 4) into a chip. The architecture still lacks the ability for the MCU to remain powered off. The addition of ultra-low power circuitry to sample and to create a configurable wake-up signal to the processor would be a valuable addition.

► **Wireless networking.** Although our wireless networking architecture and mechanisms from Chapter 5 can be applied to other wireless networks such as Openthread, some key challenges still exist. The BLE Central is still continuously powered in our work. When considering a fully battery-less star network communicating using BLE, synchronization between the edge nodes and the central is of key importance as with power restrictions, continuous receiving on the central node is often not an option. Synchronization through environmental signals could drastically improve efficiency as the time with the radio on receiving to catch the advertisement to establish a connection is reduced. Alternatively, ultra-low power wake-up radios could form a solution for this problem. However, currently often the wake-up signal itself still requires significant power to generate.

► **Other mechanisms.** So far, we have discussed future work within the scope of this dissertation; when we broaden our scope, we can speculate on other mechanisms that could improve battery-free systems. Mechanisms such as more efficient energy harvesting, even lower power consuming MCUs and radios. Efforts in hardware consolidation could drastically reduce the size of current battery-free systems, enabling even more applications such as tiny battery-free smart pills, implantables or even tooth-mounted health monitors and earables. We envision that most battery-free system components can be consolidated in a single chip through the development of an energy harvesting fully non-volatile MCU where both the processor and the peripheral state is stored and restored in hardware.

ACKNOWLEDGEMENTS

This dissertation would not be possible without the support of many, which I would like to thank. First, I want to thank my mother, father and uncle, Tanja, Marcel and Albert, for their unconditional support. The results of my elementary school final exam ranked my chances of even reaching university between zero and three percent. Thanks to your faith in me and your support, I climbed my way up from a Bachelor's degree *cum laude* to a Master's degree *cum laude* and now my academic journey has culminated with this PhD dissertation. You have inspired and enabled me to pursue my dreams and overcome the many challenges on my way. For this, I will be forever grateful.

When I started my Master's thesis with my now promotor and supervisor, Przemysław, my work quickly became a foundation for a paper targeting a top academic venue. One issue, the deadline was in less than two months, and almost everything was still a concept at that time. Przemysław, your passion for research and work ethic is to be admired; it allowed us to meet this and many other (sometimes ambitious) deadlines. During my time at TU Delft, you gave me the freedom to work on my own ideas and you have inspired and enabled me to target the best venues. As writing always has been a challenge for me, by tirelessly reviewing my penmanship, you have tremendously improved my writing. Throughout my PhD you taught me how publish and present my research, both within academia and to the wider world, shaping me into the researcher I am today.

Koen, as my promotor, I always tremendously appreciated your feedback. I could pitch an idea to you, and based on your response, I knew almost instantly how the wider systems community would receive the idea. Secondly, your feedback on my dissertation turned my work into the book it is today. Unfortunately, we did not get around to writing that paper together, but you always made time for me when it mattered.

Next, I thank all my co-authors: Vito, Carlo, Abu, Bashima, Saad, Tom, Boris and Haozhe; it was a honour working with you. Josiah, thank you for your collaboration and all the in-depth brainstorming discussions and feedback. These brainstorming sessions with sometimes seemingly outlandish ideas have turned into amazing projects and papers. Sinan, it was a pleasure collaborating with you. I want to thank Fred from VodafoneZiggo and everyone from academia and industry involved with the ZERO P1 project for the valuable feedback on my research. In addition, I would like to extend my gratitude to Simon, Omar and André from NOWI, and Oren, Charlene and Mariusz from Ambiq for assisting in my research.

During my time at TU Delft I had the pleasure to share offices with James, Nikos, Vito and Vivian. Quickly we developed office traditions in the form of celebrating when our papers got accepted however, over the years this expanded into many occasions to celebrate. Life is short, hence there is always a reason to celebrate. Our technical, non-technical and water cooler conversations always were enjoyable. You provided me

with a sounding board and often by just listening would help me in finding a way to solve pressing issues.

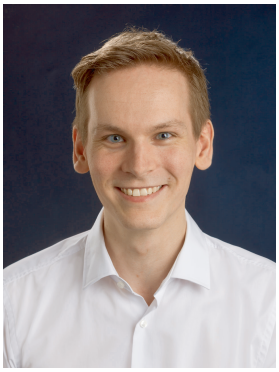
During the COVID-19 pandemic, Jorik and Vito introduced me to Counter-Strike. Even though everyone was locked indoors, our gaming endeavours (with Belma occasionally joining) always lifted my spirits. At the outset, I was terrible at the game; hence we often joked about reaching the highest rank in the game. I am happy to report that this year not only did Jorik and I achieve this goal, but our lively discussions in the process made my PhD much more enjoyable. Over time our gaming group evolved with the addition of Chenxing and Adrian, and with Fei and Selina occasionally joining our events. Chenxing and Fei, thank you for hosting most of our events and for the privilege of naming Tiptoe. All events, from games and dinner to birthdays, are always a blast thanks to all of you, Adrian, Belma, Chenxing, Fei, Jorik, Selina and Vito.

As my research spans both software and *hardware*, over the years I have submitted countless purchase orders for components. Thanks to our secretaries, Kim and Minaksie, for tirelessly approving all of these orders.

Finally, I would like to express my gratitude to all my colleagues at the Embedded and Networked Systems groups. You have made my time at TU Delft even more enjoyable.

Curriculum Vitæ

Jasper de Winkel



Jasper de Winkel was born in Wageningen, the Netherlands, on the 29th of April 1994. From a young age, he was passionate about technology and this passion further refined into his interest in embedded systems. Following this passion, he completed the shortened track of the Bachelor Embedded Systems Engineering (2014-2017) *cum laude* at the HAN University of Applied Sciences. Followed by the Master Embedded Systems (2017-2019), specialising in Software and Networking at the Delft University of Technology. During his time in Delft, he was introduced to the concept of battery-free and intermittently-powered systems that led to his Master's thesis and completing the Master's degree *cum laude*.

His MSc research led to a PhD position in the Embedded Systems Group at the Delft University of Technology, sponsored by NWO within the ZERO P1 project. He focused on reducing the environmental impact of the Internet of Things by operating solely on harvested energy without batteries, leading towards the next generation of more sustainable IoT devices. Within the domain of battery-free systems, he explored novel device architectures and ultra-low-power wireless networking, connecting software and hardware into novel systems and architectures. His research received the distinguished paper award at Ubicomp/IMWUT, was a candidate for the best paper award at SenSys and was covered by international media, including the Wall Street Journal, The Independent, The Verge, Hackaday and CNET.

List of Publications

7. Abu Bakar, Rishabh Goel, **Jasper de Winkel**, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. 2023. Protean: Adaptive Hardware-Accelerated Intermittent Computing. ACM GetMobile: Mobile Comp. and Comm. 27, 1 (March 2023), 5–10 [20]

6. Abu Bakar, Rishabh Goel, **Jasper de Winkel**, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. 2022. Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-free Computing. In Proc. SenSys. ACM, Boston, MA, USA, 207–221 [19]

5. **Jasper de Winkel**, Tom Hoefnagel, Boris Blokland, and Przemysław Pawełczak. 2022. DIPS: Debug Intermittently-Powered Systems Like Any Embedded System. In Proc. SenSys. ACM, Boston, MA, USA, 222–235 [65] *(Best Paper Candidate)*

4. **Jasper de Winkel**, Haozhe Tang, and Przemysław Pawełczak. 2022. Intermittently-Powered Bluetooth That Works. In Proc. MobiSys. ACM, Portland, OR, USA, 287–301 [68]

3. **Jasper de Winkel**, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. 2021. Battery-Free Game Boy: Sustainable Interactive Devices. ACM GetMobile: Mobile Comp. and Comm. 25, 2 (June 2021), 22–26 [67]

2. **Jasper de Winkel**, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. 2020. Battery-Free Game Boy. ACM Interact. Mob. Wearable Ubiquitous Technol. 4, 3 (September 2020), 111:1–111:34 [66] *(Distinguished Paper Award)*

1. **Jasper de Winkel**, Carlo Delle Donne, Kasım Sinan Yıldırım, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable Timekeeping for Intermittent Computing. In Proc. ASPLOS. ACM, Lausanne, Switzerland, 53–67 [64]

REFERENCES

[1] Henko Aantjes, Amjad Y. Majid, and Przemysław Pawełczak. 2016. A Testbed for Transiently Powered Computers. https://arxiv.org/abs/1606.07623.

[2] Henko Aantjes, Amjad Y. Majid, Przemysław Pawełczak, Jethro Tan, Aaron Parks, and Joshua R. Smith. 2017. Fast downstream to many (computational) RFIDs. In *Proc. INFOCOM*. IEEE, Atlanta, GA, USA, 1–9. https://doi.org/10.1109/INFOCOM.2017.8056987.

[3] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the MithræUm of Circus Maximus. In *Proc. SenSys*. ACM, Virtual Event, 368–381. https://doi.org/10.1145/3384419.3430722.

[4] Saad Ahmed, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, Naveed Anwar Bhatti, and Luca Mottola. 2018. Poster Abstract: Towards Smaller Checkpoints for Better Intermittent Computing. In *Proc. IPSN* (November 11–13). ACM/IEEE, Porto, Portugal, 132–133. https://doi.org/10.1109/IPSN.2018.00029.

[5] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. Effient Intermittent Computing with Differential Checkpointing. In *Proc. LCTES*. ACM, Phoenix, AZ, USA, 70–81. https://doi.org/10.1145/3316482.3326357.

[6] Sultan A. Alharthi, Olaa Alsaedi, Zachary O. Toups, Joshua Tanenbaum, and Jessica Hammer. 2018. Playing to Wait: A Taxonomy of Idle Games. In *Proc. CHI* (April 21–26). ACM, Montréal, QC, Canada, 210:1–210:13. https://doi.org/10.1145/3173574.3174195.

[7] Ambiq Micro Inc. 2018. Apollo3 Blue Ultra-Low Power Microcontroller. https://ambiqmicro.com/static/mcu/files/Apollo3_Blue_MCU_Data_Sheet_v0_11_0.pdf. Last accessed: Apr. 25, 2020.

[8] Ambiq Micro, Inc. 2018. Apollo4 Blue Ultra-Low Power Microcontroller. https://ambiq.com/apollo4-blue/. Last accessed: Sep. 8, 2021.

[9] Ambiq Micro, Inc. 2021. Artasie AM1815 Real-Time Clock. https://ambiq.com/artasie-am1815. Last accessed: Sep. 8, 2021.

[10] Brian Amos. 2020. *Hands-On RTOS with Microcontrollers: Building Real-Time Embedded Systems using FreeRTOS, STM32 MCUs, and SEGGER Debug Tools.* Packt Publishing Limited, Birmingham, United Kingdom.

[11] Apache Software Foundation. 2021. Apache Mynewt Operating System Bluetooth Stack Source Code Repository. https://github.com/apache/mynewt-nimble. Last accessed: Aug. 5, 2021.

[12] Apple Inc. 2023. Apple Watch Series 8. https://www.apple.com/apple-watch-series-8/. Last accessed: January 24, 2023.

[13] Konstantinos Arakadakis, Pavlos Charalampidis, Antonis Makrogiannakis, and Alexandros Fragkiadakis. 2022. Firmware Over-the-Air Programming Techniques for IoT networks - A Survey. *ACM Comput. Surv.* 54, 9 (October 2022), 1–36. https://doi.org/10.1145/3472292.

[14] Rauf Arif. 2021. With An Economic Potential Of $11 Trillion, Internet Of Things Is Here To Revolutionize Global Economy. Forbes, https://www.forbes.com/sites/raufarif/2021/06/05/with-an-economic-potential-of-11-trillion-internet-of-things-is-here-to-revolutionize-global-economy. Last accessed: Jul. 6, 2021.

[15] Arm Limited. 2020. GNU Arm Embedded Toolchain. https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads. Last accessed: Aug. 19, 2021.

[16] Arm Limited. 2021. Mbed Cordio BLE Solution Official Website. https://os.mbed.com/docs/mbed-cordio. Last accessed: Aug. 5, 2021.

[17] Nivedita Arora, Steven L. Zhang, Fereshteh Shahmiri, Diego Osorio, Yicheng Wang, Mohit Gupta, Zhengjun Wang, Thad Eugene Starner, Zhonglin Wang, and Gregory D. Abowd. 2018. SATURN: A Thin and Flexible Self-powered Microphone Leveraging Triboelectric Nanogenerator. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 2 (June 2018), 60:1–60:28. https://doi.org/10.1145/3214263.

[18] Alberto Rodriguez Arreola, Domenico Balsamo, Geoff V. Merrett, and Alex S. Weddell. 2018. RESTOP: Retaining External Peripheral State in Intermittently-powered Sensor Systems. *Sensors* 18, 1 (2018), 172. https://doi.org/10.3390/s18010172.

[19] Abu Bakar, Rishabh Goel, Jasper de Winkel, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. 2022. Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-free Computing. In *Proc. SenSys*. ACM, Boston, MA, USA, 207–221. https://doi.org/10.1145/3560905.3568561.

[20] Abu Bakar, Rishabh Goel, Jasper de Winkel, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. 2023.

Protean: Adaptive Hardware-Accelerated Intermittent Computing. *GetMobile: Mobile Comp. and Comm.* 27, 1 (May 2023), 5–10. https://doi.org/10.1145/3599184.3599186.

[21] Abu Bakar, Alexander G. Ross, Kasım Sinan Yıldırım, and Josiah Hester. 2021. REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 3 (September 2021), 87:1–87:42. https://doi.org/10.1145/3478077.

[22] Domenico Balsamo, Alex S. Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M. Al-Hashimi, Geoff V. Merrett, and Luca Benini. 2016. Hibernus++: a Self-calibrating and Adaptive System for Transiently-powered Embedded Devices. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 35, 12 (2016), 1968–1980. https://doi.org/10.1109/TCAD.2016.2547919.

[23] Abhijeet Banerjee, Sudipta Chattopadhyay, and Abhik Roychoudhury. 2016. On Testing Embedded Software. *Advances in Computers* 101 (2016), 121–153. https://doi.org/10.1016/bs.adcom.2015.11.005.

[24] Soumya C. Barathi, Daniel J. Finnegan, Matthew Farrow, Alexander Whaley, Pippa Heath, Jude Buckley, Peter W. Dowrick, Burkhard C. Wünsche, James L. J. Bilzon, Eamonn O'Neill, and Christof Lutteroth. 2018. Interactive Feedforward for Improving Performance and Maintaining Intrinsic Motivation in VR Exergaming. In *Proc. CHI* (April 21–26). ACM, Montréal, QC, Canada, 408:1–408:14. https://doi.org/10.1145/3173574.3173982.

[25] Jan Bauwens, Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, and Eli De Poorter. 2020. Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles. *IEEE Commun. Mag.* 58, 2 (February 2020), 35–41. https://doi.org/10.1109/MCOM.001.1900125.

[26] Uwe Becker. 2017. Gameboy-Emulator per STM32F746 (in German). https://mikrocontroller.bplaced.net/wordpress/?page_id=1290. Last accessed: May 5, 2020.

[27] Anil Bhaskar. 2022. How IoT Is Transforming The Manufacturing Industry. Forbes, https://www.forbes.com/sites/forbestechcouncil/2022/09/28/how-iot-is-transforming-the-manufacturing-industry/. Last accessed: Jan. 24, 2023.

[28] Naveed Bhatti and Luca Mottola. 2017. HarvOS: Efficient Code Instrumentation for Transiently-powered Embedded Devices. In *Proc. IPSN* (April 18–20). ACM/IEEE, Pittsburgh, PA, USA, 209–219. https://doi.org/10.1145/3055031.3055082.

[29] Eli Blevis. 2007. Sustainable Interaction Design: Invention & Disposal, Renewal & Reuse. In *Proc. CHI* (April 28 – May 3). ACM, San Jose, CA, USA, 503–512. https://doi.org/10.1145/1240624.1240705.

[30] Bluetooth Special Interest Group, Inc. 2021. Bluetooth Alert Notification Service. https://www.bluetooth.com/specifications/specs/alert-notification-service-1-0/. Last accessed: Sept. 14, 2021.

[31] Bluetooth Special Interest Group, Inc. 2021. Bluetooth Current Time Service. https://www.bluetooth.com/specifications/specs/current-time-service-1-1/. Last accessed: Sept. 14, 2021.

[32] Bluetooth Special Interest Group, Inc. 2021. Bluetooth Specifications List. https://www.bluetooth.com/specifications/specs. Last accessed: Aug. 8, 2021.

[33] Bosch Sensortec. 2021. BMA400 Triaxial Ultra-low Power Acceleration Sensor. https://www.bosch-sensortec.com/products/motion-sensors/accelerometers/bma400. Last accessed: Sep. 9, 2021.

[34] Adriano Branco, Luca Mottola, Muhammad Hamad Alizai, and Junaid Haroon Siddiqui. 2019. Intermittent Asynchronous Peripheral Operations. In *Proc. SenSys* (November 10–13). ACM, New York City, NY, USA, 55–67. https://doi.org/10.1145/3356250.3360033.

[35] Rodney Brooks. 2021. The Battery Revolution Is Just Getting Started. IEEE Spectrum, https://spectrum.ieee.org/energy/batteries-storage/the-battery-revolution-is-just-getting-started.

[36] Lars Büthe, Michael Hardegger, Patrick Brulisauer, and Gerhard Tröster. 2014. RFID-Die: Battery-free Orientation Sensing Using an Array of Passive Tilt Switches. In *Proc. UbiComp Adjunct*. ACM, Seattle, WA, USA, 215–218. https://doi.org/10.1145/2638728.2638733.

[37] Cadence Design Systems, Inc. 2016. Cadence Circuit Design. https://www.cadence.com/content/cadence-www/global/en_US/home/tools/custom-ic-analog-rf-design/circuit-design.html. Last accessed: Jan. 19, 2020.

[38] Bradford Campbell, Meghan Clark, Samuel DeBruin, Branden Ghena, Neal Jackson, Ye-Sheng Kuo, and Prabal Dutta. 2016. Perpetual Sensing for the Built Environment. https://doi.org/10.1109/mprv.2016.66. *Pervasive Computing* 15, 4 (Oct.–Dec. 2016), 45–55.

[39] Carlo Signer. 2017. *Batteryless Bluetooth Low Energy Communication*. Bachelor's Thesis. ETHZ, Switzerland. https://pub.tik.ee.ethz.ch/students/2017-FS/BA-2017-03.pdf.

[40] Ricardo C. Carrano, Diego Passos, Luiz C. S. Magalhães, and Célio V. N. Albuquerque. 2014. Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor Networks. *IEEE Commun. Surv. Tutorials* 16, 1 (First Quarter 2014), 181–194. https://doi.org/10.1109/SURV.2013.052213.00116.

[41] CD Projekt. 2020. European Union Projects. https://www.cdprojekt.com/en/capital-group/eu-projects. Last accessed: Apr. 28, 2020.

[42] Arunkumar Chandrasekhar, Gaurav Khandelwa, Nagamalleswara Rao Alluri, Venkateswaran Vivekananthan, and Sang-Jae Kim. 2018. Battery-Free Electronic Smart Toys: A Step toward the Commercialization of Sustainable Triboelectric Nanogenerators. *Sustainable Chemistry and Engineering* 6, 5 (April 2018), 6110–6116. https://doi.org/10.1021/acssuschemeng.7b04769.

[43] Arunkumar Chandrasekhar, Gaurav Khandelwal, Nagamalleswara Rao Alluri, Venkateswaran Vivekananthan, and Sang-Jae Kim. 2017. Sustainable Biomechanical Energy Scavenger toward Self-Reliant Kids' Interactive Battery-Free Smart Puzzle. *Sustainable Chemistry and Engineering* 5, 8 (June 2017), 7310–7316. https://doi.org/10.1021/acssuschemeng.7b01561.

[44] Tzuwen Chang, Neng-Hao Yu, Sung-Sheng Tsai, Mike Y. Chen, and Yi Ping Hung. 2012. Clip-on Gadgets: Expandable Tactile Controls For Multi-touch Devices. In *Proc. MobileHCI* (September 21–24). ACM, San Francisco, CA, USA, 163–166. https://doi.org/10.1145/2371664.2371699.

[45] Zicheng Chi, Xin Liua, Wei Wang, Yao Yao, and Ting Zhu. 2020. Leveraging Ambient LTE Traffic for Ubiquitous Passive Communication. In *Proc. SIGCOMM*. ACM, Virtual Event, 172–185. https://doi.org/10.1145/3387514.3405861.

[46] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. 2019. Achieving Stagnation-Free Intermittent Computation with Boundary-Free Adaptive Execution. In *Proc. RTAS*. IEEE, Montréal, QC, Canada, 331–344. https://doi.org/10.1109/RTAS.2019.00035.

[47] Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler Directed Speculative Intermittent Computation. In *Proc. MICRO*. ACM, Columbus, OH, USA, 399–412. https://doi.org/10.1145/3352460.3358279.

[48] Yohan Chon, Gwangmin Lee, Rhan Ha, and Hojung Cha. 2016. Crowdsensing-based Smartphone Use Guide for Battery Life Extension. In *Proc. UbiComp*. ACM, Heidelberg, Germany, 958–969. https://doi.org/10.1145/2971648.2971728.

[49] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson Sample. 2016. An Energy-interference-free Hardware/Software Debugger for Intermittent Energy-harvesting Systems. In *Proc. ASPLOS*. ACM, Atlanta, GA, USA, 577–589. https://doi.org/10.1145/2980024.2872409.

[50] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proc. OOPSLA* (Oct. 30 – Nov. 4). ACM, Amsterdam, The Netherlands, 514–530. https://doi.org/10.1145/2983990.2983995.

[51] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proc. ASPLOS*. ACM, Williamsburg, VA, USA, 767–781. https://doi.org/10.1145/3173162.3173210.

[52] Alexei Collin and Brandon Lucia. 2018. Termination Checking and Task Decomposition for Task-Based Intermittent Programs. In *Proc. CC* (February 24–25). ACM, Vienna, Austria, 183:1–183:31. https://dl.acm.org/doi/10.1145/3360609.

[53] Rodrigo Copetti. 2019. Architecture of Consoles: GameBoy. https://copetti.org/projects/consoles/game-boy. Last accessed: May 3, 2020.

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

[55] Microsoft Corp. 2022. Visual Studio Code. https://code.visualstudio.com.

[56] Taiyo Yuden Corp. 2021. EYSKBNZWB BLE Wireless Module. https://www.yuden.co.jp/eu/product/category/module/bluetooth/EYSKBNZWB.html. Last accessed: Sep. 9, 2021.

[57] Corporate Knights. 2020. 2020 Global 100 Ranking: Index of the World's Most Sustainable Corporations. https://www.corporateknights.com/reports/2020-global-100. Last accessed: Apr. 30, 2020.

[58] Bandai Corporation. 1982. LCD Solarpower Handheld Electronic Games Series. https://en.wikipedia.org/wiki/Bandai_LCD_Solarpower. Last accessed: Sep. 22, 2020.

[59] Cypress Semiconductor Corp. 2020. CYALKIT-E02 Solar-Powered BLE Sensor Beacon Reference Design Kit. https://www.cypress.com/documentation/development-kitsboards/cyalkit-e02-solar-powered-ble-sensor-beacon-reference-design. Last accessed: Aug. 4, 2021.

[60] Cypress Semiconductor Corp. 2021. CYBLE-022001-00 BLE Module. https://www.cypress.com/documentation/datasheets/cyble-022001-00-ez-ble-creator-module. Last accessed: Aug. 10, 2021.

[61] Al Danial. 2021. cloc - Count Lines of Code Source Code Repository. https://github.com/AlDanial/cloc. Last accessed: Aug. 10, 2021.

[62] Marc de Kruijf and Karthikeyan Sankaralingam. 2013. Idempotent Code Generation: Implementation, Analysis, and Evaluation. In *Proc. CGO*. ACM/IEEE, Shenzhen, China, 1–12. https://doi.org/10.1109/CGO.2013.6495002.

[63] Jasper de Winkel. 2019. *Keeping Track of Time on Energy Harvesting Systems*. Master's thesis. Delft University of Technology, Delft, The Netherlands. http://resolver.tudelft.nl/uuid:3ed18e0b-03a2-4496-a761-af65d191e135.

[64] Jasper de Winkel, Carlo Delle Donne, Kasım Sinan Yıldırım, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable Timekeeping for Intermittent Computing. In *Proc. ASPLOS*. ACM, Lausanne, Switzerland, 53–67. https://doi.org/10.1145/3373376.3378464.

[65] Jasper de Winkel, Tom Hoefnagel, Boris Blokland, and Przemysław Pawełczak. 2022. DIPS: Debug Intermittently-Powered Systems Like Any Embedded System. In *Proc. SenSys*. ACM, Boston, MA, USA, 222–235. https://doi.org/10.1145/3560905.3568543.

[66] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. 2020. Battery-Free Game Boy. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3 (September 2020), 111:1–111:34. https://doi.org/10.1145/3411839.

[67] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. 2021. Battery-Free Game Boy: Sustainable Interactive Devices. *GetMobile: Mobile Comp. and Comm.* 25, 2 (September 2021), 22–26. https://doi.org/10.1145/3486880.3486888.

[68] Jasper de Winkel, Haozhe Tang, and Przemysław Pawełczak. 2022. Intermittently-Powered Bluetooth That Works. In *Proc. MobiSys*. ACM, Portland, OR, USA, 287–301. https://doi.org/10.1145/3498361.3538934.

[69] Delft University of Technology Sustainable Systems Lab. 2019. Botoks and CHRT Source Code Repository and Website. https://github.com/tudssl/botoks. Last accessed: Jan. 18, 2020.

[70] Delft University of Technology Sustainable Systems Lab. 2020. Botoks and CHRT Artifact. https://doi.org/10.5281/zenodo.3612619. Last accessed: Mar. 9, 2023.

[71] Delft University of Technology Sustainable Systems Lab. 2020. ENGAGE Artifact. https://doi.org/10.5281/zenodo.7684869. Last accessed: Mar. 9, 2023.

[72] Delft University of Technology Sustainable Systems Lab. 2020. ENGAGE Open Source Repository. https://github.com/tudssl/engage. Last accessed: Jul. 22, 2020.

[73] Delft University of Technology Sustainable Systems Lab. 2021. FreeBie Source Code Repository: Hardware and Software. https://github.com/tudssl/FreeBie. Last accessed: Apr. 26, 2022.

[74] Delft University of Technology Sustainable Systems Lab. 2022. DIPS Artifact. https://doi.org/10.5281/zenodo.7938463. Last accessed: Mar. 9, 2023.

[75] Delft University of Technology Sustainable Systems Lab. 2022. FreeBie Artifact. https://doi.org/10.5281/zenodo.6583724. Last accessed: Mar. 9, 2023.

[76] Carlo Delle Donne. 2018. *Wake-Up Alignment for Batteryless Sensors with Zero-Energy Timekeeping*. Master's thesis. Delft University of Technology, Delft, The Netherlands. http://resolver.tudelft.nl/uuid:e871f33e-7ed2-452b-a962-1de2af9a2906.

[77] Carlo Delle Donne, Kasım Sinan Yıldırım, Amjad Yousef Majid, Josiah Hester, and Przemysław Pawełczak. 2018. Backing Out of Backscatter for Intermittent Wireless Networks. In *Proc. ENSsys* (November 3). ACM, Shenzhen, China, 38–40. https://doi.org/10.1145/3279755.3279758.

[78] Maurizio Di Paolo Emilio and Roy Anirban. 2021. Macro Environmental Effect of Micro Energy Harvesting. https://www.powerelectronicsnews.com/macro-environmental-effect-of-micro-energy-harvesting. Last accessed: Jul. 27, 2021.

[79] Christine Dierk, Molly Jane Pearce Nicholas, and Eric Paulos. 2018. AlterWear: Battery-Free Wearable Displays for Opportunistic Interactions. In *Proc. CHI* (April 21–26). ACM, Montréal, QC, Canada, 210:1–210:13. https://doi.org/10.1145/3173574.3173794.

[80] Economist Intelligence Unit. 2020. The IoT Business Index 2020: a Step Change in Adoption. https://learn.arm.com/rs/714-XIJ-402/images/economist-iot-business-index-2020-arm.pdf. Last accessed: May 7, 2020.

[81] Kristina Edström (Executive Publisher). 2020. Horizon 2020 EU Program Battery 2030+: Inventing the Sustainable Batteries of the Future: Research Needs and Future Actions. https://battery2030.eu/digitalAssets/861/c_861350-l_1-k_roadmap-27-march.pdf. Last accessed: Jul. 7, 2021.

[82] Samsung Electronics. 2021. Galaxy Watch4 Smartwatch. https://www.samsung.com/us/watches/galaxy-watch4/. Last accessed: Dec. 14, 2021.

[83] EnOcean GmbH. 2020. STM 550B Multisensor Module (BLE) with NFC Interface. https://www.enocean.com/en/products/enocean_modules_24ghz_ble/stm-550b-multisensor-module. Last accessed: Aug. 4, 2021.

[84] Joshua F. Ensworth and Matthew S. Reynolds. 2015. Every Smart Phone is a Backscatter Reader: Modulated Backscatter Compatibility with Bluetooth 4.0 Low Energy (BLE) Devices. In *Proc. RFID*. IEEE, San Diego, CA, USA, 78–85. https://doi.org/10.1109/RFID.2015.7113076.

[85] Everactive. 2021. Batteryless Eversensors. https://everactive.com/batteryless-technology. Last accessed: Aug. 5, 2021.

[86] Xenofon Fafoutis and Nicola Dragoni. 2011. ODMAC: An on-demand MAC Protocol for Energy Harvesting-Wireless Sensor Networks. In *Proc. PE-WASUN* (November 3–4). ACM, Miami, FL, USA, 49–56. https://doi.org/10.1145/2069063.2069072.

[87] Xiaoran Fan, Han Ding, Sugang Li, Michael Sanzari, Yanyong Zhang, Wade Trappe, Zhu Han, and Richard E. Howard. 2018. Energy-Ball: Wireless Power Transfer for Batteryless Internet of Things through Distributed Beamforming. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 2 (June 2018), 65:1–65:22. https://doi.org/10.1145/3214268.

[88] Duarte Fernandes, André G. Ferreira, Reza Abrishambaf, José Mendes, and Jorge Cabral. 2018. Survey and Taxonomy of Transmissions Power Control Mechanisms for Wireless Body Area Networks. *IEEE Commun. Surv. Tutorials* 20, 2 (Second Quarter 2018), 1292–1328. https://doi.org/10.1109/COMST.2017.2782666.

[89] Fluke. 2020. Fluke 87V Industrial Multimeter. https://www.fluke.com/en-us/product/electrical-testing/digital-multimeters/fluke-87v. Last accessed: Jun. 22, 2020.

[90] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh K. Gupta. 2018. Pible: Battery-Free Mote for Perpetual Indoor BLE Applications. In *Proc. BuildSys*. ACM, Shenzen, China, 168–171. https://doi.org/10.1145/3276774.3282822.

[91] Free Software Foundation, Inc. 2021. lwIP - A Lightweight TCP/IP stack Website. https://savannah.nongnu.org/projects/lwip. Last accessed: Aug. 10, 2021.

[92] Fujitsu Semiconductor Limited. 2018. MB85RS4MT 4 MB FRAM Memory with SPI Interface. https://www.fujitsu.com/global/documents/products/devices/semiconductor/fram/lineup/MB85RS4MT-DS501-00053-1v0-E.pdf. Last accessed: Jun. 13, 2021.

[93] Fujitsu Semiconductor Ltd. 2018. MB85RS4MT 512 KB SPI FRAM. https://www.fujitsu.com/uk/Images/MB85RS4MT.pdf. Last accessed: Apr. 25, 2020.

[94] Ander Galisteo, Ambuj Varshney, and Domenico Giustiniano. 2020. Two to Tango: Hybrid Light and Backscatter Networks for Next Billion Devices. In *Proc. MobiSys*. ACM, Toronto, ON, Canada, 80–93. https://doi.org/10.1145/3386901.3388918.

[95] Karthik Ganesan, Joshua San Miguel, and Natalie Enright Jerger. 2019. The What's Next Intermittent Computing Architecture. In *Proc. HPCA*. IEEE, Washington, DC, USA, 211–223. https://doi.org/10.1109/HPCA.2019.00039.

[96] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. 2019. Shepherd: a Portable Testbed for the Batteryless IoT. In *Proc. SenSys*. ACM, New York, NY, USA, 83–95. https://doi.org/10.1145/3356250.3360042.

[97] Kai Geissdoerfer and Marco Zimmerling. 2021. Bootstrapping Battery-free Wireless Networks: Efficient Neighbor Discovery and Synchronization in the Face of Intermittency. In *Proc. NSDI*. USENIX, Virtual Event, 439–455. https://www.usenix.org/system/files/nsdi21-geissdoerfer.pdf.

[98] Kai Geissdoerfer and Marco Zimmerling. 2022. Learning to Communicate Effectively Between Battery-free Devices. In *Proc. NSDI*. USENIX, Renton, WA, USA, 419–435. https://www.usenix.org/system/files/nsdi22-paper-geissdoerfer.pdf.

[99] German Federal Minister for the Environment, Nature Conservation, and Nuclear Safety. 2020. Umweltpolitische Digitalagenda (in German). https://www.bmu.de/fileadmin/Daten_BMU/Pools/Broschueren/broschuere_digitalagenda_bf.pdf. Last accessed: Apr. 28, 2020.

[100] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *Proc. ASPLOS*. ACM, Providence, RI, USA, 199–213. https://doi.org/10.1145/3297858.3304011.

[101] Andres Gomez. 2020. Demo Abstract: On-Demand Communication with the Batteryless MiroCard. In *Proc. SenSys*. ACM, Virtual Event, 629–630. https://doi.org/10.1145/3384419.3430440.

[102] Google, LLC. 2019. Google Pixel 3a. https://support.google.com/pixelphone/answer/7158570. Last accessed: May 19, 2022.

[103] Google, LLC. 2021. Android 11 Mobile Operating System. https://www.android.com/android-11. Last accessed: Aug. 8, 2021.

[104] Google, LLC. 2021. OpenThread Source Code Repository. https://github.com/openthread/openthread. Last accessed: Aug. 10, 2021.

[105] Google, LLC. 2022. Protocol buffers for Serializing Structured Data Product Website. https://developers.google.com/protocol-buffers. Last accessed: Oct. 15, 2022.

[106] Google, LLC. 2023. Google Nest Learning Thermostat. https://store.google.com/nl/product/nest_learning_thermostat_3rd_gen. Last accessed: January 24, 2023.

[107] Lewis Gordon. 2019. The Environmental Impact of a PlayStation 4. https://www.theverge.com/2019/12/5/20985330/ps4-sony-playstation-environmental-impact-carbon-footprint-manufacturing-25-anniversary. Last accessed: Apr. 28, 2020.

[108] Green Electronics Council. 2020. Electronic Product Environmental Assessment Tool. https://epeat.net. Last accessed: Apr. 30, 2020.

[109] Manoj Gulati, Farshid Salemi Parizi, Eric Whitmire, Sidhant Gupta, Shobha Sundar Ram, Amarjeet Singh, and Shwetak N. Patel. 2018. CapHarvester: A Stick-on Capacitive Energy Harvester Using Stray Electric Field from AC Power Lines. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3 (September 2018), 110:1–110:20. https://doi.org/10.1145/3264920.

[110] Mike Hayes, Giorgos Fagas, Julie Donnelly, Raphaël Salot, Guillaume Savelli, Peter Spies, Gerd vom Boegel, Mario Konijnenburg, David Stenzel, Aldo Romani, Claudio Gerbaldi, Francesco Cottone, and Alex Weddell. 2021. Research Infrastructure to Power the Internet of Things. https://www.tyndall.ie/contentfiles/EnABLES_Research_Infrastructure_Position_Paper.pdf. Last accessed: Aug. 3, 2021.

[111] Mehrdad Hessar, Ali Najafi, and Shyamnath Gollakota. 2019. NetScatter: Enabling Large-Scale Backscatter Networks. In *Proc. NSDI*. USENIX, Boston, MA, USA, 271–283. https://www.usenix.org/system/files/nsdi19-hessar.pdf.

[112] Josiah Hester, Timothy Scott, and Jacob Sorber. 2014. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-Harvesting Sensors. In *Proc. SenSys* (November 3–5). ACM, Memphis, TN, USA, 1–15. https://doi.org/10.1145/2668332.2668336.

[113] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors. In *Proc. SenSys* (November 1–4). ACM, Seoul, South Korea, 5–16. https://doi.org/10.1145/2809695.2809707.

[114] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proc. SenSys* (November 6–8). ACM, Delft, The Netherlands, 19:1–19:13. https://doi.org/10.1145/3131672.3131674.

[115] Josiah Hester and Jacob Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proc. SenSys*. ACM, Delft, The Netherlands, 21:1–21:6. https://doi.org/10.1145/3131672.3131699.

[116] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proc. SenSys* (November 6–8). ACM, Delft, The Netherlands, 17:1–17:13. https://doi.org/10.1145/3131672.3131673.

[117] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P. Burleson, and Jacob Sorber. 2016. Persistent Clocks for Batteryless Sensing Devices. *ACM Trans. Embed. Comput. Syst.* 15, 4 (August 2016), 77:1–77:28. https://doi.org/10.1145/2903140.

[118] Kasun Hewage, Ambuj Varshney, Abdalah Hilmia, and Thiemo Voigt. 2016. modbulb: A Modular Light Bulb for Visible Light Communication. In *Proc. VCLS* (October 3–7). ACM, New York City, NY, USA, 13–18. https://doi.org/10.1145/2981548.2981559.

[119] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *Proc. ISCA* (June 24–28). IEEE, Toronto, ON, Canada, 228–240. https://doi.org/10.1145/3079856.3080238.

[120] Meng-Ju Hsieh, Jr-Ling Guo, Chin-Yuan Lu, Han-Wei Hsieh, Rong-Hao Liang, and Bing-Yu Chen. 2019. RFTouchPads: Batteryless and Wireless Modular Touch Sensor Pads Based on RFID. In *Proc. UIST*. ACM, New Orleans, LA, US, 999–1011. https://doi.org/10.1145/3332165.3347910.

[121] Meng-Ju Hsieh, Rong-Hao Liang, Da-Yuan Huang, Jheng-You Ke, and Bing-Yu Chen. 2018. RFIBricks: Interactive Building Blocks Based on RFID. In *Proc. CHI* (April 21–26). ACM, Montréal, QC, Canada, 1–10. https://doi.org/10.1145/3173574.3173763.

[122] Pan Hu, Pengyu Zhang, Mohammad Rostami, and Deepak Ganesan. 2016. Braidio: An Integrated Active-Passive Radio for Mobile Devices with Asymmetric Energy Budgets. In *Proc. SIGCOMM*. ACM, Florianopolis, Brazil, 384–397. https://doi.org/10.1145/2934872.2934902.

[123] Nick Huber. 2020. Internet of Things: Smart Cities Pick up the Pace. https://www.ft.com/content/140ae3f0-1b6f-11ea-81f0-0c253907d3e0. Last accessed: May 7, 2020.

[124] Chanyou Hwang, Saumay Pushp, Changyoung Koh, Jungpil Yoon, Yunxin Liu, Seungpyo Choi, and Junehwa Song. 2017. RAVEN: Perception-aware Optimization of Power Consumption for Mobile Games. In *Proc. MobiCom*. ACM, Snowbird, UT, USA, 422–434. https://doi.org/10.1145/3117811.3117841.

[125] Kaori Ikematsu, Masaaki Fukumoto, and Itiro Siio. 2019. Ohmic-Sticker: Force-to-Motion Type Input Device for Capacitive Touch Surface. In *Proc. CHI* (May 4–9). ACM, Glasgow, Scotland, UK, LBW0223:1–LBW0223:6. https://doi.org/10.1145/3332165.3347903.

[126] Ivar in 't Veen, Qingzhi Liu, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. 2016. BLISP: Enhancing Backscatter Radio with Active Radio for Computational RFIDs. In *Proc. RFID*. IEEE, Orlando, FL, USA, 1–4. https://doi.org/10.1109/RFID.2016.7488010.

[127] Microchip Technology Inc. 2017. MIC841 Comparator with 1.25% Reference and Adjustable Hysteresis. http://ww1.microchip.com/downloads/en/DeviceDoc/20005758A.pdf. Last accessed: Jan. 19, 2020.

[128] TI Inc. 2012. SN74AUP2G79 Low-Power Dual Positive Edge-Triggered D-Type Flip-Flop. http://www.ti.com/lit/ds/symlink/sn74aup2g79.pdf. Last accessed: Jan. 19, 2020.

[129] TI Inc. 2015. TLV3691 Nanopower Comparator. http://www.ti.com/lit/ds/symlink/tlv3691.pdf. Last accessed: Jan. 19, 2020.

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

[131] TI Inc. 2016. MSP430 GCC Compiler. http://www.ti.com/tool/MSP430-GCC-OPENSOURCE. Last accessed: Jan. 19, 2020.

[132] TI Inc. 2018. MSP430FR5994 16 MHz Ultra-Low-Power MCU. www.ti.com/lit/gpn/msp430fr5994. Last accessed: Jan. 19, 2020.

[133] Texas Instruments Inc. 2005. TS5A23159 1-Ohm 2-Channel SPDT Analog Switch 5-V / 3.3-V 2-Channel 2:1 Multiplexer / Demultiplexer. ttps://www.ti.com/lit/ds/symlink/ts5a23159.pdf. Last accessed: May 9, 2022.

[134] Texas Instruments. 2015. OPAx192, Precision, Low Input Bias Current Op Amp. https://www.ti.com/lit/ds/symlink/opa192.pdf. Last accessed: May 9, 2022.

[135] Texas Instruments. 2016. TPS7A87 Dual, 500-mA, Low-Noise, LDO Voltage Regulator. https://www.ti.com/lit/ds/symlink/tps7a87.pdf. Last accessed: May 9, 2022.

[136] Texas Instruments. 2021. INA186, Current-Sense Amplifier. https://www.ti.com/lit/ds/symlink/ina186.pdf. Last accessed: May 9, 2022.

[137] Jittrapol Intarasirisawat, Cheesiang Ang, Christos Efstratiou, Luke William Feidhlim Dickens, and Rupert Page. 2019. Exploring the Touch and Motion Features in Game-Based Cognitive Assessments. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 3 (September 2019), 87:1–87:25. https://doi.org/10.1145/3351245.

[138] Maxim Integrated. 2019. Low-Voltage SPI/3-Wire RTCs with Trickle Charger. https://datasheets.maximintegrated.com/en/ds/DS1390-DS1394.pdf. Last accessed: Jan. 19, 2020.

[139] Christos Ioannou, Patrick Archard, Eamonn O'Neill, and Christof Lutteroth. 2019. Virtual Performance Augmentation in an Immersive Jump & Run Exergame. In *Proc. CHI* (May 4–9). ACM, Glasgow, Scotland, UK, 158:1–158:15. https://doi.org/10.1145/3290605.3300388.

[140] Vikram Iyer, Elyas Bayati, Rajalakshmi Nandakumar, Arka Majumdar, and Shyam Gollakota. 2017. Charging a Smartphone Across a Room Using Lasers. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4 (December 2017), 143:1–143:21. https://doi.org/10.1145/3161163.

[141] Vikram Iyer, Maruchi Kim, Shirley Xue, Anran Wang, and Shyamnath Gollakota. 2020. Airdropping Sensor Networks from Drones and Insects. In *Proc. MobiCom*. ACM, London, United Kingdom, 813–826. https://doi.org/10.1145/3372224.3419981.

[142] Ravi Jain and John Wullert II. 2002. Challenges: Environmental Design for Pervasive Computing Systems. In *Proc. MobiCom* (September 23–28). ACM, Atlanta, GA, USA, 263–270. https://doi.org/10.1145/570645.570678.

[143] Sushant Jain, Michael Demmer, Rabin Patra, and Kevin Fall. 2005. Using Redundancy to Cope with Failures in a Delay Tolerant Network. In *Proc. SIGCOMM*. ACM, Philadelphia, PA, USA, 109–120. https://doi.org/10.1145/1080091.1080106.

[144] Japan Display Inc. 2016. LPM013M126A 1.28" MIP Reflective Color LTPS TFT LCD. https://www.j-display.com/product/pdf/Datasheet/4LPM013M126A_specification_Ver02.pdf. Last accessed: Apr. 25, 2020.

[145] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. 2015. Quickrecall: A HW/SW Approach for Computing Across Power Cycles in Transiently Powered Computers. *J. Emerg. Technol. Comput. Syst.* 12, 1 (July 2015), 8:1–8:19. https://doi.org/10.1145/2700249.

[146] Asangi Jayatilaka, Quoc Hung Dang, Shengjian Jammy Chen, Renuka Visvanathan, Christophe Fumeaux, and Damith C. Ranasinghe. 2019. Designing Batteryless Wearables for Hospitalized Older People. In *Proc. ISWC*. ACM, London, UK, 91–95. https://doi.org/10.1145/3341163.3347740.

[147] Kang Eun Jeon, James She, Jason Xue, Sang-Ha Kim, and Soochang Park. 2019. LuXbeacon—A Batteryless Beacon for Green IoT: Design, Modeling, and Field Tests. *IEEE Internet Things J.* 6, 3 (June 2019), 5001–5012. https://doi.org/10.1109/JIOT.2019.2894798.

[148] Haojian Jin, Jingxian Wang, Zhijian Yang, Swarun Kumar, and Jason Hong. 2018. WiSh: Towards a Wireless Shape-aware World using Passive RFIDs. In *Proc. MobiSys*. ACM, Munich, Germany, 428–441. https://doi.org/10.1145/3210240.3210328.

[149] Kumara Kahatapitiya, Chamod Weerasinghe, Jinal Jayawardhana, Hiranya Kuruppu, Kanchana Thilakarathna, and Dileeka Dias. 2018. Low-power Step Counting Paired with Electromagnetic Energy Harvesting for Wearables. In *Proc. ISWC* (October 8–12). ACM, Singapore, 218–219. https://doi.org/10.1145/3267242.3267291.

[150] Mustafa Emre Karagozler, Ivan Poupyrev, Gary K. Fedder, and Yuri Suzuki. 2013. Paper Generators: Harvesting Energy from Touching, Rubbing and Sliding. In *Proc. UIST*. ACM, St. Andrews, UK, 23–30. https://doi.org/10.1145/2501988.2502054.

[151] Mohamad Katanbaf, Anthony Weinand, and Vamsi Talla. 2021. Simplifying Backscatter Deployment: Full-Duplex LoRa Backscatter. In *Proc. NSDI*. USENIX, Virtual Event, 955–972. https://www.usenix.org/system/files/nsdi21spring-katanbaf.pdf.

[152] Keiko Katsuragawa, Ju Wang, Ziyang Shan, Ningshan Ouyang, Omid Abari, and Daniel Vogel. 2019. Tip-Tap: Battery-free Discrete 2D Fingertip Input. In *Proc. UIST*. ACM, New Orleans, LA, US, 1045–1057. https://doi.org/10.1145/3332165.3347907.

[153] Giannis Kazdaridis, Nikos Sidiropoulos, Ioannis Zografopoulos, Polychronis Symeonidis, and Thanasis Korakis. 2020. Nano-things: Pushing Sleep Current Consumption to the Limits in IoT Platforms. In *Proc. IoT*. ACM, Malmö, Sweden, 1–8. https://doi.org/10.1145/3410992.3410998.

[154] Keithley Instruments, LLC. 2021. 2450 SourceMeter Source Measurement Unit Instrument. https://download.tek.com/datasheet/1KW-60904-2_2450_Datasheet_072021.pdf. Last accessed: Sep. 11, 2021.

[155] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R. Smith, and David Wetherall. 2014. Wi-Fi Backscatter: Internet Connectivity for RF-Powered Devices. In *Proc. SIGCOMM*. ACM, Chicago, IL, USA, 607–618. https://doi.org/10.1145/2740070.2626319.

[156] Ben Kenwright. 2012. Fast Efficient Fixed-Size Memory Pool: No Loops and No Overhead. In *Proc. Computation Tools*. IARIA, Nice, France, 1–6.

[157] Azam Khan. 2011. Swimming Upstream in Sustainable Design. *Interactions* 18, 5 (September 2011), 12—-14. https://doi.org/10.1145/2008176.2008181.

[158] Mostafa Khoshnevisan and J. Nicholas Laneman. 2017. Intermittent Communication. *IEEE Trans. Inf. Theory* 63, 7 (July 2017), 4089–4102. https://doi.org/10.1109/TIT.2017.2692239.

[159] Daeyong Kim, Junick Ahn, Jun Shin, and Hojung Cha. 2021. Ray Tracing-based Light Energy Prediction for Indoor Batteryless Sensors. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 1 (March 2021), 17:1–17:27. https://doi.org/10.1145/3448086.

[160] Hyung-Sin Kim, Michael P. Andersen, Kaifei Chen, Sam Kumar, William J. Zhao, Kevin Ma, and David E. Culler. 2018. System Architecture Directions for Post-SoC/32-bit Networked Sensors. In *Proc. SenSys* (November 4–7,). ACM, Shenzhen, China, 264–277. https://doi.org/10.1145/3274783.3274839.

[161] Yoojung Kim, Arpita Bhattacharya, Julie A. Kientz, and Jin Ha Lee. 2020. "It Should Be a Game for Fun, Not Exercise": Tensions in Designing Health-Related Features for Pokémon GO. In *Proc. CHI* (April 25–30). ACM, Honolulu, HI, USA, 1–13. https://doi.org/10.1145/3313831.3376830.

[162] Bran Knowles, Lynne Blair, Mike Hazas, and Stuart Walker. 2013. Exploring Sustainability Research in Computing: Where we Are and Where we go Next. In *Proc. UbiComp* (September 8–12). ACM, Zurich, Switzerland, 305–314. https://doi.org/10.1145/2493432.2493474.

[163] Koninklijke Philips N.V. 2021. Hue Smart Light Bulb White Ambiance E27. https://www.philips-hue.com/en-gb/p/hue-white-ambiance-1-pack-e27/8718699673147. Last accessed: Aug. 19, 2021.

[164] Vito Kortbeek, Abu Bakar, Stefany Cruz Kasım Sinan Yıldırım, Przemysław Pawełczak, and Josiah Hester. 2020. BFree: Enabling Battery-free Sensor Prototyping with Python. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 4 (December 2020), 135:1–111:39. https://doi.org/10.1145/3432191.

[165] Vito Kortbeek, Souradip Ghosh, Josiah Hester, Simone Campanoni, and Przemysław Pawełczak. 2022. WARio: Efficient Code Generation for Intermittent Computing. In *Proc. PLDI* (June 13–17). ACM, San Diego, CA, USA, 777–791. https://doi.org/10.1145/3519939.3523454.

[166] Vito Kortbeek, Kasım Sinan Yıldırım, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. 2020. Time-sensitive Intermittent Computing Meets Legacy Software. In *Proc. ASPLOS*. ACM, Lausanne, Switzerland, 85–99. https://doi.org/10.1145/3373376.3378476.

[167] Andre Lam, Ivan Talev, and Jennifer Kim. 2020. Design life-Cycle: University of California, Davis, CA, USA, Department of Design's Undergraduate Students Project Designed by Christina Cogdell. http://www.designlife-cycle.com/nintendo-switch. Last accessed: Apr. 30, 2020.

[168] Trong Nhan Le, Alain Pegatoquet, Olivier Berder, and Olivier Sentieys. 2015. Energy-Efficient Power Manager and MAC Protocol for Multi-Hop Wireless Sensor Networks Powered by Periodic Energy Harvesting Sources. *IEEE Sensors J.* 15, 12 (December 2015), 7208–7220. https://doi.org/10.1109/JSEN.2015.2472566.

[169] Seulki Lee, Bashima Islam, Yubo Luo, and Shahriar Nirjon. 2019. Intermittent Learning: On-Device Machine Learning on Intermittently Powered System. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4 (December 2019), 141:1–141:30. https://doi.org/10.1145/3369837.

[170] Christoph Lenzen, Philipp Sommer, and Roger Wattenhofer. 2015. PulseSync: An Efficient and Scalable Clock Synchronization Protocol. *IEEE/ACM Trans. Netw.* 23, 3 (June 2015), 717–727. https://doi.org/10.1109/TNET.2014.2309805.

[171] Dong Li, Feng Ding, Qian Zhang, Run Zhao, Jinshi Zhang, and Dong Wang. 2017. TagController: A Universal Wireless and Battery-free Remote Controller using Passive RFID Tags. In *Proc. MobiQuitous*. ACM, Melbourne, VIC, Australia, 166–175. https://doi.org/10.1145/3144457.3144498.

[172] Hanchuan Li, Eric Brockmeyer, Elizabeth J. Carter, Josh Fromm, Scott E. Hudson, Shwetak N. Patel, and Alanson Sample. 2016. PaperID: A Technique for Drawing Functional Battery-Free Wireless Interfaces on Paper. In *Proc. CHI* (May 7–12). ACM, San Jose, CA, USA, 5885–5896. https://doi.org/10.1145/2858036.2858249.

[173] Tianxing Li and Xia Zhou. 2018. Battery-Free Eye Tracker on Glasses. In *Proc. MobiCom* (October 29 — November 2). ACM, New Delhi, India, 67–82. https://doi.org/10.1145/3241539.3241578.

[174] Yong Li, Pan Hui, Depeng Jin, and Sheng Chen. 2015. Delay-Tolerant Network Protocol Testing and Evaluation. https://doi.org/10.1109/MCOM.2015.7010543. *IEEE Communications Magazine* 53, 1 (January 2015), 258–266.

[175] Yichen Li, Tianxing Li, Xing-Dong Yang Ruchir A. Patel, and Xia Zhou. 2018. Self-Powered Gesture Recognition with Ambient Light. In *Proc. UIST*. ACM, Berlin, Germany, 595–608. https://doi.org/10.1145/3242587.3242635.

[176] Yang Li, Rui Tan, and David K. Y. Yau. 2017. Natural Timestamping using Powerline Electromagnetic Radiation. In *Proc. IPSN* (April 18–20). ACM/IEEE, Pittsburgh, PA, USA, 55–66. https://doi.org/10.1145/3055031.3055075.

[177] Zhuying Li, Yan Wang, Wei Wang, Weikang Chen, Ti Hoang, Stefan Greuter, and Florian 'Floyd' Mueller. 2019. HeatCraft: Designing Playful Experiences with Ingestible Sensors via Localized Thermal Stimuli. In *Proc. CHI* (May 4–9). ACM, Glasgow, Scotland, UK, 576:1–576:12. https://doi.org/10.1145/3290605.3300806.

[178] Rong-Hao Liang, Meng-Ju Hsieh, Jheng-You Ke, Jr-Ling Guo, and Bing-Yu Chen. 2018. RFIMatch: Distributed Batteryless Near-Field Identification Using RFID-Tagged Magnet-Biased Reed Switches. In *Proc. UIST*. ACM, Berlin, Germany, 473–483. https://doi.org/10.1145/3242587.3242620.

[179] Lightricity Limited. 2021. EXL2-1V50 Solar Panel Module. https://lightricity.co.uk/excelllight-exl2-1v50-1. Last accessed: Sep. 9, 2021.

[180] Roman Lim, Balz Maag, and Lothar Thiele. 2016. Time-of-Flight Aware Time Synchronization for Wireless Embedded Systems. In *Proc. EWSN* (Graz, Austria, February 15–17). ACM, Graz, Austria, 149–158.

[181] Qingzhi Liu, Wieger IJntema, Anass Drif, Przemysław Pawełczak, Marco Zuniga, and Kasım Sinan Yıldırım. 2021. Perpetual Bluetooth Communications for the IoT. *IEEE Sens. J.* 21, 1 (January 2021), 829–837. https://doi.org/10.1109/JSEN.2020.3012814.

[182] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall, and Joshua R. Smith. 2013. Ambient Backscatter: Wireless Communication out of Thin Air. In *Proc. SIGCOMM* (August 12–16). ACM, Hong Kong, China, 39–50. https://doi.org/10.1145/2486001.2486015.

[183] Xiao Lu, Ping Wang, Dusit Niyato, Dong In Kim, and Zhu Han. 2015. Wireless Networks with RF Energy Harvesting: A Contemporary Survey. *IEEE Commun. Surveys Tuts.* 17, 2 (2015), 757–789. https://doi.org/10.1109/COMST.2014.2368999.

[184] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *Proc. SNAPL*. Schloss Dagstuhl, Alisomar, CA, USA, 8:1–8:14. https://drops.dagstuhl.de/opus/volltexte/2017/7131/pdf/LIPIcs-SNAPL-2017-8.pdf.

[185] Brandon Lucia and Benjamin Ransford. 2015. A simpler, Safer Programming and Execution Model for Intermittent Systems. In *Proc. PLDI* (August 13–17). ACM, Portland, OR, USA, 575–585. https://doi.org/10.1145/2737924.2737978.

[186] Dong Ma, Guohao Lan, Mahbub Hassan, Wen Hu, Mushfika Baishakhi Upama, Ashraf Uddin, and Moustafa Youssef. 2019. SolarGest: Ubiquitous and Battery-free Gesture Recognition using Solar Cells. In *Proc. MobiCom* (April 21–25). ACM, Los Cabos, Mexico, 12:1–12:15. https://doi.org/10.1145/3300061.3300129.

[187] Junchao Ma, Wei Lou, Yanwei Wu, Xiang-Yang Li, and Guihai Chen. 2009. Energy Efficient TDMA Sleep Scheduling in Wireless Sensor Networks. In *Proc. INFOCOM* (April 19–25,). IEEE, Rio de Janeiro, Brazil, 630–638. https://doi.org/10.1109/INFCOM.2009.5061970.

[188] Yunfei Ma, Nicholas Selby, and Fadel Adib. 2017. Drone Relays for Battery-Free Networks. In *Proc. SIGCOMM* (August 21–25). ACM, Los Angeles, CA, USA, 335—-347. https://doi.org/10.1145/3098822.3098847.

[189] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent Execution without Checkpoints. In *Proc. OOPSLA* (October 22–27). ACM, Vancouver, BC, Canada, 96:1–96:30. https://doi.org/10.1145/3133920.

[190] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2018. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *Proc. OSDI* (October 8–10). USENIX, Carlsbad, CA, USA, 129–144. https://www.usenix.org/system/files/osdi18-maeng.pdf.

[191] Kiwan Maeng and Brandon Lucia. 2019. Supporting Peripherals in Intermittent Systems with Just-in-time Checkpoints. In *Proc. PLDI*. ACM, Phoenix, AZ, USA, 1101–1116. https://doi.org/10.1145/3314221.3314613.

[192] Andrea Maioli, Luca Mottola, Muhammad Hamad Alizai, and Junaid Haroon Siddiqui. 2021. Discovering the Hidden Anomalies of Intermittent Computing. https://www.ewsn.org/file-repository/ewsn2021/Article1.pdf. In *Proc. EWSN* (February 17–19). ACM, Delft, The Netherlands, 1–12.

[193] Amjad Yousef Majid, Carlo Delle Donne, Kiwan Maeng, Alexei Colin, Kasım Sinan Yıldırım, Brandon Lucia, and Przemysław Pawełczak. 2020. Dynamic Task-based Intermittent Execution for Energy-harvesting Devices. *ACM Trans. Sens. Netw.* 16, 1 (February 2020), 5:1–5:24. https://doi.org/10.1145/3360285.

[194] Amjad Yousef Majid, Michel Jansen, Guillermo Ortas Ortas, Kasım Sinan Yıldırım, and Przemysław Pawełczak. 2019. Multi-hop Backscatter Tag-to-Tag Network. In *Proc. INFOCOM*. IEEE, Paris, France, 721–729. https://doi.org/10.1109/INFOCOM.2019.8737551.

[195] Jennifer C. Mankoff, Eli Blevis, Alan Borning, Batya Friedman, Susan R. Fussell, Jay Hasbrouck, Allison Woodruff, and Phoebe Sengers. 2007. Environmental Sustainability and Interaction. In *Proc. CHI* (April 28 – May 3). ACM, San Jose, CA, USA, 2121–2124. https://doi.org/10.1145/1240866.1240963.

[196] Gaia Maselli, Mauro Piva, Giorgia Ramponi, and Deepak Ganesan. 2016. Demo: JoyTag: a battery-less videogame controller exploiting RFID backscattering. In *Proc. MobiCom*. ACM, New York City, NY, USA, 515–516. https://doi.org/10.1145/2973750.2985628.

[197] Yogesh Kumar Meena, Krishna Seunarine, Deepak Ranjan Sahoo, Simon Robinson, Jennifer Pearson, Chi Zhang, Matt Carnie, Adam Pockett, Andrew Prescott, Suzanne K. Thomas, Harrison Ka Hin Lee, and Matt Jones. 2020. PV-Tiles: Towards Closely-Coupled Photovoltaic and Digital Materials for Useful, Beautiful and Sustainable Interactive Surfaces. In *Proc. CHI* (April 25–30). ACM, Honolulu, HI, USA, 1–12. https://doi.org/10.1145/3313831.3376368.

[198] Hashan Roshantha Mendis and Pi-Cheng Hsiu. 2019. Accumulative Display Updating for Intermittent Systems. *ACM Transactions on Embedded Computing Systems* 18, 5s (October 2019), 72:1–72:22. https://doi.org/10.1145/3358190.

[199] Geoff V. Merrett and Bashir M. Al-Hashimi. 2017. Energy-Driven Computing: Rethinking the Design of Energy Harvesting Systems. In *Proc. DATE*. IEEE, Lausanne, Switzerland, 960–965. https://doi.org/10.23919/DATE.2017.7927130.

[200] Ambiq Micro. 2019. AM18xx Family Ultra-Low Power RTCs Data Sheet. https://www.ambiqmicro.com/static/rtc/files/AM18X5_Data_Sheet_DS0003V1p3.pdf. Last accessed: Jan. 19, 2020.

[201] Microchip Technology Inc. 2016. SAM4L8 Xplained Pro Evaluation Kit. https://www.microchip.com/en-us/development-tool/ATSAM4L8-XPRO. Last accessed: Jun. 16, 2022.

[202] Evan Mills, Norman Bourassa, Leo Rainer, Jimmy Mai, Arman Shehabi, and Nathaniel Mills. 2019. Toward Greener Gaming: Estimating National Energy Use and Energy Efficiency Potential. *The Computer Games Journal* 8 (October 2019), 157–178. https://doi.org/10.1007/s40869-019-00084-2.

[203] Neeru Mittal, Alazne Ojanguren, Markus Niederberger, and Erlantz Lizundia. 2021. Degradation Behavior, Biocompatibility, Electrochemical Performance, and Circularity Potential of Transient Batteries. *Advanced Science* 8, 2004814 (May 2021), 1–26. https://doi.org/10.1002/ADVS.202004814.

[204] Noor Mohammed, Rui Wang, Robert W. Jackson, Yeonsik Noh, Jeremy Gummeson, and Sunghoon Ivan Lee. 2021. ShaZam: Charge-Free Wearable Devices via Intra-Body Power Transfer from Everyday Objects. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 2 (June 2021), 75:1–75:25. https://doi.org/10.1145/3463505.

[205] Florian 'Floyd' Mueller, Richard Byrne, Josh Andres, and Rakesh Patibanda. 2018. Experiencing the Body as Play. In *Proc. CHI* (April 21–26). ACM, Montréal, QC, Canada, 210:1–210:13. https://doi.org/10.1145/3173574.3173784.

[206] Saman Naderiparizi, Mehrdad Hessar, Vamsi Talla, Shyamnath Gollakota, and Joshua R. Smith. 2018. Towards Battery-Free HD Video Streaming. In *Proc. NSDI* (April 9–11). USENIX, Renton, WA, USA, 233–247. https://www.usenix.org/system/files/conference/nsdi18/nsdi18-naderiparizi.pdf.

[207] Saman Naderiparizi, Aaron N. Parks, Zerina Kapetanovic, Benjamin Ransford, and Joshua R. Smith. 2015. WISPCam: A Battery-Free RFID Camera. In *Proc. IEEE RFID*. IEEE, San Diego, CA, USA, 166–173. https://doi.org/10.1109/RFID.2015.7113088.

[208] Saman Naderiparizi, Yi Zhao, James Youngquist, Alanson P. Sample, and Joshua R. Smith. 2015. Self-Localizing Battery-Free Cameras. In *Proc. UbiComp*. ACM, Osaka, Japan, 445–449. https://doi.org/10.1145/2750858.2805846.

[209] Yuji Nakamura. 2019. Peak Video Game? Top Analyst Sees Industry Slumping in 2019. https://www.bloomberg.com/news/articles/2019-01-23/peak-video-game-top-analyst-sees-industry-slumping-in-2019. Last accessed: Jan. 7, 2020.

[210] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: a Batteryless Long-Range Remote Visual Sensing System. In *Proc. ENSsys*. ACM, New York, NY, USA, 8–14. https://doi.org/10.1145/3362053.3363491.

[211] Nexperia. 2021. 74LVC2T45; 74LVCH2T45 Dual Supply Translating Transceiver; 3-state. https://assets.nexperia.com/documents/data-sheet/74LVC_LVCH2T45.pdf. Last accessed: May 9, 2022.

[212] Phuc Nguyen, Ufuk Muncuk, Ashwin Ashok, Kaushik Roy Chowdhury, Marco Gruteser, and Tam Vu. 2016. Battery-Free Identification Token for Touch Sensing Devices. In *Proc. SenSys*. ACM, Stanford, CA, USA, 109–122. https://doi.org/10.1145/2994551.2994566.

[213] Thien D. Nguyen, Jamil Y. Khan, and Duy T. Ngo. 2016. An adaptive MAC Protocol for RF Energy Harvesting Wireless Sensor Networks. In *Proc. GLOBECOM* (December 4–8). IEEE, Washington, DC, USA, 1–6. https://doi.org/10.1109/GLOCOM.2016.7841577.

[214] Nintendo Co., Ltd. 2019. Dedicated Video Game Sales Units. https://www.nintendo.co.jp/ir/en/finance/hard_soft/index.html. Last accessed: Apr. 30, 2020.

[215] Nintendo Co., Ltd. 2020. Nintendo Game Boy. https://en.wikipedia.org/wiki/Game_Boy. Last accessed: Jul. 23, 2020.

[216] Nordic Semiconductor ASA. 2019. S140 BLE protocol stack (SoftDevice) for the nRF52811, nRF52820, nRF52833 and nRF52840 SoCs. https://www.nordicsemi.com/Products/Development-software/s140. Last accessed: Sep. 9, 2021.

[217] Nordic Semiconductor ASA. 2020. nRF52 DK BLE and Bluetooth Mesh Single Board Development Kit for the nRF52810 and nRF52832 SoCs. https://www.nordicsemi.com/Products/Development-hardware/nRF52-DK. Last accessed: Sep. 9, 2021.

[218] Nordic Semiconductor ASA. 2020. nRF52840 DK BLE, Bluetooth Mesh, Near-Field Communication (NFC), Thread and Zigbee Single Board Development Kit for the nRF52840 SoC. https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK. Last accessed: Sep. 9, 2021.

[219] Nordic Semiconductor ASA. 2021. nRF51822 BLE and 2.4 GHz SoC. https://infocenter.nordicsemi.com/topic/struct_nrf51/struct/nrf51822.html. Last accessed: Sep. 9, 2021.

[220] Nordic Semiconductor ASA. 2021. nRF52840 Multiprotocol Bluetooth 5.2 SoC supporting BLE, Bluetooth mesh, NFC, Thread and Zigbee. https://www.nordicsemi.com/Products/nRF52840. Last accessed: Dec. 5, 2021.

[221] Nordic Semiconductor ASA. 2021. Online Power Profiler for Bluetooth LE. https://devzone.nordicsemi.com/power/w/opp/2/online-power-profiler-for-bluetooth-le. Last accessed: Dec. 5, 2021.

[222] NOWI B.V. 2021. NH2D0245 Energy Harvesting Power Management Integrated Circuit. https://www.nowi-energy.com/products-nh2. Last accessed: Sep. 9, 2021.

[223] Sam Nussey and Christopher Cushing. 2020. Nintendo Seen Extending Profit Streak as Housebound Consumers Switch On. https://www.reuters.com/article/us-nintendo-results-preview/nintendo-seen-extending-profit-streak-as-housebound-consumers-switch-on-idUSKBN22C0B4. Last accessed: May 15, 2020.

[224] NXP Semiconductors N.V. 2013. Freedom Development Platform for the Kinetis KL05 and KL04 MCUs. https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-the-kinetis-kl05-and-kl04-mcus:FRDM-KL05Z. Last accessed: Sep. 11, 2021.

[225] Netta Ofer, Idan David, Hadas Erel, and Oren Zuckerman. 2019. Coding for Outdoor Play: A Coding Platform for Children to Invent and Enhance Outdoor Play Experiences. In *Proc. CHI* (May 4–9). ACM, Glasgow, Scotland, UK, 1–12. https://doi.org/10.1145/3290605.3300394.

[226] Kazuya Oharada, Buntarou Shizuki, and Shin Takahashi. 2017. AccelTag: A Passive Smart ID Tag With an Acceleration Sensor for Interactive Applications. In *Proc. UIST Adjunct*. ACM, Québec City, Canada, 63–64. https://doi.org/10.1145/3131785.3131808.

[227] Open Source Community Contributors. 2021. Unicorn: a Lightweight, Multi-platform, Multi-architecture Central Processing Unit (CPU) Emulator Framework based on QEMU. https://github.com/unicorn-engine/unicorn. Last accessed: May 19, 2022.

[228] Open Source Community Developers. 2022. Black Magic Debug Repository. https://github.com/blackmagic-debug. Last accessed: May 9, 2022.

[229] Open Source Community Developers. 2022. GDB: The GNU Project Debugger Repository. https://sourceware.org/git/binutils-gdb.git. Last accessed: May 10, 2022.

[230] Charalampos Orfanidis, Konstantinos Dimitrakopoulos, Xenofon Fafoutis, and Martin Jacobsson. 2019. Towards Battery-Free LPWAN Wearables. In *Proc. ENSsys*. ACM, New York, NY, USA, 52–53. https://doi.org/10.1145/3362053.3363488.

[231] Packetcraft, Inc. 2021. Packetcraft Protocol Software Source Code Repository. https://github.com/packetcraft-inc/stacks. Last accessed: Aug. 5, 2021.

[232] M. R. Palacín and A. de Guibert. 2016. Why do Batteries Fail? *Science* 351, 6273 (2016), 1–7. https://doi.org/10.1126/science.1253292.

[233] Panasonic Electric Works Europe AG. 2019. AM-1417CA Amorphous Silicon Solar Cell. https://www.panasonic-electric-works.com/cps/rde/xbcr/pew_eu_en/ca_amorton_solar_cells_en.pdf. Last accessed: Apr. 25, 2020.

[234] Panic Inc. 2020. Playdate Console Home Page. https://play.date. Last accessed: May 2, 2020.

[235] Arielle Pardes. 2020. The WIRED Guide to the Internet of Things. WIRED, https://www.wired.com/story/wired-guide-internet-of-things. Last accessed: Jul. 6, 2021.

[236] Aaron N. Parks, Angli Liu, Shyamnath Gollakota, and Joshua R. Smith. 2014. Turbocharging Ambient Backscatter Communication. In *Proc. SIGCOMM*. ACM, Chicago, IL, USA, 619–630. https://doi.org/10.1145/2740070.2626312.

[237] Carlos Perez-Penichet, Fredrick Hermans, Ambuj Varshney, and Thiemo Voigt. 2016. Augmenting IoT Networks with Backscatter-Enabled Passive Sensor Tags. In *Proc. HotWireless*. ACM, New York City, NY, USA, 23–27. https://doi.org/10.1145/2980115.2980132.

[238] Matthai Philipose, Joshua R. Smith, Bing Jiang, Alexander Mamishev, Sumit Roy, and Kishor Sundara-Rajan. 2005. Battery-Free Wireless Identification and Sensing. *IEEE Pervasive Comput.* 4, 1 (Jan.–Mar. 2005), 37–45. https://doi.org/10.1109/MPRV.2005.7.

[239] Rajeev Piyare, Amy L. Murphy, Csaba Kiraly, Pietro Tosato, and Davide Brunell. 2017. Ultra Low Power Wake-Up Radios: A Hardware and Networking Survey. *IEEE Commun. Surv. Tutorials* 19, 4 (Fourth Quarter 2017), 2117–2157. https://doi.org/10.1109/COMST.2017.2728092.

[240] Powercast Co. 2016. TX91501 Powercaster Transmitter. www.powercastco.com/wp-content/uploads/2016/11/User-Manual-TX-915-01-Rev-A-4.pdf. Last accessed: Jan. 19, 2020.

[241] Powercast Corp. 2018. P2110 Powerharvester Evaluation Board. https://www.powercastco.com/products/development-kits/#P2110-EVB. Last accessed: Aug. 10, 2021.

[242] R. Venkatesha Prasad, Shruti Devasenapathy, Vijay S. Rao, and Javad Vazifehdan. 2014. Reincarnation in the Ambiance: Devices and Networks with Energy Harvesting. *IEEE Commun. Surveys Tuts.* 11, 1 (First Quarter 2014), 195–213. https://doi.org/10.1109/SURV.2013.062613.00235.

[243] United Nations Environment Programme. 2020. Playing for the Planet Consortium. https://playing4theplanet.org. Last accessed: Apr. 28, 2020.

[244] Qt Group. 2022. Qt Software Development Framework Product Website. https://www.qt.io/product/framework. Last accessed: Oct. 15, 2022.

[245] Amir Rahmat, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P. Burleson, and Kevin Fu. 2012. TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks. In *Proc. Security* (August 8–10). USENIX, Bellevue, WA, USA, 1–16. https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final71.pdf.

[246] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. https://doi.org/10.1145/1950365.1950386. In *Proc. ASPLOS*. ACM, Newport Beach, CA, USA, 159–170.

[247] Reelight. 2011. SL150 Hub Lights. https://www.reelight.com/products/hub-lights. Last accessed: Jan. 19, 2020.

[248] David Richardson, Arshad Jhumka, and Luca Mottola. 2021. Protocol Transformation for Transiently Powered Wireless Sensor Networks. In *Proc. SAC*. ACM, Virtual Event, 1112–1121. https://doi.org/10.1145/3412841.3441985.

[249] Mohammad Rostami, Jeremy Gummeson, Ali Kiaghadi, and Deepak Ganesan. 2018. Polymorphic Radios: A New Design Paradigm for Ultra-low Power Communication. In *Proc. SIGCOMM*. ACM, Budapest, Hungary, 446–460. https://doi.org/10.1145/3230543.3230571.

[250] Michel Rotteluthner, Thomas C. Schmidt, and Matthias Wählisch. 2021. Sense Your Power: The ECO Approach to Energy Awareness for IoT Devices. *ACM Trans. Embed. Comput. Syst.* 20, 3 (March 2021), 24:1–14:23. https://dl.acm.org/doi/10.1145/3441643.

[251] Emily Ruppel and Brandon Lucia. 2019. Transactional Concurrency Control for Intermittent, Energy-Harvesting Computing Systems. In *Proc. PLDI*. ACM, Phoenix, AZ, USA, 1085–1100. https://doi.org/10.1145/3314221.3314583.

[252] Kimiko Ryokai, Peiqi Su, Eungchan Kim, and Bob Rollins. 2014. EnergyBugs: Energy Harvesting Wearables for Children. In *Proc. CHI* (Apr. 26 – May 1). ACM, Toronto, ON, Canada, 1039–1048. https://doi.org/10.1145/2556288.2557225.

[253] Ali Saffari, Mehrdad Hessar, Saman Naderiparizi, and Joshua R. Smith. 2019. Battery-Free Wireless Video Streaming Camera System. In *Proc. RFID*. IEEE, Phoenix, AZ, USA, 1–8. https://doi.org/10.1109/RFID.2019.8719264.

[254] Ali Saffari, Sin Yong Tan, Mohamad Katanbaf, Homagni Saha, Joshua R. Smith, and Soumik Sarkar. 2021. Battery-Free Camera Occupancy Detection System. In *Proc. EMDL*. ACM, Virtual, WI, USA, 13–18. https://doi.org/10.1145/3469116.3470013.

[255] Saleae, Inc. 2021. Logic Pro 16 USB Logic Analyzer. http://downloads.saleae.com/specs/Logic+Pro+16+Product+Fact+Sheet.pdf. Last accessed: Aug. 19, 2021.

[256] Saleae Inc. 2021. Logic Pro 8 USB Logic Analyzer. http://downloads.saleae.com/specs/Logic+Pro+8+Product+Fact+Sheet.pdf. Last accessed: Jun. 16, 2022.

[257] Alanson P. Sample, Daniel J. Yeager, Pauline S. Powledge, Alexander V. Mamishev, and Joshua R. Smith. 2008. Design of an RFID-based battery-free programmable sensing platform. *IEEE Trans. Instrum. Meas.* 57, 11 (November 2008), 2608–2615. https://doi.org/10.1109/TIM.2008.925019.

[258] Nurani Saoda and Bradford Campbell. 2019. No Batteries Needed: Providing Physical Context with Energy-Harvesting Beacons. In *Proc. ENSSys*. ACM, New York, NY, USA, 15–21. https://doi.org/10.1145/3362053.3363489.

[259] Takuya Sasatani, Chouchang Jack Yang, Matthew J. Chabalko, Yoshihiro Kawahara, and Alanson P. Sample. 2018. Room-Wide Wireless Charging and Load-Modulation Communication via Quasistatic Cavity Resonance. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4 (December 2018), 188:1–188:23. https://doi.org/10.1145/3287066.

[260] Seth Schiesel. 2020. For the Uninitiated and Bored, an Introduction to the World of Gaming. https://www.nytimes.com/2020/04/01/arts/gaming-introduction-basics-quarantine-coronavirus.html. Last accessed: Apr. 28, 2020.

[261] Oliver Schneider, Jotaro Shigeyama, Robert Kovacs, Thijs Roumen, Sebastian Marwecki, Nico Boeckhoff, Daniel Amadeus Gloeckner, Jonas Bounama, and Patrick Baudisch. 2018. DualPanto: A Haptic Device that Enables Blind Users to Continuously Interact with Virtual Worlds. In *Proc. UIST*. ACM, Berlin, Germany, 877–887. https://doi.org/10.1145/3242587.3242604.

[262] Jason Schreier. 2020. Gaming Sales Are Up, but Production Is Down. https://www.nytimes.com/2020/04/21/technology/personaltech/coronavirus-video-game-production.html. Last accessed: Apr. 29, 2020.

[263] SEGGER Microcontroller GmbH. 2021. J-Link Educational Debug Probe. https://www.segger.com/products/debug-probes/j-link/models/j-link-edu. Last accessed: Sep. 9, 2021.

[264] Seiko Instruments Inc. 2021. CPX3225A752D Chip-type Electric Double Layer Capacitor. https://www.sii.co.jp/en/me/datasheets/chip-capacitor/cpx3225a752d. Last accessed: Sep. 9, 2021.

[265] Semiconductor Components Industries LLC. 2017. NSR1030QMUTWG Schottky Full Diode Bridge. https://www.onsemi.com/pub/Collateral/NSR1030QMUD.PDF. Last accessed: Apr. 25, 2020.

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

[267] Semtech. 2012. SX1503 4/8/16 Channel Low Voltage GPIO with NINT and NRESET. https://www.semtech.com/products/smart-sensing/io-expanders/sx1503. Last accessed: Apr. 25, 2020.

[268] Uvis Senkans, Domenico Balsamo, Theodoros D. Verykios, and Geoff V. Merrett. 2017. Applications of Energy-Driven Computing: A Transiently-Powered Wireless Cycle Computer. In *Proc. ENSsys* (Delft, The Netherlands). ACM, New York, NY, USA, 1–7. https://doi.org/10.1145/3142992.3142993.

[269] Sharp Corporation. 2016. LS013B7DH03 1.28" TFT-LCD Module. https://www.sharpsde.com/fileadmin/products/Displays/Specs/LS013B7DH03_25Apr16_Spec_LD-28410A.pdf. Last accessed: Sep. 8, 2021.

[270] Esther Shein. 2021. A Battery-Free Internet of Things. *Commun. ACM* 64, 7 (2021), 16–18. https://doi.org/10.1145/3464937.

[271] Hafiz Husnain Raza Sherazi, Luigi Alfredo Grieco, and Gennaro Boggia. 2018. A Comprehensive Review on Energy Harvesting MAC protocols in WSNs: Challenges and Tradeoffs. *Ad Hoc Networks* 71 (2018), 117–134. https://doi.org/10.1016/j.adhoc.2018.01.004.

[272] Rishi Shukla, Neev Kiran, Rui Wang, Jeremy Gummeson, and Sunghoon Ivan Lee. 2019. SkinnyPower: Enabling Batteryless Wearable Sensors via Intra-Body Power Transfer. In *Proc. SenSys* (November 10–13). ACM, New York City, NY, USA, 55–67. https://doi.org/10.1145/3356250.3360034.

[273] Lukas Sigrist, Rehan Ahmed, Andres Gomez, and Lothar Thiele. 2020. Harvesting-Aware Optimal Communication Scheme for Infrastructure-Less Sensing. *ACM Trans. Internet Things* 1, 4 (October 2020), 22:1–22:26. https://doi.org/10.1145/3395928.

[274] Patrice Simon, Yury Gogotsi, and Bruce Dunn. 2014. Where Do Batteries End and Supercapacitors Begin? *Science* 343, 6176 (March 2014), 1210–1211. https://doi.org/10.1126/science.1249625.

[275] Sivert T. Sliper, Oktay Cetinkaya, Alex S. Weddell, Bashir Al-Hashimi, and Geoff V. Merrett. 2020. Energy-driven Computing. *Phil. Trans. R. Soc. A.* 378, 2164 (February 2020), 1–18. https://doi.org/10.1098/rsta.2019.0158.

[276] Joshua R. Smith, Alanson P. Sample, Pauline S. Powledge, Sumit Roy, and Alexander Mamishev. 2006. A Wirelessly-Powered Platform for Sensing and Computation. In *Proc. UbiComp* (September 17–21). ACM, Orange County, CA, USA, 495–506. https://doi.org/10.1007/11853565_29.

[277] SparkFun Electronics. 2019. RedBoard Artemis ATP. https://www.sparkfun.com/products/15442. Last accessed: Jun 16, 2022.

[278] Richard Stallman, Roland H. Pesch, and Stan Shebs. 2011. *Debugging with GDB: The GNU Source-Level Debugger, V 7.3.1.* Free Software Foundation, Boston, MA, USA.

[279] STMicroelectronics. 2016. Mainstream Mixed signals MCUs Arm Cortex-M4 core with DSP and FPU, 256KB of Flash memory, 72MHz CPU, MPU, 16-bit SDADC. https://www.st.com/en/microcontrollers-microprocessors/stm32f373cc.html. Last accessed: May 9, 2022.

[280] STMicroelectronics. 2018. Mainstream Performance Line, Arm Cortex-M3 MCU with 512KB of Flash memory, 72MHz CPU, Motor control, USB and CAN. https://www.st.com/en/microcontrollers-microprocessors/stm32f103re.html. Last accessed: May 9, 2022.

[281] STMicroelectronics. 2018. X-NUCLEO-LPM01A Nucleo Expansion Board for Power Consumption Measurement. https://www.st.com/en/evaluation-tools/x-nucleo-lpm01a.html. Last accessed: Jan. 19, 2020.

[282] STMicroelectronics. 2019. ST M41T62 Low-power Serial Real-time Clock Data Sheet. https://www.st.com/resource/en/datasheet/m41t62.pdf. Last accessed: Jan. 19, 2020.

[283] STMicroelectronics N.V. 2018. X-NUCLEO-LPM01A 1.8 V to 3. V Programmable Power Supply Source. https://www.st.com/en/evaluation-tools/x-nucleo-lpm01a.html. Last accessed: Sep. 11, 2021.

[284] Kazunobu Sumiya, Takuya Sasatani, Yuki Nishizawa, Kenji Tsushio, Yoshiaki Narusue, and Yoshihiro Kawahara. 2019. Alvus: A Reconfigurable 2-D Wireless Charging System. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2 (June 2019), 68:1–68:29. https://doi.org/10.1145/3332533.

[285] Miljana Surbatovich, Limin Lia, and Brandon Lucia. 2019. I/O Dependent Idempotence Bugs in Intermittent Systems. In *Proc. OOPSLA* (October 23–25). ACM, Athens, Greece, 183:1–183:31. https://dl.acm.org/doi/10.1145/3360609.

[286] Taiwan Semiconductor Manufacturing Company, Ltd. 2019. 0.18-micron Technology. https://www.tsmc.com/english/dedicatedFoundry/technology/0.18um.htm. Last accessed: Jan. 19, 2020.

[287] Vamsi Talla, Mehrdad Hassar, Bryce Kellogg, Ali Najafi, Joshua R. Smith, and Shyam Gollakota. 2017. LoRa Backscatter: Enabling the Vision of Ubiquitous Connectivity. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3 (September 2017), 105:1–105:24. https://doi.org/10.1145/3130970.

[288] Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua R Smith. 2017. Battery-free Cellphone. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2 (June 2017), 25:1–25:19. https://doi.org/10.1145/3090090.

[289] Jethro Tan, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. 2016. Wisent: Robust Downstream Communication and Storage for Computational RFIDs. In *Proc. INFOCOM*. IEEE, San Francisco, CA, USA, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524574.

[290] Haydn Taylor. 2020. COVID-19: The State of the Games Industry. https://www.gamesindustry.biz/articles/2020-04-09-covid-19-the-state-of-the-games-industry. Last accessed: Apr. 28, 2020.

[291] Pietro Tedeschi, Kang Eun Jeon, James She, Simon Wong, Spiridon Bakiras, and Roberto Di Pietro. 2021. Privacy-Preserving and Sustainable Contact Tracing Using Batteryless BLE Beacons. https://arxiv.org/pdf/2103.06221.pdf.

[292] Texas Instruments Inc. 2013. BQ25570 Ultra Low power Harvester power Management IC with Boost Charger, and Nanopower Buck Converter. https://www.ti.com/lit/ds/symlink/bq25570.pdf. Last accessed: Jun. 19, 2022.

[293] Texas Instruments, Inc. 2013. CC2420 Single-Chip 2.4GHz IEEE 802.15.4 Compliant and ZigBee Ready RF Transceiver. https://www.ti.com/lit/ds/symlink/cc2420.pdf. Last accessed: May 19, 2022.

[294] Texas Instruments, Inc. 2016. CC2650 32-bit Arm Cortex-M3 Multiprotocol 2.4 GHz wireless MCU with 128 kB Flash. https://www.ti.com/product/CC2650. Last accessed: Aug. 10, 2021.

[295] Texas Instruments Inc. 2016. TPS61099 0.7 V_{in} Synchronous Boost Converter with 800 nA Ultra-Low Quiescent Current. http://www.ti.com/lit/ds/symlink/tps61099.pdf. Last accessed: Apr. 25, 2020.

[296] Texas Instruments, Inc. 2017. MSP430FR59xx Mixed-Signal Microcontrollers (Rev. F). http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf. Last accessed: Sep. 13, 2021.

[297] Texas Instruments, Inc. 2018. OPT3004 Digital Ambient Light Sensor with Increased angular IR Rejection. https://www.ti.com/product/OPT3004. Last accessed: Sep. 9, 2021.

[298] Texas Instruments, Inc. 2018. TPL5111 Ultra Low Power System Timer (35 nA) for Power Gating in Duty Cycled Applications. https://www.ti.com/product/TPL5111. Last accessed: Sep. 9, 2021.

[299] Texas Instruments, Inc. 2021. BQ25570 Ultra Low Power Harvester power Management IC with Boost Charger and Nanopower Buck Converter. https://www.ti.com/product/BQ25570. Last accessed: Sep. 9, 2021.

[300] Bill Tomlinson, M. Six Silberman, Don Patterson, Yue Pan, and Eli Blevis. 2012. Collapse Informatics: Augmenting the Sustainability & ICT4D Discourse in HCI. In *Proc. CHI*. ACM, Austin, TX, USA, 655–664. https://doi.org/10.1145/2207676.2207770.

[301] Hoang Truong, Shuo Zhang, Ufuk Muncuk, Phuc Nguyen, Nam Bui, Anh Nguyen, Qin Lv, Kaushik Chowdhury, Thang Dinh, and Tam Vu. 2018. CapBand: Battery-free Successive Capacitance Sensing Wristband for Hand Gesture Recognition. In *Proc. SenSys* (November 4–7). ACM, Shenzhen, China, 54–67. https://doi.org/10.1145/3274783.3274854.

[302] Uni-Trend Technology Co. Limited. 2020. UT383 Mini Light Meter. https://www.uni-trend.com/html/product/Environmental/Environmental_Tester/Mini/UT383.html. Last accessed: Aug. 19, 2021.

[303] University of Michigan, MI, USA. 2011. UMich Moo GitHub Page. https://github.com/spqr/umichmoo. Last accessed: Apr. 19, 2020.

[304] University of Washington, Seattle, WA, USA. 2010. Wireless Identification and Sensing Platform GitHub Page. https://github.com/wisp. Last accessed: Apr. 19, 2020.

[305] Valve Inc. 2020. Source 3D Game Engine. https://developer.valvesoftware.com/wiki/source. Last accessed: May 2, 2020.

[306] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent Computation Without Hardware Support or Programmer Intervention. In *Proc. OSDI* (November 2–4). ACM, Savannah, GA, USA, 17–32. https://www.usenix.org/system/files/conference/osdi16/osdi16-van-der-woude.pdf.

[307] Virag Varga, Gergely Vakulya, Alanson Sample, and Thomas R. Gross. 2017. Enabling Interactive Infrastructure with Body Channel Communication. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4 (December 2017), 169:1–169:. https://dl.acm.org/doi/10.1145/3161180.

[308] Virag Varga, Gergely Vakulya, Alanson Sample, and Thomas R. Gross. 2017. Playful Interactions with Body Channel Communication: Conquer it!. In *Proc. UIST Adjunct*. ACM, Québec City, Canada, 81–82. https://doi.org/10.1145/3131785.3131798.

[309] Vishay Siliconix. 2020. SIP32432 Ultra Low Leakage and Quiescent Current and Load Switch with Reverse Blocking. https://www.vishay.com/docs/66597/sip32431.pdf. Last accessed: Sep. 9, 2021.

[310] Anandghan Waghmare, Qiuyue Xue, Dingtian Zhang, Yuhui Zhao, Shivan Mittal, Nivedita Arora, Ceara Byrne, Thad Starner, and Gregory D. Abowd. 2020. UbiquiTouch: Self Sustaining Ubiquitous Touch Interfaces. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 1 (March 2020), 27:1–27:22. https://doi.org/10.1145/3380989.

[311] Ju Wang, Liqiong Chang, Omid Abari, and Srinivasan Keshav. 2019. Are RFID Sensing Systems Ready for the Real World?. In *Proc. MobiSys*. ACM, Seoul, Korea, 366–377. https://doi.org/10.1145/3307334.3326084.

[312] Jingxian Wang, Chengfeng Pan, Haojian Jin, Vaibhav Singh, Yash Jain, Jason Hong, Carmel Majidi, and Swarun Kumar. 2019. RFID Tattoo: A Wireless Platform for Speech Recognition. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4 (December 2019), 155:1–155:24. https://doi.org/10.1145/3369812.

[313] Katelyn Wiley, Sarah Vedress, and Regan Mandryk. 2020. How Points and Theme Affect Performance and Experience in a Gamified Cognitive Task. In *Proc. CHI* (April 25–30). ACM, Honolulu, HI, USA, 1—-15. https://doi.org/10.1145/3313831.3376697.

[314] Wiliot. 2021. Wiliot Battery Free IoT Pixel BLE Tags. https://www.wiliot.com/product/iot-pixel. Last accessed: Jul. 22, 2021.

[315] Harrison Williams, Xun Jian, and Matthew Hicks. 2020. Forget Failure: Exploiting SRAM Data Remanence for Low-overhead Intermittent Computation. In *Proc. ASPLOS* (March 16–20). ACM, Lausanne, Switzerland, 53—-67. https://doi.org/10.1145/3373376.3378478.

[316] Allison Woodruff, Jay Hasbrouck, and Sally Augustin. 2008. A Bright Green Perspective on Sustainable Choices. In *Proc. CHI*. ACM, Florence, Italy, 313–322. https://doi.org/10.1145/1357054.1357109.

[317] Chenren Xu, Lei Yang, and Pengyu Zhang. 2018. Practical Backscatter Communication Systems for Battery-Free Internet of Things. *IEEE Signal Process. Mag.* 35, 5 (September 2018), 16–27. https://doi.org/10.1109/MSP.2018.2848361.

[318] Jian Xu, Suwen Zhu, Aruna Balasubramanian, Xiaojun Bi, and Roy Shilkrot. 2018. Ultra-Low-Power Mode for Screenless Mobile Interaction. In *Proc. UIST*. ACM, Berlin, Germany, 557–568. https://doi.org/10.1145/3242587.3242614.

[319] Xieyang Xu, Yang Shen, Junrui Yang, Chenren Xu, Guobin Shen, Guojun Chen, and Yunzhe Ni. 2017. PassiveVLC: Enabling Practical Visible Light Backscatter Communication for Battery-free IoT Applications. In *Proc. MobiCom* (October 16–20). ACM, Snowbird, UT, USA, 180–192. https://doi.org/10.1145/3117811.3117843.

[320] Wataru Yamada, Hiroyuki Manabe, and Daizo Ikeda. 2018. CamTrackPoint: Camera-Based Pointing Stick Using Transmitted Light through Finger. In *Proc. UIST*. ACM, Berlin, Germany, 313–320. https://doi.org/10.1145/3242587.3242641.

[321] Fan Yang, Ashok Samraj Thangarajan, Sam Michiels, Wouter Joosen, and Danny Hughes. 2021. Morphy: Software Defined Charge Storage for the IoT. In *Proc. SenSys*. ACM, Coimbra, Portugal, 248–260. https://doi.org/10.1145/3485730.3485947.

[322] Lohit Yerva, Brad Campbell, Apoorva Bansal, Thomas Schmid, and Prabal Dutta. 2012. Grafting Energy-Harvesting Leaves onto the Sensornet Tree. https://doi.org/10.1145/2185677.2185733. In *Proc. IPSN*. ACM, Beijing, China, 197–208.

[323] Kasım Sinan Yıldırım, Henko Aantjes, Przemysław Pawełczak, and Amjad Yousef Majid. 2018. On the Synchronization of Computational RFIDs. *IEEE Trans. Mobile Comput.* 18, 9 (September 2018), 2147–2159. https://doi.org/10.1109/TMC.2018.2869873.

[324] Kasım Sinan Yıldırım, Ruggero Carli, and Luca Schenato. 2018. Adaptive Proportional–Integral Clock Synchronization in Wireless Sensor Networks. *IEEE Trans. Control Syst. Technol.* 26, 2 (March 2018), 610–623. https://doi.org/10.1109/TCST.2017.2692720.

[325] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemysław Pawełczak, and Josiah Hester. 2018. InK: Reactive Kernel for Tiny Batteryless Sensors. In *Proc. SenSys.* ACM, Shenzhen, China, 41–53. https://doi.org/10.1145/3274783.3274837.

[326] Eren Yıldız, Lijun Chen, and Kasım Sinan Yıldırım. 2022. Immortal Threads: Multithreaded Event-driven Intermittent Computing on Ultra-Low-Power Microcontrollers. In *Proc. OSDI* (July 11–13). USENIX, Carlsbad, CA, USA, 339–355. https://www.usenix.org/system/files/osdi22-yildiz.pdf.

[327] Neng-Hao Yu, Sung-Sheng Tsai, I-Chun Hsiao, Dian-Je Tsai, Meng-Han Lee, Mike Y. Chen, and Yi-Ping Hung. 2011. Clip-on Gadgets: Expanding Multi-touch Interaction Area with Unpowered Tactile Controls. In *Proc. UIST* (October 16–19). ACM, Santa Barbara, CA, USA, 367–371. https://doi.org/10.1145/2047196.2047243.

[328] G. Pascal Zachary. 2016. The Search for a Better Battery. IEEE Spectrum, https://spectrum.ieee.org/at-work/innovation/the-search-for-a-better-battery. Last accessed: Jul. 7, 2021.

[329] Tom Zeller Jr. 2010. Green Guide to Electronics Is Disputed, but Influential. https://www.nytimes.com/2010/01/11/business/energy-environment/11green.html. Last accessed: Apr. 30, 2020.

[330] Zephyr Project. 2021. Zephyr Real-Time Operating System Source Code Repository. https://github.com/zephyrproject-rtos/zephyr. Last accessed: Aug. 5, 2021.

[331] ZF Friedrichshafen AG. 2015. AFIG-0007 Energy Harvesting Generator. https://switches-sensors.zf.com/us/wp-content/uploads/sites/7/2019/11/19_10_16-TS_AFIG-0007.pdf [specification], https://switches-sensors.zf.com/product/energy-harvesting-generators/ [summary]. Last accessed: Apr. 25, 2020.

[332] Chi Zhang, Sidharth Kumar, and Dinesh Bharadia. 2019. Capttery: Scalable Battery-like Room-level Wireless Power. In *Proc. MobiSys.* ACM, Seoul, Korea, 1–13. https://doi.org/10.1145/3307334.3326077.

[333] Maolin Zhang, Si Chen, Jia Zhao, and Wei Gong. 2021. Commodity-Level BLE Backscatter. In *Proc. MobiSys.* ACM, Virtual Event, 402–414. https://doi.org/10.1145/3458864.3466865.

[334] Pengyu Zhang, Mohammad Rostami, Pan Hu, and Deepak Ganesan. 2016. Enabling Practical Backscatter Communication for On-body Sensors. In *Proc. SIGCOMM*. ACM, Florianopolis, Brazil, 370–381. https://doi.org/10.1145/2934872.2934901.

[335] Yang Zhang, Yasha Iravantchi, Haojian Jin, Swarun Kumar, and Chris Harrison. 2019. Sozu: Self-Powered Radio Tags for Building-Scale Activity Sensing. In *Proc. UIST*. ACM, New Orleans, LA, USA, 973–985. https://doi.org/10.1145/3332165.3347952.

[336] Chen Zhao, Sam Yisrael, Joshua R. Smith, and Shwetak N. Patel. 2014. Powering Wireless Sensor Nodes with Ambient Temperature Changes. In *Proc. UbiComp* (September 13–17). ACM, Seattle, WA, USA, 383—387. https://doi.org/10.1145/2632048.2632066.