Pontifícia Universidade Católica
do Rio de Janeiro

Isabella de Freitas Lima Aguiar Mariz

Control Interfaces for Intermittent Operation in Microcontrollers Powered by Energy Harvesting

Final Project

Thesis presented to the Programa de Graduação em Engenharia da Computação, do Departamento de Informática da PUC-Rio in partial fulfilment of the requirements for the degree of Bacharel em Engenharia da Computação.

Advisor: Prof. Markus Endler

Rio de Janeiro
July 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Isabella de Freitas Lima Aguiar Mariz

Control Interfaces for Intermittent Operation in Microcontrollers Powered by Energy Harvesting

Thesis presented to the Programa de Graduação em Engenharia da Computação da PUC-Rio in partial fulfilment of the requirements for the degree of Bacharel em Engenharia da Computação. Approved by the Examination Committee:

Prof. Markus Endler
Advisor
Departamento de Informática – PUC-Rio

Prof. Noemi de La Rocque Rodriguez
PUC-Rio

Prof. Adriano Francisco Branco
PUC-Rio

Rio de Janeiro, July 20th, 2023

Isabella de Freitas Lima Aguiar Mariz

Bibliographic data

de Freitas Lima Aguiar Mariz, Isabella

Control Interfaces for Intermittent Operation in Microcontrollers Powered by Energy Harvesting / Isabella de Freitas Lima Aguiar Mariz; advisor: Markus Endler. – 2023.

51 f: il. color. ; 30 cm

Projeto Final (graduação) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2023.

Inclui bibliografia

1. Informática – Teses. 2. Internet das Coisas. 3. Energy harvesting. 4. Armazenamento de energia. 5. Computação intermitente. 6. Microcontroladores. 7. FRAM. I. Endler, Markus. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

Ao meu orientador Markus Endler pelo estímulo e parceria para a realização deste trabalho.

Ao professor Adriano Branco pela disposição e auxílio no processo de traçar o caminho seguido pelo projeto.

A PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Aos membros do laboratório GistLab, em especial João Gabriel Drumond e Pablo Nascimento, cuja contribuição foi essencial para a realização dos experimentos realizados nesse trabalho.

A minha familia por todo suporte e carinho durante todos os anos de faculdade, e em especial à minha mãe Lizia, que sempre foi minha maior torcida.

Aos meus melhores amigos Anna, Leo, Ana Vono, Duda e Pablo por toda a paciência e apoio incondicional durante os meus momentos mais difíceis.

Aos meus colegas de trabalho pela compreensão, motivação e reconhecimento de todo o meu esforço.

Ao meu cachorro Balu pelo comforto oferecido nas horas que eu mais precisava.

Abstract

de Freitas Lima Aguiar Mariz, Isabella; Endler, Markus (Advisor). **Control Interfaces for Intermittent Operation in Microcontrollers Powered by Energy Harvesting**. Rio de Janeiro, 2023. 51p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The number of globally connected IoT devices continues to rise every year. The power requirements faced by them have evolved beyond the limitations brought by the usage of batteries as the main energy source. This has paved the way for the development of a new type of power supply: energy harvesting. New challenges arose alongside it, including the intermittent operation of computer systems sustained by it. Many solutions have been presented, but related works lacked easily comparable parameters for studying their efficiency, as energy harvesting applications vary considerably. Therefore, a control interface was proposed to facilitate integration between strategies for intermittent computing and energy management. The implemented design focused more on the execution control of the microcontroller, but was evaluated experimentally with a real energy harvester. Despite the difficulties with energy harvesting architecture, a suitable solution was reached, leaving room for improvement in future works.

Keywords

Internet of Things; Energy harvesting; Energy storage; Intermittent computting; Microcontrollers; FRAM.

Resumo

de Freitas Lima Aguiar Mariz, Isabella; Endler, Markus. **Interfaces de Controle para Operação Intermitente em Microcontroladores Alimentados por Energy Harvesting**. Rio de Janeiro, 2023. 51p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O número de dispositivos IoT conectados continua a crescer. Os requesitos de energia enfrentados por eles evoluiram além das limitações trazidas pelo uso de baterias como fonte de energia principal. Isso abriu o caminho para o desenvolvimento de um novo tipo de fornecimento de energia: o energy harvesting. Novos desafios acompanharam seu surgimento, incluindo a operação intermitente de sistemas computacionais por ele alimentados. Diversas soluções foram propostas, mas os trabalhos relacionados não possuem parâmetros facilmente comparáveis para o estudo da sua eficiência, devido à variabilidade das aplicações de energy harvesting. Logo, uma interface de controle foi proposta para facilitar a integração entre as estratégias de computação intermitente e controle de energia. O projeto implementado tem seu foco maior no controle de execução do microcontrolador, mas foi avaliado experimentalmente com uma fonte real de energy harvesting. Mesmo perante dificuldades com o sistema de energia, uma solução adequada foi atingida.

Palavras-chave

Internet das Coisas; Energy harvesting; Armazenamento de energia; Computação intermitente; Microcontroladores; FRAM.

Table of contents

List of Abreviations

IoT – Internet of Things

EH – Energy Harvesting

RF – Radio-Frequency

RFID – Radio-Rrequency Identification

CPU – Central Processing Unit

MCU – Microcontroller

RAM – Random Access Memory

FRAM – Ferroelectric Random Access Memory

EDLC – Electric Double Layer Supercapacitor

EB – Electrochemical Battery

PMIC – Power Management Integrated Circuit

AC – Alternating Current

DC – Direct Current

EHCD – Energy Harvesting Computing Devices

SRAM – Static Random Access Memory

NVM – Non-Volatile Memory

NVRAM – Non-Volatile Random Access Memory

TI – Texas Instruments

IDE – Integrated Development Environment

CCCS – Code Composer Studio

CTPL – Compute Through Power Loss

, .

1
Introduction

According to the research platform Iot Analytics [Hasan, 2022, Sinha, 2023], the number of globally connected Internet of Things (IoT) devices grew 18% between 2021 and 2022, from 12.2 billion to 14.3 billion devices. As of 2023, the prediction is that this number will reach nearly 30 billion in 2027, as shown in the projection below:

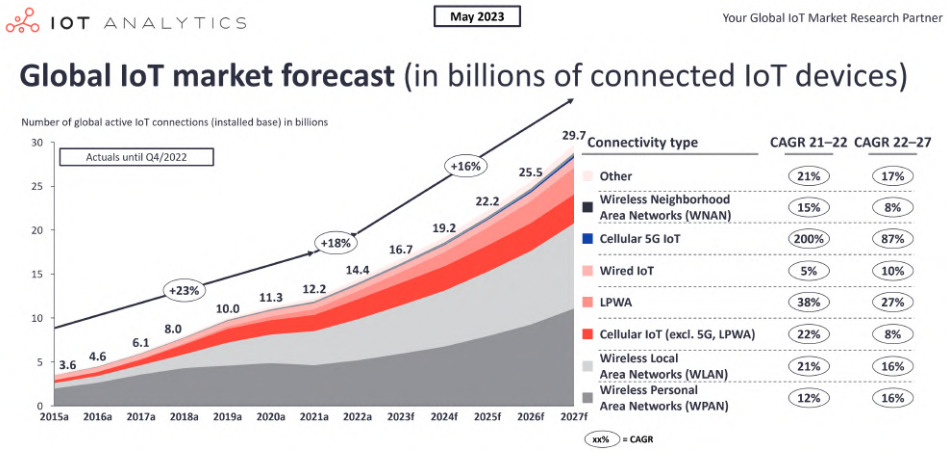

Figure 1.1: Globally connected IoT devices market prediction [Sinha, 2023]

This increase in the number of devices naturally accompanies a modern trend of significant growth in market opportunities for IoT applications. A study provided by the consulting company Fortune Business Insights on *Global IoT Market Share* [Fortune, 2023], showed that the IoT market size was valued at USD 544.38 billion in 2022, stipulating a growth of up to USD 3.35 trillion by 2030. End-of-use for this market varies, with healthcare, manufacturing and telecommunications being the most prominent industry spenders on IoT technology globally in 2022, as shown by Figure 1.2.

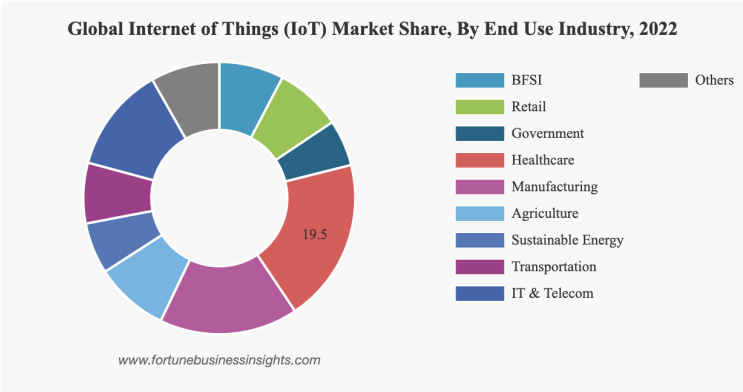

Figure 1.2: Global Internet of Things Market Share [Fortune, 2023]

Consequently, as the demand for IoT devices and solutions continues to rise worldwide, so does the interest in finding ways to power these devices as efficiently as possible. Jayakumar et al. [2014] brought forth this discussion in their publication *Powering the Internet of Things*, where it was argued that *"one of the biggest challenges to realising this IoT vision is the problem of powering these tens of billions of IoT devices"*. Unsurprisingly, this situation is still under eager debate nearly a decade later, with a recent article by the magazine Control Automation proclaiming power as the biggest challenge IoT devices face [Dietrich, 2022].

As of late, the leading solution for powering these devices has remained heavily centred around the usage of batteries [Chatterjee et al., 2023], and this could be because of numerous reasons, such as *"cost, convenience, or the need for untethered operation"* [Jayakumar et al., 2014]. The current forecast is for the IoT battery market to grow at a compound annual rate of 10.16%, rising from its current value of USD 9.5 billion in 2021 to USD 22.7 billion by 2030, as reported by market researcher and consulting organization Precedence Research [Precedence, 2022] in Figure 1.3.

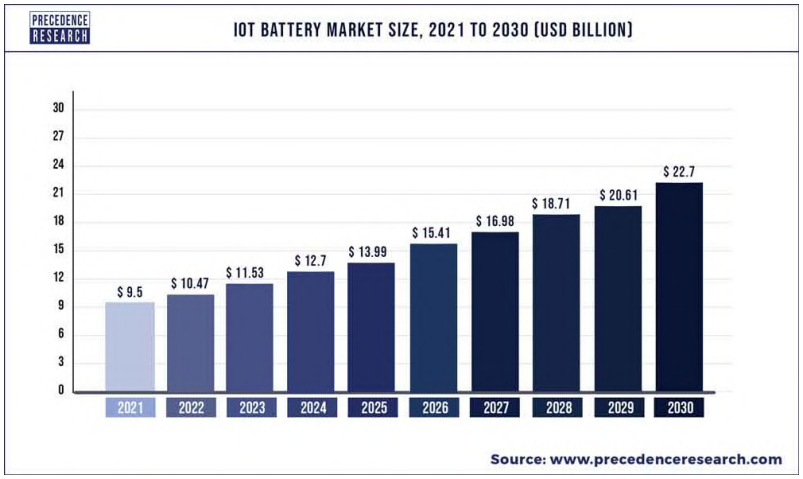

Figure 1.3: IoT Battery Market Size [Precedence, 2022]

This reality brings with it a unique set of challenges, as it is desirable that these IoT devices maintain long operational lifetimes despite facing significant difficulties that come with the use of batteries [Jayakumar et al., 2014], such as the need for frequent replacements, unreliable operation due to their natural limitations, and their negative impact on the environment. Although there appears to be *"a rich body of literature [...] on deploying battery-powered embedded sensing systems at different scales and in various environments"*, a common insight in late research is that a significant source of frustration to many of these studies is a *"hectic experience with frequent battery replacements"* [Afanasov et al., 2020].

Additionally, as the number of IoT wireless devices continues to grow at an expressive rate, a pattern of frequent battery substitution for these devices could lead to an equally expressive number of battery disposals. According to the European Union's Community Research and Development Information Service (CORDIS) [EnABLES, 2022], around 78 million batteries used to power IoT devices will be discarded globally every day by 2025. This is a significant concern, as improper disposal of batteries could bring severe consequences to the environment, since the toxic heavy metals in their composition can irreparably contaminate the soil, underground layers of water-bearing and even living organisms [Ribeiro et al., 2022]. For example, the lithium-ion battery is the most popular type among IoT devices [Partida, 2021], but the lithium in their composition has an end-of-life recycling rate lower than 1% [Statista, 2020].

Therefore, in order to mitigate the many challenges induced by powering wireless IoT devices with batteries, academic and industrial research communities have been turning to alternative powering strategies, such as Energy Harvesting (EH) [Ku et al., 2016]. This methodology consists of using a self-powered subsystem capable of harnessing sufficient power from ambient energy sources and converting it into electrical energy, such as thermal, solar, wind, vibrations, radio-frequency (RF), and others [Chatterjee et al., 2023, Elahi et al., 2020]. Hence, depending on the energy sources available and the efficiency of the energy conversion technology used, EH systems are more than capable of powering a variety of IoT devices [Chatterjee et al., 2023]. This type of technology has been often advertised as *"a direct alternative to battery-powered operation"* [Afanasov et al., 2020] and is understood to *"extend deployment lifetimes as well as reduce battery replacement and overall maintenance costs"* [Bakar and Hester, 2018].

Despite its attractive advantages, however, systems powered by EH are not without their limitations. As indicated by Bakar and Hester [2018], battery-less wireless sensing is *"essential to a sustainable future of computing"*, but these systems *"do not always have the energy necessary to maintain operation because of the unpredictability of energy harvesting"*. The amount of energy collected depends greatly on the natural inconsistency of the environment, putting these systems at risk of multiple power failures that vary in frequency and duration [Bakar and Hester, 2018, Kwak et al., 2021].

Consequently, operation in devices powered by EH becomes intermittent and so does the execution of any software running on its computer systems, a phenomenon known as intermittent computing. This way, any executing programs must continue in bursts through power failures, with periods of in-

activity when energy is not available and of execution when it is, and one cannot assume they will be run to completion before a loss of power occurs [Lucia et al., 2017]. Intermittent computing brings with it several new challenges including, but not limited to, inconsistent control flows, compromising program continuity, memory and data inconsistency and complicated communication between devices. These challenges appear to be heavily influenced by the choice of hardware and software design, and so to better understand the intermittent operation of devices powered by EH, it is important to examine how the chosen system architecture influences its own behaviour [Lucia et al., 2017].

Research towards EH applications has gained significant traction in the past few years, with studies ranging from RF [Pinuela et al., 2013, Talla et al., 2015], vibrational and thermal energy [Maiwa et al., 2012] and even kinetic energy [Magno et al., 2016], to piezoelectric [Jousimaa et al., 2016] and photovoltaic power [Simjee and Chou, 2006, Xie et al., 2021]. Moreover, low-power computing components such as Central Processing Units (CPUs) and microcontrollers (MCUs) are often present in EH deployment as a way of processing data [Lucia et al., 2017]. Great interest has been shown towards strategies to handle intermittent program execution in these components by use of techniques that adequately save and restore the application's state in the event of a power loss [Balsamo et al., 2015, Colin and Lucia, 2016, Hoseinghorban et al., 2021, Ransford et al., 2011]. For this reason, Lucia et al. [2017] highlights the following design parameters as essential for analysis: (a) energy harvesting and storage mechanisms, (b) memory system and program execution strategy, and (c) development environment.

Therefore, the final project seeks to investigate the effectiveness of different control interfaces for intermittent operation in microcontrollers powered by EH. Through a careful study of existing literature on EH architecture and intermittent software processing, the main objective is to experimentally compare the energy efficiency of possible interfaces for these systems. These will be based on a review of previous proposals for EH power systems and intermittent computing strategies, seeking to further understand the advantages and limitations of the existing procedures. Additionally, this project seeks to amend certain observed gaps in previous works, as several prominent solutions (detailed in Chapter 2) were limited to using simulated intermittent energy sources instead of real EH systems [Balsamo et al., 2015, Bhatti and Mottola, 2017, Hoseinghorban et al., 2021, Jayakumar et al., 2017, Ransford et al., 2011, Ruppel et al., 2022]. The goal is to experiment with a system genuinely powered by an EH architecture, in order to better observe and understand its

behaviour. This will be done by assembling an EH circuit that collects energy from photovoltaic cells and will be responsible for powering the chosen computational system, a microcontroller with embedded non-volatile memory. A software routine for handling intermittent computing will be implemented to run concurrently with a test program, in order to assess the effectiveness of the control interface in handling an intermittent operation.

This way, the development of the presented project makes use of extensive technical knowledge obtained throughout the undergraduate course of Computer Engineering, such as:

- Understanding the operation of electric and electronic circuits, especially those involving low-power electronics;
- Designing projects based on digital electronics techniques;
- A thorough study of computer architecture and operating systems, obtaining detailed knowledge of both computer hardware and software.
- Acquiring skills in several programming languages and different software development environments, also obtaining familiarity with constructing computational routines and algorithms.
- Extensive practice in devising and implementing projects that involve software and hardware control of microcontrollers, integrated circuits and embedded systems.

This document is structured as follows. In Chapter 2 previous work relevant to the described problem is presented. In Chapter 3 the proposed solution and project design are explained. In Chapter 4 the results of the experimental work are displayed and discussed. Finally, in Chapter 5 the final conclusions and discussions of future works are expressed.

2
Related Works

This Chapter is divided into the following sections.

Section 2.1 further elaborates on the problem introduced on Chapter 1 and the motivations behind this work.

Section 2.2 describes relevant EH architectures proposed up until recently, contemplating different energy sources, energy storage mechanisms and power management integrated circuits for controlling the energy flow through the system.

Section 2.3 surveys prominent techniques for overcoming the difficulties of intermittent computing, through an adequate choice of memory system and program execution model.

2.1
Motivation

Recent research involving IoT and wireless sensor systems have advertised about the challenges of working with battery-powered deployments. A notorious example is Afanasov et al. [2020], who experimented with battery-less embedded sensing at the Mithræum of the Circus Maximus in Italy. Their first deployment of an embedded sensing system, *Kingdom*, was fully battery-powered and it suffered numerous failures where batteries were responsible for all such cases, except for two. According to the study, the peculiar environmental conditions of the Mithræum cavern made *"predicting the system lifetime extremely difficult"*, with high humidity and fluctuations in temperature causing the alkaline batteries to *"fail unpredictably"*. Despite their efforts to use sturdier types of batteries, such as industry-grade alkaline, pro-alkaline and lithium, no improvement was observed in their system's operational lifetime and *"maintenance [represented] a hampering factor regardless of the value of the data"*. The study also pointed out that commercial chemical batteries were considered dangerous by the restorers working on site, as the natural conditions were not favourable to the physical integrity of the batteries. It was explained that *"with average relative humidity values in excess of 90% at the Mithræum, [. . .] the chances that batteries start leaking greatly increase"*, and this was not ideal for a monitoring deployment in such a sensitive environment.

Hence, it cannot be overlooked how the first deployment of the sensing system at the Mithræum was hindered greatly by their choice of a battery-powered approach. The batteries behaved quite unpredictably when faced with

unfavourable ambient conditions, leading to frequent operational failures and a constant need for maintenance. This further frustrated all involved as the sensitive nature of the Mithræum site required maintenance operations to be kept to a minimum so as to preserve the site's structural integrity and the safety of the restorers. In addition, the chemical composition of the batteries posed an environmental concern to the already sensitive environment as well as a risk to the personnel present. These challenges, however, are not limited to the scope of the previously presented Mithræum case study, and the need for frequent battery replacements continues to be a challenge for many in the fields of IoT and wireless devices. Jayakumar et al. [2014] argued that *"many IoT devices will be required to have long operational lifetimes (from a few days to possibly several years) without the need for battery replacement, because frequent battery replacement at scale is not only expensive, but often not even feasible"*. Likewise, Bi et al. [2015] further highlighted the negative aspects of maintaining this practice when explaining that *"the performance of wireless communication is fundamentally constrained by the limited battery life of wireless devices, the operations of which are frequently disrupted due to the need of manual battery replacement/recharging"*.

In an attempt to overcome these challenges, many in the fields of both academics and industry opted to seek energy sources other than batteries to power IoT and wireless sensor applications. Especially motivated by recent advances in green technology, a shift was made towards more renewable energy sources, like Energy Harvesting (EH) [Ku et al., 2016]. Afanasov et al. [2020] followed through with this exact strategy in an attempt to improve their deployment of *Kingdom*, by turning *"a battery-operated system into an energy-harvesting one"*. Its successor, *Republic*, made use of two EH strategies for powering each of their sensor systems: a thermoelectric energy generator for the temperature and humidity nodes, and piezoelectric energy harvester for the accelerometer and inclinometer. Thus, choosing EH as an alternative power source to batteries seems like a suitable strategy, as energy harvesters provide wireless communication applications with promising benefits that traditional battery power cannot offer, such as self-sustainability, no need for battery replacement and easier deployment in difficult environments [Ku et al., 2016].

2.2 Energy Harvesting Architecture

Energy Harvesting is understood as the process of converting available energy from the environment into usable electrical energy [Zeadally et al., 2020]. By means of this process, an EH system combines several subsystems

that work concurrently to generate continuous power for low-power applications (e.g. IoT) [Elahi et al., 2020]. Extensive research has been made over the years on EH architecture geared towards IoT devices, offering several techniques for collecting and storing energy, and exploring integrated circuits for power management. A review of previous literature [Afanasov et al., 2020, Balsamo et al., 2016, Elahi et al., 2020, Zeadally et al., 2020] has shown that the design of these systems is heavily characterized by the following elements:

- The **energy harvester** through which energy collected from the environment is converted into electrical energy.
- The choice of **energy storage**, when a decision is made to reserve harvested energy for future use.
- The **power management integrated circuit** responsible for balancing the generation of energy by the harvester and the consumption of power by the application's IoT device.

This architecture is presented more clearly through the block diagram in Figure 2.1, illustrating the flow of energy starting from the environmental resource, through the energy harvesting device and storage mechanism, until it is finally consumed by the endpoint IoT application device.

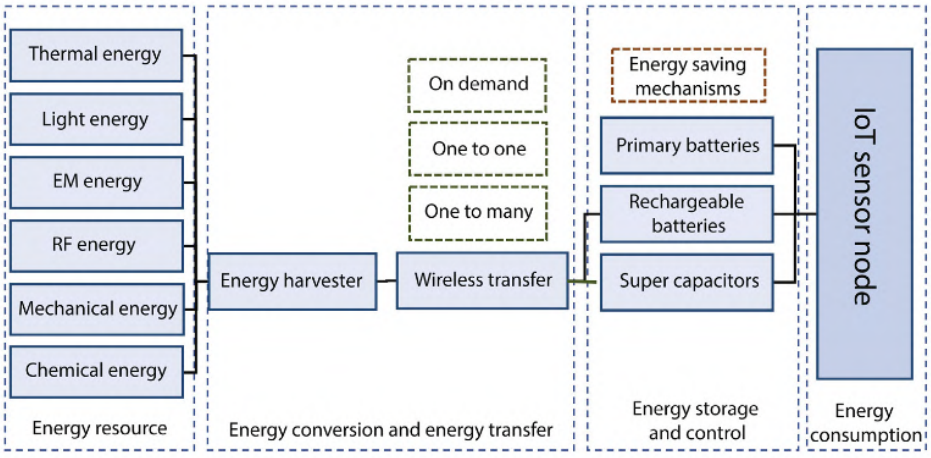

Figure 2.1: Block diagram explaining the general architecture of an EH system [Zeadally et al., 2020]

2.2.1 Energy Harvesting Devices

Energy harvesters vary significantly, as different environments offer different sources of energy from which these devices can draw power [Zeadally et al., 2020]. Each of these devices usually work with transducers responsible for the conversion of the collected energy into electrical energy, which may change according to the device's electrical characteristics [Elahi et al., 2020]. Photovoltaic cells can harvest and convert both sunlight and artificial light, for example, while RF energy can be harnessed from radio, television, Wi-Fi and cellular signals, and piezoelectric devices can collect mechanical energy from both motion or vibration.

Figure 2.2 depicts a comparison between the specific power requirements of various wireless sensors and the energy-supplying capabilities of different EH devices. It shows how one type of harvesting device is capable of supplying enough power to a significant range of applications.

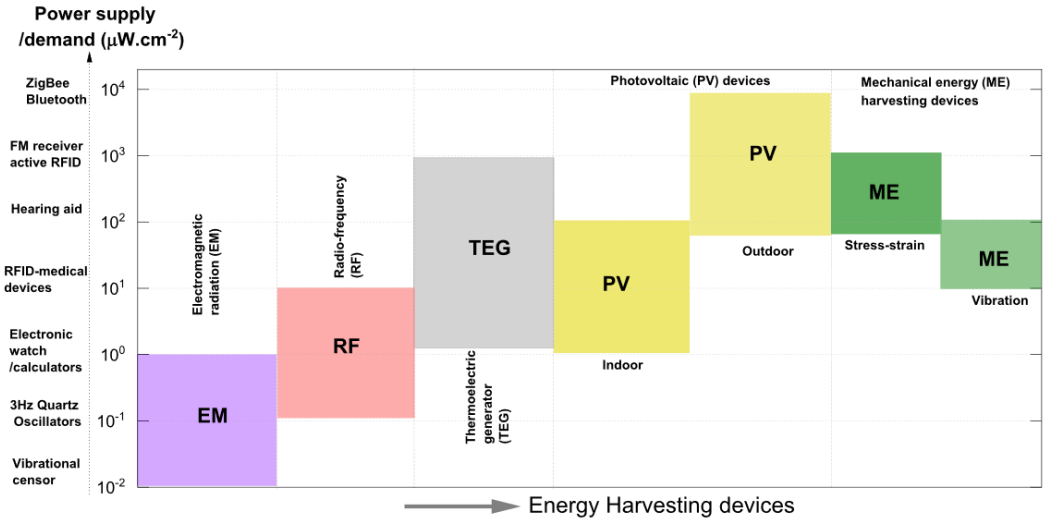

Figure 2.2: Power requirements of different wireless sensors versus energy-supplying capabilities of different harvesting devices [Chatterjee et al., 2023]

For example, thermoelectric generators are able to maintain electronic watches and calculators, as well as radio-frequency identification (RFID) medical devices. Alternatively, one single application could have a wide variety of EH choices available to it, as illustrated by hearing aids having their energy demand met by thermoelectric generators, photovoltaic devices and mechanical energy harvesters.

When designing EH systems, it is seen as important to choose an energy source that is appropriate and easily available in the environment where the application is being deployed [Zeadally et al., 2020]. For example, Pinuela et al. [2013] investigated the possibility of implementing an RF harvesting platform

in urban and semi-urban settings, where this type of energy is abundantly available. Likewise, Talla et al. [2015] proposed a wearable temperature sensor that harvests RF energy from Wi-Fi transmissions, whilst Magno et al. [2016] designed a kinetic harvester circuit to power autonomous wearable devices through energy produced by human motion.

2.2.2 Energy Storage

After energy has been properly collected and converted by the harvesting device, two main strategies could be followed: energy is put to use immediately to power the application device (known as harvest-use architecture), or it is stored in an energy buffer for future use (known as harvest-store-use architecture) [Elahi et al., 2020]. Energy storage could be used to reserve enough energy to meet the power requirements of the application's endpoint device [Zeadally et al., 2020]. The system would first accumulate energy (while consuming as little as possible), and with sufficient power stored, the system would begin operation until all energy was depleted [Lucia et al., 2017].

The choice of energy buffering device is seen as critical for the EH system being designed, as it determines the system's physical size and heavily influences its operational lifetime [Elahi et al., 2020, Lucia et al., 2017]. A wide range of energy storage devices exist, differing in properties such as power, capacity and charge/discharge rates [Elahi et al., 2020], with the two most traditional choices being rechargeable/non-rechargeable electrochemical batteries (EBs) and electric double-layer capacitors (EDLCs) [Vatamanu and Bedrov, 2015]. As previously discussed in Section 2.1, deployments of IoT applications can suffer greatly from the usage of batteries, then making capacitors the advantageous alternative. A capacitor with a higher power or energy density than ordinary capacitors is known as a supercapacitor [Zeadally et al., 2020], with its notable advantages being a large number of charge/discharge cycles [Elahi et al., 2020] and a fast recharging operation with a 98% charging efficiency [Zeadally et al., 2020]. Even more interestingly, they are known to have significantly longer lifetimes than batteries, because they use purely electrostatic processes for storing energy, although their main limitation is having a relatively low energy density when compared to batteries [Vatamanu and Bedrov, 2015].

Previous studies have shown that capacitors and supercapacitors seem to have been continuously used as energy storage devices in the deployment of EH systems, although their configuration can vary. In the early 2000s, Simjee and Chou [2006] published *Everlast*, a design for a sensor node powered by

a supercapacitor that was recharged through solar harvesting. More recently, Colin et al. [2018] proposed *Capybara*, an energy storage architecture for EH that uses an array of capacitor banks that is reconfigurable according to the system's energy demand. Another relevant project is *UFoP* by Hester et al. [2015], an energy storage system comprised of multiple independent (federated) small capacitors powered by RF and solar energy harvesters. Afanasov et al. [2020] used a single 20μF capacitor as an energy buffer in each of their sensing platforms for all deployment versions. *Morphy* by Yang et al. [2021] is a charge storage system for IoT applications composed of a polymorphic capacitor array that can be configured in different topologies by software control.

2.2.3 Power Management Integrated Circuits

Having understood the processes behind harvester devices and storage mechanisms in EH architecture, a last point of observation is the use of Power Management Integrated Circuits (PMICs).

PMICs are often used in order to control the energy flow between the energy harvester, buffer storage and endpoint IoT device [Zeadally et al., 2020], but could also help enhance the EH system's lifespan by reducing the power consumption of the IoT device [Elahi et al., 2020]. A traditional PMIC could be comprised of two subsystems, an rectifying circuit and a power converter, whose output voltage is fixed to a specified DC voltage [Ballo et al., 2021].

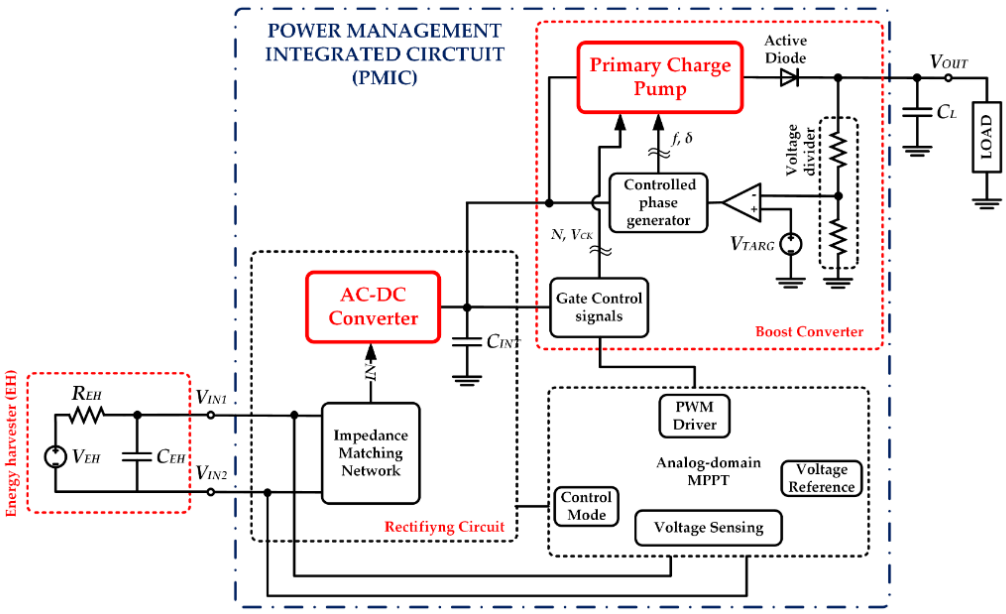

Figure 2.3: Block diagram of a traditional power management integrated circuit (PMIC) [Ballo et al., 2021].

As Figure 2.3 shows, the rectifying circuit is an AC-DC converter used to convert input alternating current (AC) into output direct current (DC), while the power converter is normally used to regulate the output voltage to the load circuit (either by boosting or limiting, depending on the application).

However, it must be taken into consideration that, because of the different electric properties of EH devices, each one might require a different type of PMIC [Ryu et al., 2019]. For example, piezoelectric and electromagnetic generators produce an AC output voltage, usually requiring output rectification. But thermoelectric generators and photovoltaic cells generate a DC output, therefore an AC-DC converter is not necessarily required, depending on the application. Illustrating even further, Afanasov et al. [2020] made use of two distinct PMIC strategies in order to manage the two EH systems deployed. For both the thermoelectric generator and the piezoelectric harvester, the BQ25570 module [Texas Instruments, 2023a] was used as both a charger for the energy buffer and an output voltage regulator between the energy harvester and the buffer. Interestingly enough, however, since the BQ25570 module only accepts positive input and the thermoelectric harvester can produce a negative output, an additional low-power rectifier was needed between them. Hence, this example clarifies the point of how PMICs can vary according to the EH system implemented and the application it is being used for.

2.3 Intermittent Computing

It has been agreed that "a primary challenge in developing IoT systems with micro-powered environmental energy harvesters is the unpredictable nature of the sources" [Balsamo et al., 2016]. As mentioned previously, the amount of power produced by energy harvesters highly depends on the application's deployment location and the specifications of the harvester itself. Consequently, energy supply becomes irregular and inconsistent, thus resulting in an unstable operation of the computer systems powered by them, i.e. intermittent computing [Colin and Lucia, 2016]. Under these circumstances, software progress becomes dependent on the system's power cycle, restarting execution from the beginning after the energy ends [Balsamo et al., 2016].

This way, previous research on intermittent computing began to characterise a new class of computer systems that specifically deal with intermittent operation as a result of an EH power supply [Colin and Lucia, 2016, Maeng et al., 2017, Majid et al., 2020, Ransford et al., 2011]. These Energy Harvesting Computing Devices (EHCDs) are usually marked by the presence of an energy buffer (typically a capacitor), that when fully charged allows the EHCD to be-

gin and continue operation until the buffer is depleted, moment at which the EHCD shuts down. Additionally, they normally execute software according to an intermittent execution model, i.e. "an intermittent execution includes the power failures, in contrast to continuously powered execution that ends at a power failure" [Colin and Lucia, 2016].

Therefore, in order to better understand how EHCDs handle the effects of intermittent operation, it is important to study the memory system and execution model techniques established for preserving software progress.

2.3.1 Memory System

A point of concern for EHCDs is that when they lose power, all volatile content is cleared from memory, including SRAM data and CPU registers, which hold vital information as to the current state of program execution [Colin and Lucia, 2016]. This way, EHCD solutions have often used non-volatile memory (NVM) to store a copy of the execution state, seeing as their data is preserved through a power loss [Maeng et al., 2017].

The first studies on intermittent computing usually used Flash as their choice of NVM, but more recently, researchers have been turning to non-volatile RAM (NVRAM) technology, such as ferroelectric RAM (FRAM). On top of saving and restoring program state more efficiently than Flash, NVRAMs can also be used as regular RAM, making them a more attractive choice [Berthou et al., 2020]. One of the founding publications on intermittent computing, *Mementos* [Ransford et al., 2011] made use of Flash memory, but more recently, many studies have worked with MCUs embedded with FRAM, such as Afanasov et al. [2020], *Hibernus++* [Balsamo et al., 2016], *Alpaca* [Maeng et al., 2017] and *Chain* [Colin and Lucia, 2016].

2.3.2 Program Execution Model

A strategy to maintain the executing program's control flow in-between periods of intermittency is also crucial to EHCDs [Lucia et al., 2017], with checkpointing and task-based systems being some of the most notable solutions to preserve a system's state and progress.

Mementos [Ransford et al., 2011] famously introduced the concept of checkpointing. Their proposed strategy is to insert function calls that estimate available energy during compile time, and during run time, *Mementos* predicts when a power loss will occur and saves the program state to NVM. This way, the execution state is restored and the program continues from where

it left off, instead of restarting from the beginning. Therefore, checkpointing is a technique that *"periodically captures a consistent system state and after a reboot resumes the execution by restoring the state captured in the checkpoint"* [Colin and Lucia, 2016]. Other noteworthy EHCD works, such as *Kingdom* by Afanasov et al. [2020] and *Hibernus++* [Balsamo et al., 2016], have also used the strategy of checkpointing to save and restore program state.

In contrast, studies such as *Alpaca* [Maeng et al., 2017] and *Chain* [Colin and Lucia, 2016] identified some disadvantages in using checkpointing methodology. It was noticed that the size of the checkpoints appears to determine the program's energy and time overhead, and so in implementations where the program state was large, checkpointing presented significant overhead and energy costs. Instead, several previous works chose to operate with task-based systems instead of checkpointing.

In the task-based approach, the programmer can decompose the application program into tasks, described as user-defined regions of code that execute in a transactional manner and over a consistent snapshot of memory [Afanasov et al., 2020, Maeng et al., 2017]. With Alpaca, Maeng et al. [2017] proposed a task-based strategy where the programmer specifies the order in which the tasks will be executed, and the program's control flow is determined by the sequence of execution of these tasks. Additionally, *Alpaca* uses a control strategy that if a power failure occurs during the execution of a task, its results are scrapped and the task is re-executed and the memory is updated before control advances to the following tasks. Afanasov et al. [2020] also implements a task-based strategy for their system *Empire*, adding an energy-aware scheduling system that only writes the results of a task's execution after it is finished and if there is sufficient energy to do so.

3
Methodology

As previously illustrated, by attempting to overcome the many challenges caused by batteries in deployments of IoT applications, a popular growing solution has been to replace them with Energy Harvesting architecture. However, this approach brings a new set of challenges, as its operation provides an unpredictable power supply to the endpoint IoT device, resulting in an intermittent execution of any running software. As such, many solutions have been proposed to tackle intermittent operations in computer systems powered by EH. The analysis made in Chapter 2 attempts to understand the similarities and differences between some of the most featured publications, in an effort to trace a baseline between them.

Ultimately, it was observed that each surveyed solution had its own application context and focus of study, and thus their approaches to a solution varied accordingly to their objectives. For example, the goal of *Mementos* [Ransford et al., 2011] was to introduce a new software strategy for checkpointing a program's execution based on the measured energy levels. Hence, no energy storage was used and they forewent testing with a real EH supply entirely. Likewise, *Hibernus++* [Balsamo et al., 2016] proposed a checkpointing strategy where they also monitored energy thresholds without the need for energy buffers. But, their system was an improvement of *Mementos*, as it self-calibrated according to the available energy and a practical validation with real EH sources was done. Meanwhile, works like *Capybara* [Colin et al., 2018] and *Chance* [Hoseinghorban et al., 2021] are both task-based solutions that use capacitors as energy buffers and monitor available energy levels, but their capacitor topology and charge management strategies still differed. As *Capybara*'s objective was to design a reconfigurable energy storage architecture, its power management technique was to calculate and store the amount of energy needed by each task, and then switch the capacitor bank configuration in case more energy was needed. However, *Chance* used a single fixed-size capacitor to store energy and managed it by using a voltage trigger controller. This controller acted as a switch between the capacitor and the computing system, disconnecting them from one another when energy was low and an imminent power failure was detected.

Therefore, when it came to integrating the power management of EH architecture, with control techniques for intermittent software execution, prior works differed significantly when specifying their chosen approach. The in-

terface between these systems varied according to each solution's focus and objectives, and no explicit definition exists as to how to properly categorize their approaches. A preliminary tabulation of the strategies used by a selected few of these related works can be seen in Figure 3.1:

Solution	Energy Monitor	Monitoring Strategy	Energy Storage	Charge Management	Execution Model	Execution Control	NVM
Mementos	Internal ADC (MCU's)	Active (MCU polls the energy supply)	None	Simulated MCU/ capacitor/EH using software	Checkpointing based on the measured energy levels	At the end of every loop or function, verify current energy levels. If below minimal threshold voltage, save the program state to NVM	Flash
Hibernus	Internal comparator (MCU's)	Passive (interrupts MCU when energy level is below or above the minimum voltage thresholds) .	None	Assumes that the MCU's development board has enough decoupling capacitance as to not require additional energy storage	Checkpointing (based on the measured energy levels)	When below threshold voltage, saves the program state and enters deep sleep (hibernate). When above threshold voltage, program state is restored and execution resumes (restore)	FRAM
Hibernus++	External comparator circuit	Passive (interrupts MCU when energy level is below or above the minimum voltage thresholds)	None	Like its predecessor Hibernus, uses the development board's decoupling capacitance	Dynamic checkpointing (based on the measured energy levels)	Same as Hibernus, but the hibernation and restoring thresholds are dynamically adjusted according to the system's power consumption and on-board decoupling capacitance	FRAM
Chance	Internal ADC (MCU's)	Passive (controller interrupts MCU when energy supply is below the minimum threshold)	Capacitor	Capacitor voltage trigger controller that disconnects MCU from EH supply when capacitor voltage drops below the threshold voltage	Task-based system (estimates the energy consumption of the tasks to estimate the minimum energy required by each one)	Saves system state every time a task is successfully executed. In case of a power failure, the system re-executes any unfinished tasks in the next power cycle.	FRAM
Capybara	None	None	Array of capacitor banks	Programmer chooses the number of banks and the energy capacity of each bank according to their target application	Task-based system (each task is annotated with their unique energy requirements relative to the configured system capacitance)	When a task starts executing, Capybara issues a command to the power system to configure the energy storage to the capacity required to execute it. When the buffer is full, the task executes. it assumes that energy storage will reconfigure itself to meet task energy requirements.	None
UFoP	External comparators and ADCs	Active (MCU monitors the energy supply of the peripherals)	Federated capacitors (multiple independent small capacitors, one for core processing and one for each peripheral)	Capacitors are charged and used independently of the charge state of other components. A charge controller circuit is responsible for turning on/off the MCU and charging an array of peripheral capacitors.	Task-based system (capacitor voltage thresholds are calculated for the energy required by the tasks)	Tasks are executed when there is enough stored energy in the capacitors to process them all to completion. The system assumes that new energy will be harvested and stored to replace the energy being used during the task execution.	None

Figure 3.1: Classifying related works according to their Charge Management Interface and Program Execution Control.

From the examinations summarized in Figure 3.4, it was observed that the proposed solutions could be classified according to the two following elements:

1. **Charge Management Strategy -** Refers to the techniques adopted for controlling the energy storage mechanism, including
   - the type of energy storage mechanism e.g. a single capacitor, super-capacitors, capacitor banks, polymorphic or federated capacitors.
   - the energy monitoring strategy chosen to observe the energy levels, normally by use of hardware devices, e.g. analogue-digital converters (ADCs), voltage comparators. Two monitoring strategies were isolated from the available works:
     - **Active** - The computer system is actively monitoring to know when the energy supply is too low to continue software execution.
     - **Passive** - The computer system is informed (e.g. by an interrupt routine) when the energy supply is too low and/or when there is sufficient energy again to continue software execution.

2. **Program Execution Control -** Refers to how the computer system progresses with program execution given the intermittent energy supply, by use (or not) of the already proposed execution models (e.g. checkpoints or tasks). The challenge here is understanding how to best implement a *secondary* program, that operates on top of the main program in order to save/restore its state, whilst also being affected by the same energy intermittency. This includes

   – the technique used for saving and restoring the main program's state, examining what volatile data must be preserved and how to persist it on non-volatile storage.
   – the strategy of when to call this secondary routine in relation to the execution of the main program.

Ideally, these factors can be used to properly organise and categorize solutions that study the control between an intermittently operated computer system and the EH architecture that powers it. With this system, the main objective is to help outline a proper classification scheme for deployments of MCUs intermittently powered by EH, based on their choice of control interface, which is characterised by its charge management strategy and program execution control. This way, future works that strive to further evolve the current solutions can have a clearer grasp of where to begin.

Hence, the main objective of this final project is to devise and implement an example of a simple control interface for a microcontroller operating under intermittent operation and powered by an EH architecture. The goal is to evaluate experimentally the performance of the presented system, in order to better understand the advantages and limitations of its chosen methodology. The hope is that this project will serve as the beginning of a road map, which will facilitate comparison between different control interfaces based on the efficiency of their chosen architecture. Due to the broadness of the proposed control interface methodology, however, the study parameters of this project are limited solely to the examination of the Program Execution Control element. No practical study was made as to the Charge Management Strategy of the control interface.

3.1 Technology and Platforms

In order to replicate a control interface for computer systems powered by EH, a choice had to be made as to what technologies and platforms would be used to implement each system. Regarding the microcontroller, a desire

was to use FRAM as the choice of NVM, because of its benefits over other memory systems, such as Flash. The choice for the *Texas Instruments* (TI) MSP430 was an attractive one, as they had been known for developing fine MCUs with embedded FRAM, as well as having ready-to-buy development kits that facilitated practical experimentation. As to the EH architecture, it was desired to keep as uncomplicated as possible, with an energy harvester that could easily collect energy from the application's deployment location and an uncomplicated storage system.

Thus, based on the previously defined parameters, the following technologies were chosen for developing the intended testbench:

1. The EH architecture is a circuit comprised of photovoltaic cells as an energy harvester, a CJMCU-2557 as a PMIC [AliExpress, 2023] and a 1mF capacitor as an energy storage.

2. The computing system is the development board of the MSP430FR5994 microcontroller by *Texas Instruments*, with embedded 256KB of FRAM and 8KB of SRAM [Texas Instruments, 2023c].

3. The integrated development environment (IDE) is *Code Composer Studio* (CCS) [Texas Instruments, 2023b], the *Texas Instruments* official platform for programming and debugging its MCUs.

4. *Windows* was chosen as the preferred operating system as it handled CCS's user interface better.

5. Programming is done on CCS with both C/C++ and assembly languages being used.

3.2 Understanding the MCU and IDE

Since the focus of the project is the Program Execution Control system, the logical first step was to understand how to work the microcontroller and the IDE. Although previous experience was had in programming microcontrollers of all levels of computer language (e.g. Arduino, RaspberryPi, NodeMCU ESP8266), the *Texas Instruments* MCU and IDE were not one of them. A simplifying factor, however, is that CCS is an Eclipse-based IDE and previous background with programming Java in the Eclipse platform facilitated the initial operation of CCS. Hence, project development began with understanding how to program the MSP430FR5994 development kit through the operation of CCS.

By reading through the appropriate MCU and CCS datasheets, and reading through example programs offered by TI, the following key findings were obtained:

- The MSP430FR5994 development board offers an on-chip debugger and bootstrap loader, discarding the need for external devices for booting and debugging the microcontroller.
- The board also contains an embedded energy monitoring system called *EnergyTracing Technology*, which allows the user to trace the energy consumption of the MCU's systems.
- During debugging mode with the execution paused, the user can make use of the *Memory Browser* and *Register* features offered by CCS to visualize the MCU's memory contents.
- CCS allows users to pick which compiler will be used for the project, with the main ones being its default TI C/C++ compiler or the GNU C compiler (GCC). The current project chose to use the default TI compiler.
- The default TI compiler offers a configurable module called linker command file [Texas Instruments, 2023f], where the user can manually configure the MCU's memory map for the running application.

These were all features of interest, as they allow the user to trace an energy profile for the running programs, configure the memory sections as the best fit for the application being developed, and provide a friendly display of the content being written into memory and registers. The latter was especially useful when debugging a program's execution flow, as it showed a step-by-step of the values altered in the memory system as the program progressed.

Figure 3.2 shows a reference picture for the CCS's user interface, displaying the *Memory Browser* and *Register* windows:

3.3 Program Execution Control

The first part of understanding how a control interface implements a Program Execution Control mechanism, is studying the technique used for saving and restoring the executing program's state. As stated in Section 3.4, this is done by assessing what volatile data should be preserved and how it will be persisted in non-volatile storage. Hence, this part of the project's development focused on creating a secondary program that would save the necessary volatile data of the primary program into the MSP430FR5994's

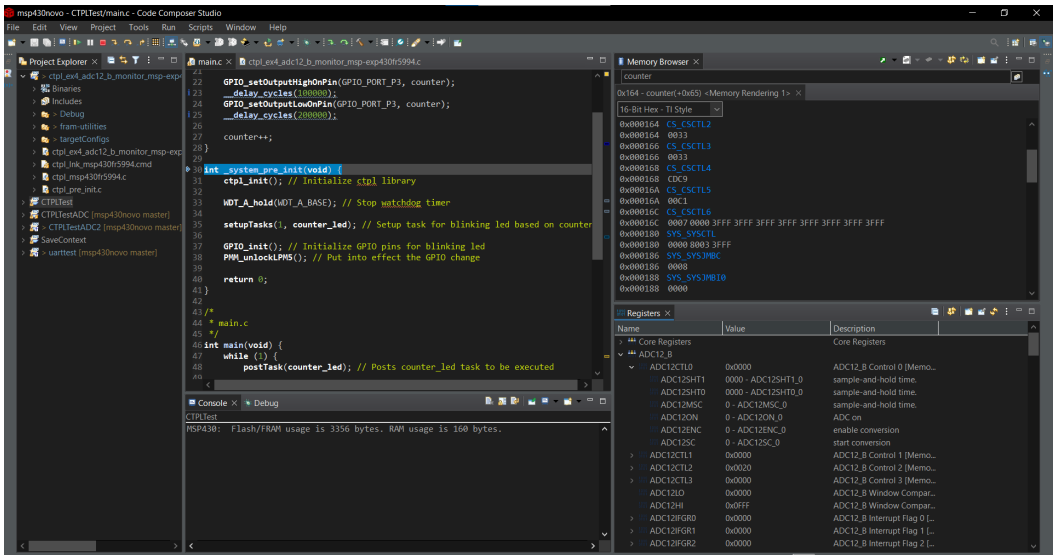

Figure 3.2: User interface for *Code Composer Studio*, TI's IDE for MCUs.

FRAM. The second part, however, would be choosing an adequate strategy for executing this second program concurrently with the primary program it seeks to preserve the context of, as they will both be affected by the energy intermittency.

This development was split into different processes, listed below:

1. **Save-and-Restore Library -** In order to develop a software routine that saves and restores a program's state, the following steps were taken:
   - *Mementos*[1] and *Hibernus++*[2] both made their software available in public online platforms and so the first approach was to study their code to identify what data needed to be saved and how they saved it.
   - The next step was to try and replicate their logic for the MSP430FR5994 specifically, determining what information needed to be saved and how it could be written into the FRAM.
   - Then, an attempt was made to develop a library from scratch based off *Mementos* and *Hibernus++*, that would be used to copy contents from non-volatile sections of memory into the FRAM and correctly restore it back, so the execution would continue where it left off after a power failure.
2. **Intermittent Execution Model -** Having a library that saves and restores the execution state, a strategy was developed for testing this routine with an example test program.

[1]https://github.com/spqr/mementos/tree/wolverine
[2]http://www.transient.ecs.soton.ac.uk/transient.php

3.3.1 Case Studies: Mementos and Hibernus++

Firstly, it was noticed that *Mementos* and *Hibernus++* used a somewhat similar strategy to saving and restoring program context. They both evaluated that it would be necessary to save the CPU registers (Program Counter, Stack Pointer, Status Register and General Purpose Registers), CPU stack and any required RAM data, which are all erased when power to the MCU is shut off. Both applications did experiments on the MSP430FR framework, where global and static variables are allocated in the RAM by default, whilst constants and program code are stored in the main non-volatile memory (FRAM, ROM or Flash). This is equally true for the MSP430FR5994, with constants and program code being stored in the FRAM by default. Therefore, in order to properly restore the application execution state after a power loss in the target MCU, the save-and-restore program must save the CPU registers and stack, as well as the global and static data stored in the RAM.

Mementos and *Hibernus++* followed this same procedure, but while *Hibernus++* decided to save the entire RAM content, *Mementos* only saved the necessary sections of RAM that held the static and global variables. *Hibernus++* followed a less complex approach, hard-coding the origin and destination addresses of the relevant memory locations into their program, and using in-line assembly code in their C file to call these routines (Figure 3.3). To decide where in the FRAM to save the copied data, they used a dedicated section of the FRAM that was manually configured in the MCU's linker command file.

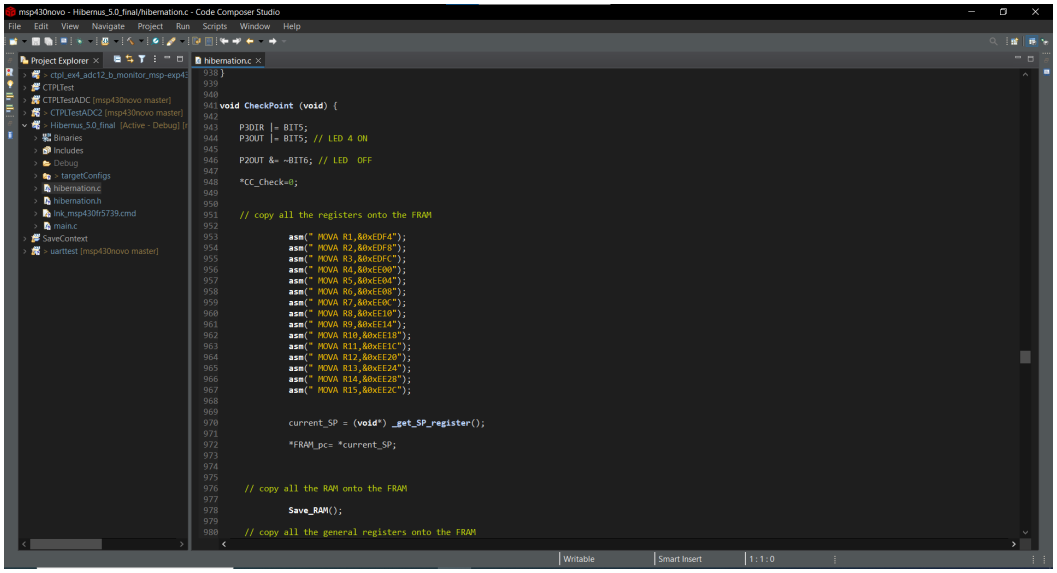

Figure 3.3: A snippet of Hibernus++'s save-and-restore program state routine.

Mementos, however, uses a more sophisticated methodology by dynam-

ically calculating the size of the data it needs to allocate into the NVM. To save the CPU registers, they use in-line assembly instructions to pile them into the CPU stack. Since they knew where the original stack began, they now have the original stack size summed with the size of the CPU registers. Next, they calculate the size of the RAM section used by global and static variables. Lastly, having estimated the total size of the data they need to save, they now iterate over the FRAM to allocate the CPU and RAM data, starting at the next available address (that they also calculate). This strategy is illustrated in Figure 3.4.

This way, unlike *Hibernus++*, *Mementos* does not need to save the entire RAM content into the FRAM, as they calculate and allocate only the space they need. This makes *Mementos* approach more efficient in terms of memory usage, but they do have the disadvantage of having to recalculate the addresses of the memory sections every time. In contrast, *Hibernus++* appointed an available address range in the FRAM separated exclusively for holding the CPU registers, stack and the full RAM data they sought to save. This way, no address calculations were necessary and the configuration of the memory map guaranteed a dedicated memory section. Not to mention that, by saving the entire RAM, all its data is preserved and there is no risk in calculating the wrong address for restoring the data later. However, if the memory map changes, the user needs to manually alter the hard-coded address values in the code, which would not be an issue for *Memento*'s implementation.

3.3.2 Save-and-Restore Library

Therefore, having properly mapped the strategies used by *Mementos* and *Hibernus++* to save and restore program context, the next step in the project's development was to make an effort to implement a similar and simpler routine from scratch, without having to rely on the specificities and framework constraints of *Mementos*'s or *Hibernus++*'s code.

A choice was made to base this new save-and-restore library on the approach followed by *Hibernus++*, mainly due to their simplicity in methodology when compared to *Mementos*'s strategy. The memory addressing was easily configurable in the linker command file and having to save the entire RAM was seen as less of a disadvantage when considering that the MSP-EXP430FR5994 has a dedicated FRAM storage of 256KB while the RAM size is merely 8KB. Not to mention, *Mementos* makes use of the GCC compiler which does not have the same linker command file as the TI compiler. This made it harder to understand the memory addressing calculations they made, let alone how

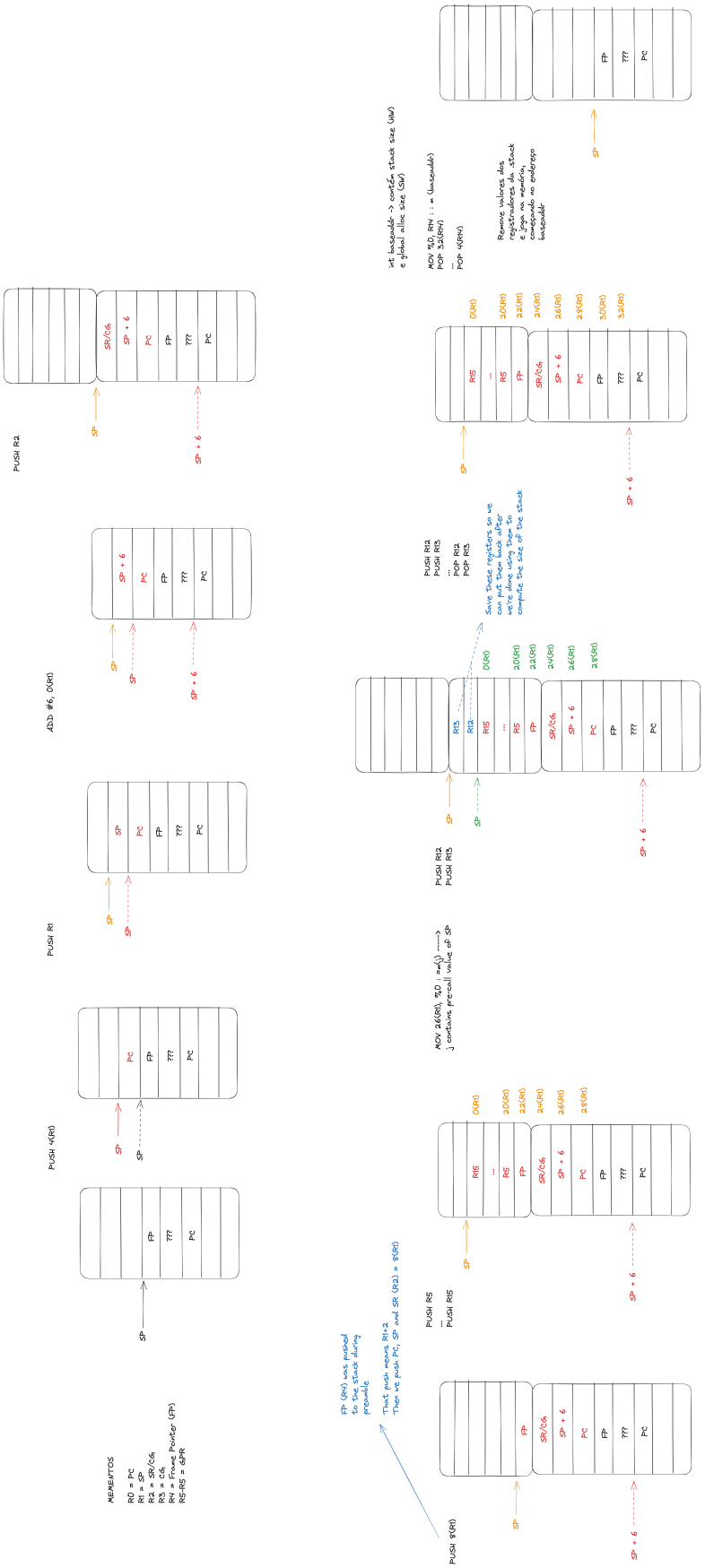

Figure 3.4: A diagram explaining Memento's save-and-restore program context routine.

to replicate them, so following *Hibernus++* strategy felt like a more clear and uncomplicated approach.

Following this strategy, the MCU's linker command file was reconfigured to create a dedicated section in the FRAM for the volatile data that the library needs to save and restore. Considering the physical FRAM addressing space ranges from 0x004000 to 0x043FFF (with a dedicated section for interrupt vectors between 0x00FF80 and 0x00FFF), Figure 3.5 shows the designed memory map for the prototype memory library. In total, 10KB were assigned to store the RAM's 8KB data (FRAM_RAM), while 64B were used to store the CPU registers. Since the MSP430 processor was configured to use 20-bit sized registers (except for the Status Register with 16 bits), a total of 15 CPU registers would need 38B, but extra space was allocated just in case, totalling 64B (FRAM_REG).

MEMORY SECTION (.cmd)	START	END	SIZE	SAVED ON
TINYRAM	0xA	0x20	0x16	RAM
BSL	0x1000	0x1800	0x800	ROM
INFO D	0x1800	0x1880	0x80	FRAM
INFO C	0x1880	0x1900	0x80	FRAM
INFO B	0x1900	0x1980	0x80	FRAM
INFO A	0x1980	0x1A00	0x80	FRAM
RAM	0x1C00	0x2C00	0x1000	RAM
LEARAM	0x2C00	0x3AC8	0xEC8	RAM
LEASTACK	0x3AC8	0x3C00	0x138	RAM
FRAM	0x4000	0xFF80	0xBF80	FRAM
INTERRUPT VECTORS AND SIGNATURES	0xFF80	0x10000	0x7F	FRAM
FRAM_REG	0x10000	0x10040	0x40	FRAM
FRAM_RAM	0x10040	0x12840	0x2800	FRAM
FRAM2	0x12840	0x43FF8	0x317B8	FRAM

Figure 3.5: Memory map outlined for the save-and-restore context prototype library.

Having mapped the address range for allocation in the FRAM, a C module was developed that called the same in-line assembly function as *Hibernus++* and *Mementos*, in order to move content from the registers and RAM addresses to the mapped FRAM destination. Figure 3.6 shows a snippet of that code[3], specifically implementing save and restore functions for the CPU registers.

[3]https://github.com/bellamariz/MSP430FR-SaveContext

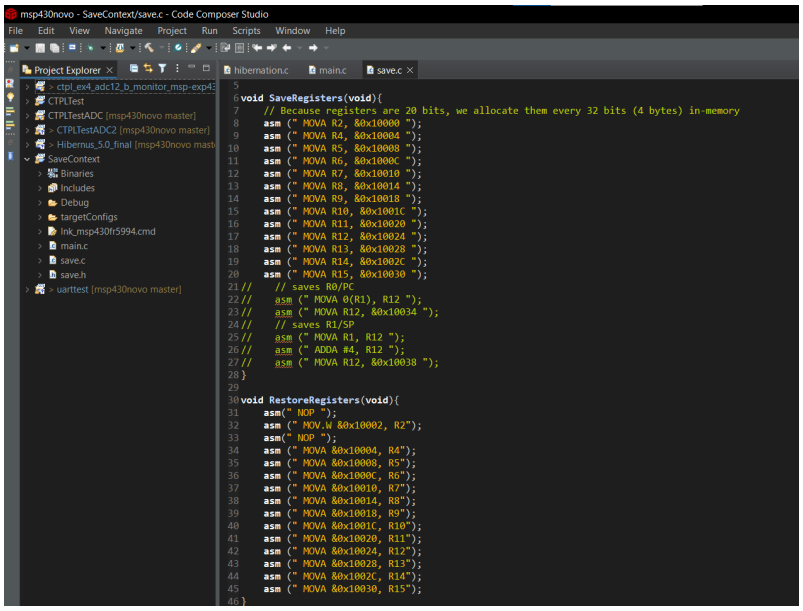

Figure 3.6: Snippet code of the save-and-restore context prototype library.

Nevertheless, despite a strong effort to replicate *Hibernus++*'s strategy and create an independent save-and-restore context library, several difficulties and limitations were observed during development. For example, the value of the Status Register was being saved correctly in the FRAM, but every time the system was rebooted, the last byte of its value kept getting overwritten in the FRAM address where it was stored. The source of this issue was investigated, suspecting an overwrite triggered by the MCU's reset or a configuration of the assembly instruction used. However, no satisfactory solution was found at the time.

Moreover, it was felt that the library developed did not adequately implement the full objective of a save-and-restore context routine. Although critical volatile data from the CPU and the RAM was handled decently, no strategy was developed to handle the persistence of any peripherals, as that would take an even higher complexity. As a consequence, it was necessary to follow a different strategy and a solution was found in the TI official API package *FRAM Utilities*.

3.3.3 Computing Through Power Loss

It was discovered that TI microcontrollers from the MSP430FRx family can make use of the *FRAM Utilities* library, a collection of software modules created by *Texas Instruments* to help users develop with the MCU's FRAM. One of the utilities provided was an API called *Compute Through Power Loss* (CTPL) that "allows an application to save and restore critical system com-

ponents when a power loss is detected" [Texas Instruments, 2023d]. Therefore, this API seemed like a suitable alternative for a save-and-restore context library, and so it was sought to understand how it worked in order to rate its applicability in the final project.

The CTPL API offers two main functions, *ctpl_enterLpm* and *ctpl_enterShutdown*, which when called save to FRAM: the state of all peripheral registers, the context of the CPU and the active stack to non-volatile RAM. The difference between them is:

- *ctpl_enterLpm* - After being called, program execution is halted, the current system state is saved to FRAM, and the MCU enters into sleeping mode. The restoring routine will occur and execution will resume only on a device reset/power on or if an interrupt event is detected.
- *ctpl_enterShutdown* - After being called, program execution is halted, the current system state is saved to FRAM, and the MCU waits in active mode for an imminent shutdown. Here, interrupt and wake-up sources are disabled, the restoring routine will occur and execution will resume only on a device reset/power up, or if the timeout value passed as parameter is exceeded. That is, after the MCU begins the shutdown, if the timeout value is exceeded and no power loss occurs, the saved state is restored and execution resumes.

Notably, when a system state is saved, it is flagged as valid, and when it is restored, the API marks it as invalid. This means that, once a state was restored, it cannot be used again, and program execution must wait until the next API call is made to have the system state saved to FRAM. This is because the API was designed specifically in use cases where a loss of power is a *certainty*, not just a possibility, and that's why its strategy involves stopping the execution of the MCU on top of saving the system state. Despite this, it was understood that the CTPL library could be well-adapted and utilized as a save-and-restore library for this final project.

3.3.4 Program Execution Model

With a library selected for saving and restoring the execution state of a program after a power loss occurs, the next development stage in completing the interface's Program Execution Control is to define an execution model. Here, the challenge is how to efficiently use the proposed library in relation to the concurrent execution of the primary program it operates over. That is,

how to guarantee the execution of a program that is used to persist the state of *another* program, in spite of both being at risk of a power loss.

At this point, it was necessary to define a test program that would be used for evaluating the performance of the chosen save-and-restore library. Seeking to begin with a naive simplistic approach, a code was developed for a 3-bit counter that posts the counter value to three GPIO ports, each connected to a LED to display the counter value. This counter is a global variable that resets every time it reaches eight (8) since three bits in binary can only count up to seven (7). Additionally, since the standard for programming with the MSP430FR5994 microcontroller is to operate directly on the MCU registers, an additional TI library called *DriverLib* [Texas Instruments, 2023e] was used as a higher-level language interface for the register bitwise operations. This library was used to initialize and alter the state of the GPIO pins in the MSP430FR5994 development board.

Then, with the test program created, an execution model was needed to control the execution flow between this program and the save-and-restore library. It was decided to implement a task-based model, where the tasks would be set up, posted for execution and then processed by a scheduler. This scheduler receives pointers to functions (tasks) and queues them for execution, only processing them one at a time. Therefore, the main test code consists of an eternal loop that calls the scheduler function that processes the posted tasks. This solution schedules only one task for in-loop execution, the counter function, so each task is an instance of the counter being incremented and posted to the GPIO pins. The counter function posts itself for future processing every time it is called, guaranteeing that there will always be a task to be processed. Figure 3.7 shows the complete code[4] for the main test program.

[4]https://github.com/bellamariz/MSP430FR5994-SaveRestoreContext

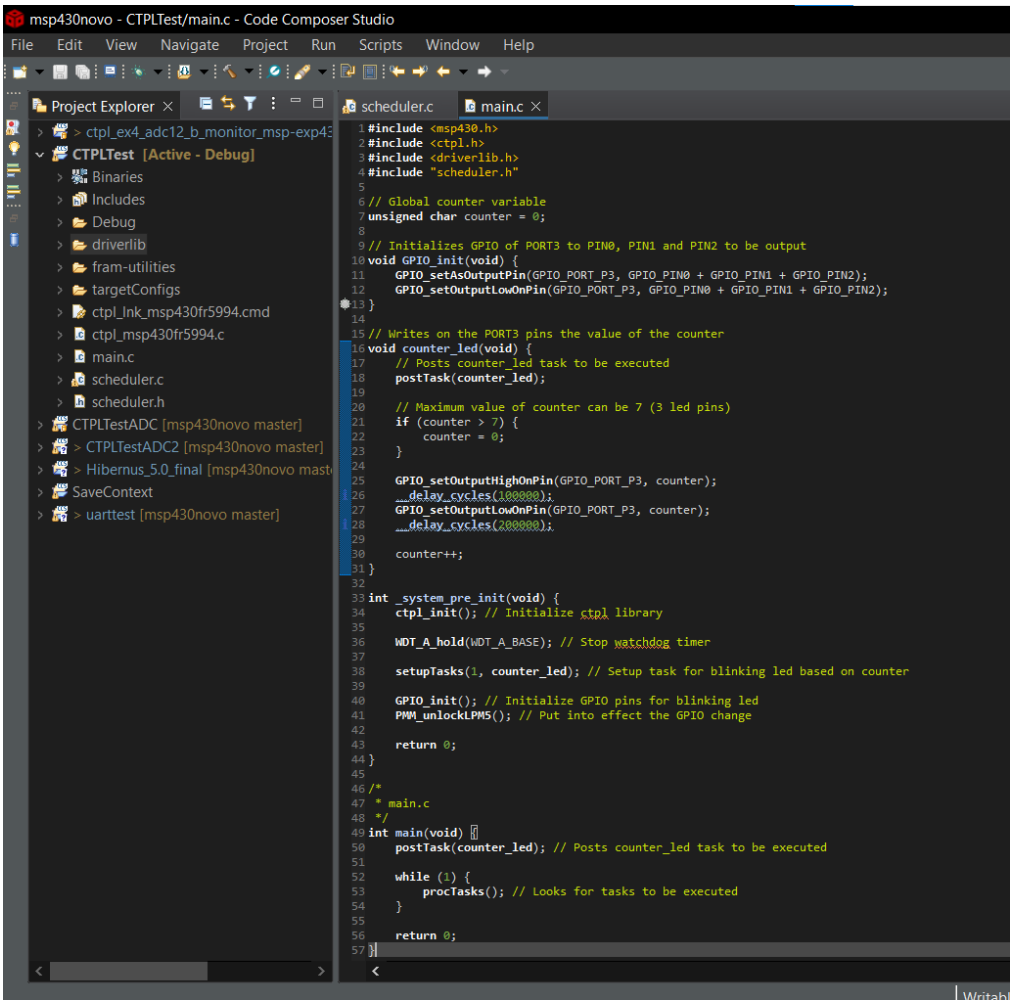

Figure 3.7: Code snippet of the developed test code.

This way, all there was left to evaluate is where to place the call to the CTPL library, in order to save and restore the system state through a power loss. The examined related works about task-based systems normally processed the sets of tasks atomically, by executing them one at a time (as was done here), but also by waiting for the successful completion of a task before executing the next one. Seeking to somewhat reproduce this technique, it was decided that the call to the CTPL library would be done inside the scheduler function that processes the tasks, placed right before the execution of the next task. This code can be seen in Figure 3.8.

The chosen CTPL function was *ctpl_enterShutdown* with the timeout parameter, and a modification was made to the library so that the saved states would not be invalidated after restoring. This way, each time a task is successfully executed, the CTPL library saves the current system state to the FRAM. If a power loss occurs before the timeout is reached, the system state will be restored when the MCU powers on again. However, if the CTPL detects a timeout and energy is still available, it restores the saved system state

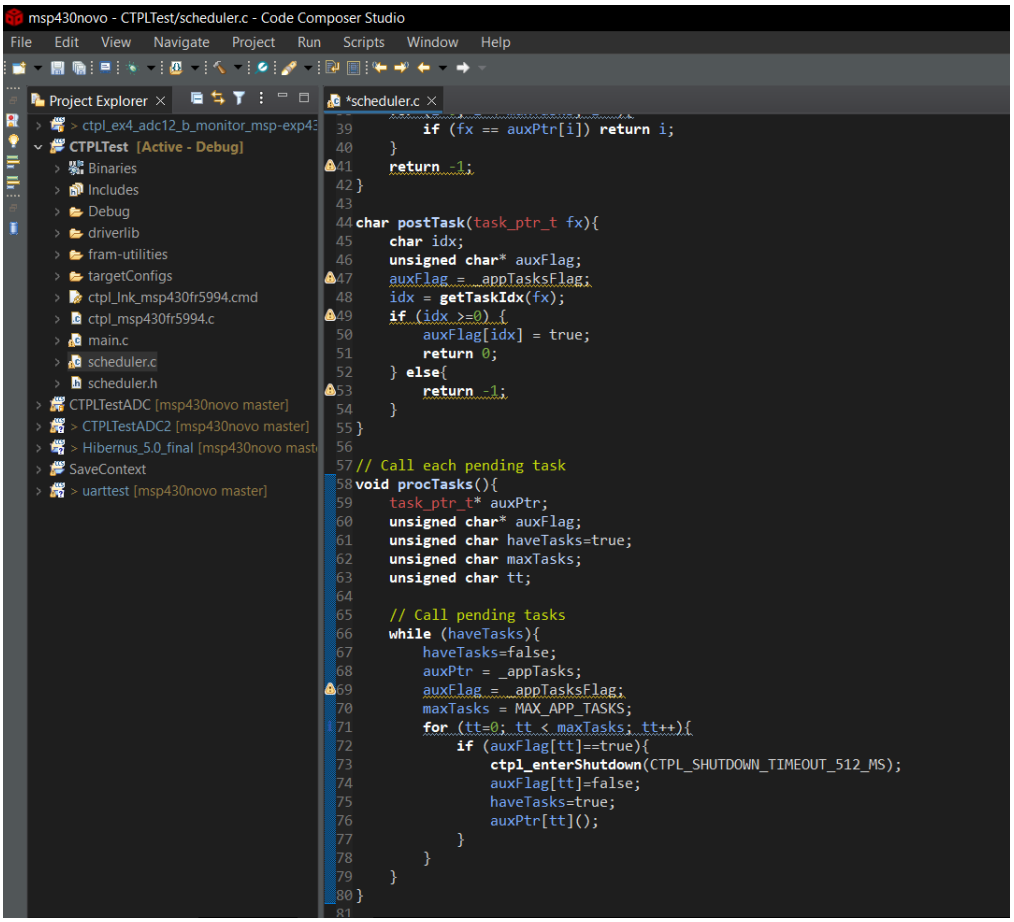

Figure 3.8: Call to save-and-restore function inside the scheduler model.

and continues the execution flow for the next queued task. If a power failure occurs during the execution of a task, the system state is saved. When the MCU eventually powers on again, the restored context identifies the last executed task as the one that was finished *before* the one that was executing when the power loss occurred. Therefore, when the system powers on, it re-executes the task that was interrupted from the beginning, instead of continuing execution in the middle of the task where it was cut off.

3.4 Project Design of Control Interface

At the beginning of this Chapter, the concept of a control interface for computing systems powered by EH was introduced. It was outlined, that based on an analysis of related works, a proposal for this interface was defined by two main systems: the charge management of the EH system's energy buffer, and the methodology chosen for controlling software execution given the power intermittency. Since the scope of the final project was narrowed to focus on the development of the program execution control, this report does not approach detailed solutions for charge management strategies for EH systems.

Nonetheless, one of the main project objectives was to implement and evaluate an example of control interface with a real EH architecture. In Section 3.1, the proposed EH technology was introduced, and these can be drawn together to the MCU system to compose the following circuit:

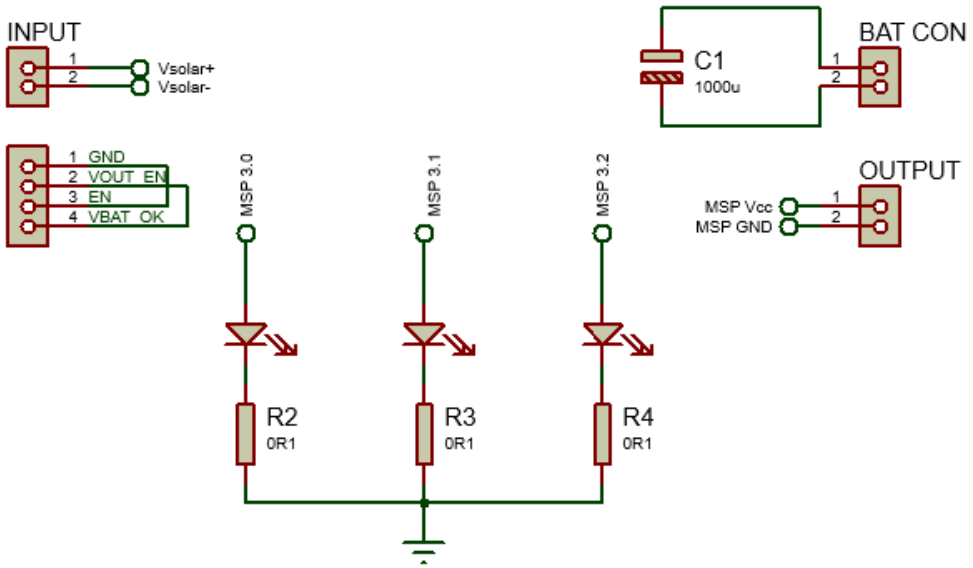

Figure 3.9: Schematic of the proposed control interface.

As shown in Figure 3.9, the energy harvester implemented was a photovoltaic cell, given the usual abundant sunlight availability in the deployed location. The harvester is then connected to the power management device CJMCU-2557, that has an internal power converter with a 3.3V reference voltage. This device has two outputs: (1) one to charge the 1mF capacitor chosen for energy buffer (VBAT) and one to supply to the MSP-EXP430FR5994 board (VOUT). When the stored voltage is higher than the reference voltage, the device converts this exceeding voltage into electrical current to optimize the system's power. The usage of this technique is important, since the operating voltage of the MCU is between 1.8V and 3.6V, and the capacitor is capable of providing a higher voltage than that.

Finally, connected to the MCU through three GPIO pins, there is 3-LED circuit used to evaluate the operation of the counter test program. As explained in Section 3.3.4, the program counts up to seven (7) before resetting the count to zero (0) and beginning again, and the counter progress is represented by three LEDs in binary. Each LED was naturally coupled with a resistor, and although a resistance of 100Ω was tested and proven to be enough, it was decided to use a value as high as possible without compromising the LED's brightness. This is because, given the scarcity of energy in EH deployments, a high resistance would guarantee a lower current and, thus, a lower power

consumption for the same voltage value. At the end, after testing a few different values, it was decided that a 4.7kΩ resistor was enough.

In the end, the final project's experimental model can be summarized as illustrated in Figure 3.10.

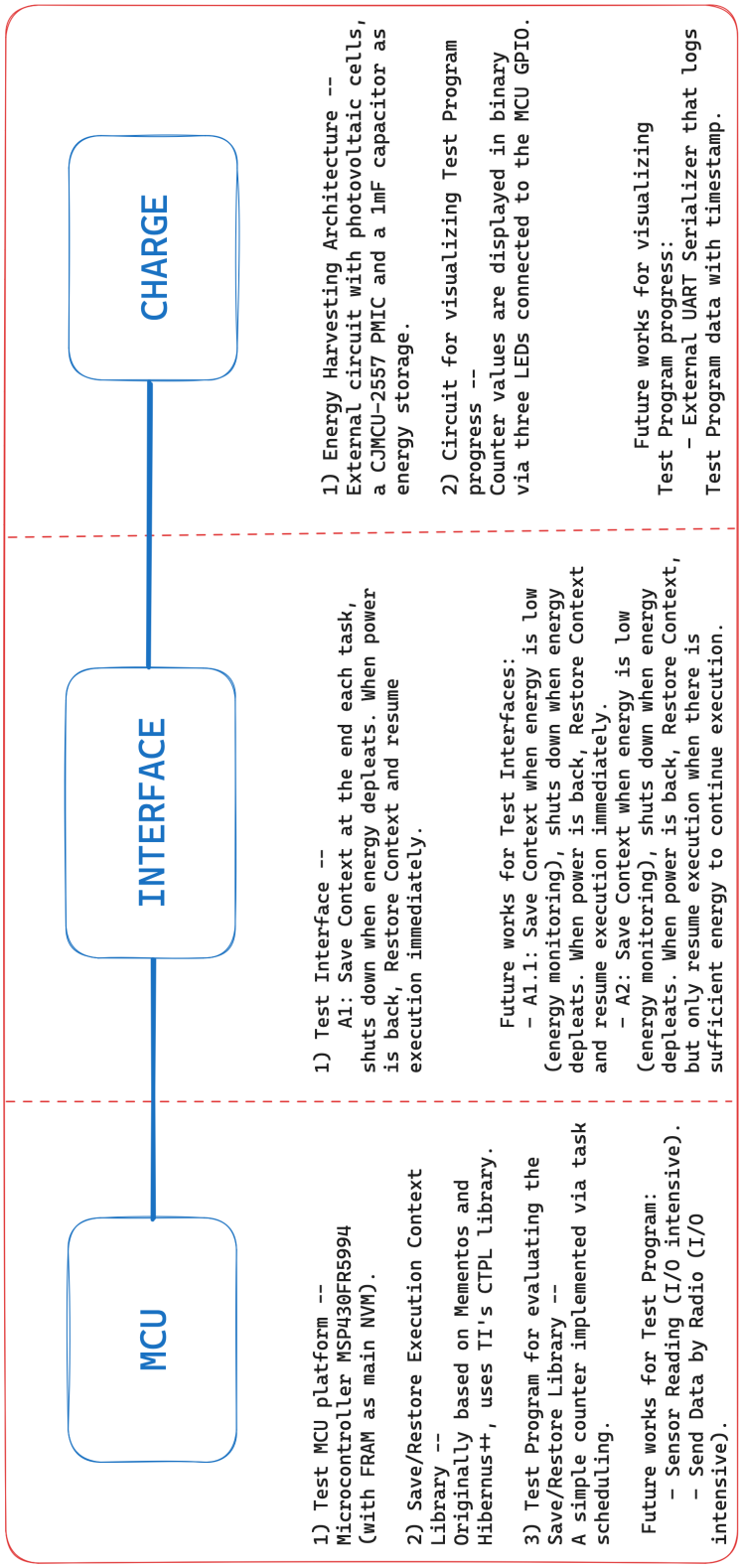

Figure 3.10: Final project's experimental model platforms.

4
Results

The first batch of tests were made without the influence of the EH architecture. Since a relevant part of the final project's methodology is implementing a solution for a program execution control, the performance of the designed software was evaluated with simulated power outages. This was done by simply unplugging the USB cable that connects the MSP-EXP430FR5994 board to the computer. Figure 4.1 shows the first test bench for the proposed system, with only the MCU and the LED circuit.

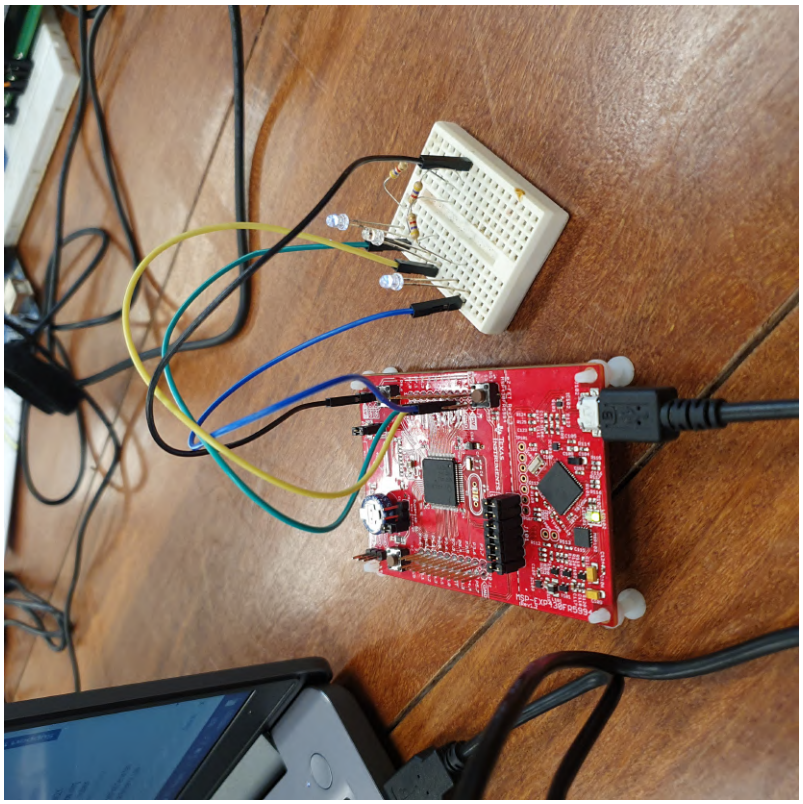

Figure 4.1: Test bench for the MCU coupled with the LED counter.

After the test program discussed in Chapter 3 was loaded into the MCU, it ran as smoothly as it was expected. The LEDs were powered in a combination that correctly represented the counter value, from zero (0) to seven (7) and then repeated this cycle on a loop. In order to test the Program Execution Control methodology implemented, a power failure was introduced by unplugging the USB cable in two moments: (1) in between tasks i.e. in between incremented values of the counter, (2) in the middle of the program counter execution, just as the LEDs were changing state. For the first test case, after the MCU was powered up, the test program execution resumed exactly from where it left off

between tasks. If the last state was the counter equal to three, when power was restored, execution continued where it left off, with the LEDS displaying four, and so on. On the second test case, however, as execution was shut off before the task could be completed, when the power was restored, the program correctly re-executed the task that was interrupted. If the interrupted task was in the middle of couting up to four, then the MCU began execution with the LEDs displaying three and then incremented to four. This behaviour was exactly what was expected, as it reflects the task-based strategy proposed in the the methodology.

An additional test was made that followed the exact strategy as the previous one, but instead of being powered by the computer via USB cable, an attempt was made to test the on-board supercapacitor featured on the MSP-EXP430FR5994 as the power supply. The supercapacitor was then charged accordingly the datasheet instructions, and a jumper was used to select its *Use* operation, before the USB cable was disconnected. Almost immediately, the LEDs dimmed and their brightness was not as strong as it was when connected to the computer, which is expected. Then, the previously described tests were repeated, but when a power failure was staged by disconnecting the jumper, the results were not as satisfactory. For the first test case, the control flow of the program was restored successfully, but for the second case, it was noticed that the program execution reset the counter entirely. Although no detailed explanation could be found, a possible explanation could be that the power fluctuation when connecting and disconnecting the jumper negatively interfered in the restore routine.

Subsequently, the next and most anticipated test case, was experimenting with EH as the energy supply source, and as such the MCU system was connected to the EH architecture, as illustrated in Figure 4.2. In this test case, the first problem arose when the circuit was connected, but the MCU did not power on. After checking the capacitor storage and verifying it was around 4V - more than enough to power the MCU - a quick re-inspection of the circuit was made. It was noticed that, apparently, the Vout output of the PMIC module was not supplying energy correctly to the microcontroller, despite the capacitor being more than sufficiently charged. An attempt was made to open the jumper connection of the on-board debugging probe on the EXP-MSP430FR5997, since it was theorized that the internal regulator present on that section of the developed board could add a heavy energy consumption load to the circuit. However, this attempt made no difference and the MCU still did not power on. Another more probable theory was that the EH circuit just could not generate enough power to feed the MCU. This could have been likely

influenced by the constant rainy weather during the week when the tests were implemented, on top of the fact it was already winter season when daylight is shorter.

Therefore, a workaround solution was found to try and at least evaluate the test program within the scope of the EH architecture. Instead of supplying the MCU with the output voltage from the power regulator, two jumpers were used to connect the capacitor in parallel with the MCU and a resistor was added to shield the MCU from the higher voltage of the capacitor. When the MCU was connected, it immediately turned on and the engraved firmware with the test program began to execute correctly. A power shortage was triggered by disconnecting the Vcc jumper that supplies the MCU, and when reconnected, the program state was restored and execution continued correctly, without resetting the counter.

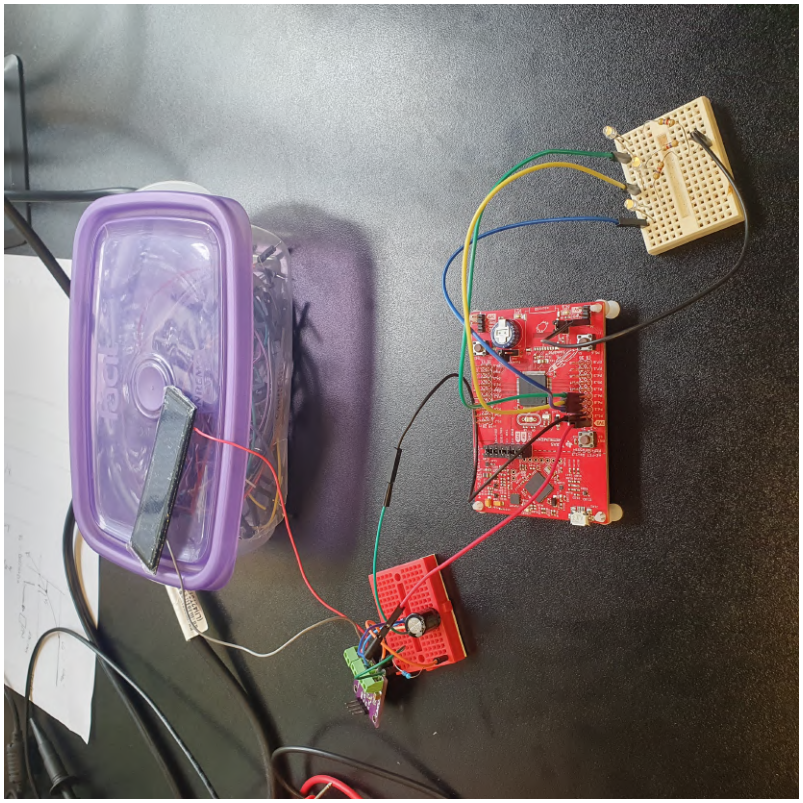

Figure 4.2: Test bench for the entire control interface (MCU and EH).

5
Conclusion and Future Works

In conclusion, the final project managed to present a suitable solution for intermittent operation in a microcontroller powered by EH. Despite the unexpected behaviour of the EH supply and the difficulties in developing software for program execution control, the results presented were satisfactory and represented a real chance at testing the proposed methodology.

One of the main project objectives was to suggest an outline for a control interface for computer systems powered by EH, in an attempt to help categorize the many related works that remain difficult to compare. Despite the plurality of EH applications making it harder to fit all these related works into a single category system, the hope is that this final project can help promote the idea of new control interface solutions. This would help in comparing their strategy and efficiency, in order to develop better and more clever solutions in the future.

Additionally, another goal that was hoped to be achieved was to be able to experiment practically with a real energy harvesting system. Even though the unfavourable environmental conditions hindered the effort of properly powering the MCU with the correct EH architecture, this was seen as an excellent learning opportunity. Related works had already mentioned the importance of choosing an appropriate and readily available energy source when deploying an EH system. It was expected that, since the city location where project experimentation took place was well-known for its sunny weather, that a photovoltaic energy harvester - capable of even harnessing energy from artificial light - would be a good choice. However, this was not the case, and the unpredictable ambient conditions significantly restrained the performance of the EH system.

Furthermore, the designing of this project helped immensely in applying and refining a considerable amount of technical knowledge that was necessary to implement it. Concepts on low-power electronics were revised in order to assemble the different circuits of both the MCU and the EH. The attempt to develop a save-and-restore program context software from scratch was very challenging, as it dealt with a lot low-level programming and having to analyse the MCU's registers one by one. Although this process was quite taxing, it was an excellent educational opportunity, allowing for a very thorough understanding of how the MSP430FR5994 microcontroller functions. This will make any future developments using this embedded system considerably less

strenuous.

Finally, throughout the development of the final project, several points for future works were raised and are listed below.

- Developing and experimenting with more complicated interfaces, that use complex storage systems or more efficient methodology for calling the save-and-restore application. For example, using hardware to monitor the system energy levels and trigger the intermittent execution model.
- Testing with other EH devices that have more resource availability than the photovoltaic cells, or that are at least capable of drawing more power than the ones used.
- Improving on top of *Texas Instrument's* CTPL API, so that the save-and-restore functions do not freeze the program execution after saving the system state. This would make it harder to test alternatives for program execution control, since these would be restricted by the API's mandatory execution freeze.
- Develop more complex test programs, that use sensor readings and even data transmission through radio, for example. These would allow for studying how the state of peripheral devices is affected by an intermittent operation.
- Improve the visualization strategy used to externalize the program data and verify its correctness through periods of power intermittency. Instead of using LEDs, the MCU's own UART could be used with a serial module to create log files of more expressive data.

6
Bibliography

Afanasov, M., Bhatti, N. A., Campagna, D., Caslini, G., Centonze, F. M., Dolui, K., Maioli, A., Barone, E., Alizai, M. H., Siddiqui, J. H., and Mottola, L. (2020). Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 368–381, Virtual Event Japan. ACM.

AliExpress (2023). CJMCU-2557 bq25570 nano power energy harvester.

Bakar, A. and Hester, J. (2018). Making sense of intermittent energy harvesting. In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, pages 32–37, Shenzhen China. ACM.

Ballo, A., Bottaro, M., and Grasso, A. D. (2021). A Review of Power Management Integrated Circuits for Ultrasound-Based Energy Harvesting in Implantable Medical Devices. *Applied Sciences*, 11(6). Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

Balsamo, D., Weddell, A. S., Das, A., Arreola, A. R., Brunelli, D., Al-Hashimi, B. M., Merrett, G. V., and Benini, L. (2016). Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):1968–1980.

Balsamo, D., Weddell, A. S., Merrett, G. V., Al-Hashimi, B. M., Brunelli, D., and Benini, L. (2015). Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters*, 7(1):15–18.

Berthou, G., Marquet, K., Risset, T., and Salagnac, G. (2020). MPU-based incremental checkpointing for transiently-powered systems. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 89–96, Kranj, Slovenia. IEEE.

Bhatti, N. A. and Mottola, L. (2017). HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, IPSN '17, pages 209–220, Pittsburgh, PA, USA. Association for Computing Machinery.

Bi, S., Ho, C. K., and Zhang, R. (2015). Wireless powered communication: opportunities and challenges. *IEEE Communications Magazine*, 53(4):117–125. Conference Name: IEEE Communications Magazine.

Chatterjee, A., Lobato, C. N., Zhang, H., Bergne, A., Esposito, V., Yun, S., Insinga, A. R., Christensen, D. V., Imbaquingo, C., Bjørk, R., Ahmed, H., Ahmad, M., Ho, C. Y., Madsen, M., Chen, J., Norby, P., Chiabrera, F. M., Gunkel, F., Ouyang, Z., and Pryds, N. (2023). Powering internet-of-things from ambient energy: a review. *Journal of Physics: Energy*, 5(2):022001. Publisher: IOP Publishing.

Colin, A. and Lucia, B. (2016). Chain: Tasks and Channels for Reliable Intermittent Programs. In *2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pages 514–530. Association for Computing Machinery.

Colin, A., Ruppel, E., and Lucia, B. (2018). A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 767–781, New York, NY, USA. Association for Computing Machinery.

Dietrich, S. (2022). Solving the Biggest Challenge of IoT Devices: Power.

Elahi, H., Munir, K., Eugeni, M., Atek, S., and Gaudenzi, P. (2020). Energy Harvesting towards Self-Powered IoT Devices. *Energies*, 13(21):5528.

EnABLES (2022). Up to 78 million batteries will be discarded daily by 2025, researchers warn.

Fortune (2023). Internet of Things [IoT] Market Size, Share & Growth by 2030.

Hasan, M. (2022). State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally.

Hester, J., Sitanayah, L., and Sorber, J. (2015). Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 5–16, New York, NY, USA. Association for Computing Machinery.

Hoseinghorban, A., Bahrami, M. R., Ejlali, A., and Abam, M. A. (2021). CHANCE: Capacitor Charging Management Scheme in Energy Harvesting Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):419–429. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

Jayakumar, H., Lee, K., Lee, W. S., Raha, A., Kim, Y., and Raghunathan, V. (2014). Powering the internet of things. In *Proceedings of the 2014 international*

symposium on Low power electronics and design, pages 375–380, La Jolla California USA. ACM.

Jayakumar, H., Raha, A., Stevens, J. R., and Raghunathan, V. (2017). Energy-Aware Memory Mapping for Hybrid FRAM-SRAM MCUs in Intermittently-Powered IoT Devices. *ACM Transactions on Embedded Computing Systems*, 16(3):1–23.

Jousimaa, O. J., Xiong, Y., Niskanen, A. J., and Tuononen, A. J. (2016). Energy harvesting system for intelligent tyre sensors. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 578–583, Gotenburg, Sweden. IEEE.

Ku, M.-L., Li, W., Chen, Y., and Ray Liu, K. J. (2016). Advances in Energy Harvesting Communications: Past, Present, and Future Challenges. *IEEE Communications Surveys & Tutorials*, 18(2):1384–1412.

Kwak, J., Kim, H., and Cho, J. (2021). ICEr: An Intermittent Computing Environment Based on a Run-Time Module for Energy-Harvesting IoT Devices with NVRAM. *Electronics*, 10(8):879.

Lucia, B., Balaji, V., Colin, A., Maeng, K., and Ruppel, E. (2017). Intermittent Computing: Challenges and Opportunities. In Lerner, B. S., Bodík, R., and Krishnamurthi, S., editors, *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, volume 71 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:14, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Maeng, K., Colin, A., and Lucia, B. (2017). Alpaca: intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–30.

Magno, M., Spadaro, L., Singh, J., and Benini, L. (2016). Kinetic energy harvesting: Toward autonomous wearable sensing for Internet of Things. In *2016 International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM)*, pages 248–254, Capri, Italy. IEEE.

Maiwa, H., Ishizone, Y., and Sakamoto, W. (2012). Thermal and vibrational energy harvesting using PZT- and BT-based ceramics. In *Proceedings of ISAF-ECAPD-PFM 2012*, pages 1–4, Aveiro, Portugal. IEEE.

Majid, A. Y., Donne, C. D., Maeng, K., Colin, A., Yildirim, K. S., Lucia, B., and Pawełczak, P. (2020). Dynamic Task-based Intermittent Execution for Energy-harvesting Devices. *ACM Transactions on Sensor Networks*, 16(1):1–24.

Partida, D. (2021). What Kinds of Batteries Are Best for IoT Devices?

Pinuela, M., Mitcheson, P. D., and Lucyszyn, S. (2013). Ambient RF Energy Harvesting in Urban and Semi-Urban Environments. *IEEE Transactions on Microwave Theory and Techniques*, 61(7):2715–2726.

Precedence (2022). IoT Battery Market Size To Hit Around USD 22.7 Bn by 2030.

Ransford, B., Sorber, J., and Fu, K. (2011). Mementos: System Support for Long-Running Computation on RFID-Scale Devices. *ACM SIGARCH Computer Architecture News*, 39(1):159–170.

Ribeiro, J. G. R., Chagas, N. S., and dos Santos, M. F. (2022). *O Impacto causado ao meio ambiente pelo descarte incorreto de pilhas e baterias*. Graduação e especialização, Faculdade Una Pouso Alegre - Minas Gerais.

Ruppel, E., Surbatovich, M., Desai, H., Maeng, K., and Lucia, B. (2022). An Architectural Charge Management Interface for Energy-Harvesting Systems. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 318–335, Chicago, IL, USA. IEEE.

Ryu, H., Yoon, H.-J., and Kim, S.-W. (2019). Hybrid Energy Harvesters: Toward Sustainable Energy Harvesting. *Advanced Materials*, 31(34).

Simjee, F. and Chou, P. (2006). Everlast: Long-life, Supercapacitor-operated Wireless Sensor Node. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, volume 2006 of *ISLPED'06*, pages 197–202, Tegernsee, Germany. Association for Computing Machinery.

Sinha, S. (2023). State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally.

Statista (2020). End of life recycling rates of battery metals worldwide, by type.

Talla, V., Pellerano, S., Xu, H., Ravi, A., and Palaskas, Y. (2015). Wi-Fi RF energy harvesting for battery-free wearable radio platforms. In *2015 IEEE International Conference on RFID (RFID)*, pages 47–54, San Diego, CA, USA. IEEE.

Texas Instruments (2023a). BQ25570 data sheet, product information and support.

Texas Instruments (2023b). CCSTUDIO code composer studio integrated development environment (ide).

Texas Instruments (2023c). MSP-EXP430FR5994 msp430fr5994 launchpad development kit.

Texas Instruments (2023d). Msp mcu fram utilities version 03.10.00.10 user's guide.

Texas Instruments (2023e). Msp430 driverlib for msp430fr5xx 6xx devices user's guide.

Texas Instruments (2023f). Msp430 family linker description.

Vatamanu, J. and Bedrov, D. (2015). Capacitive Energy Storage: Current and Future Challenges. *The Journal of Physical Chemistry Letters*, 6(18):3594–3609. Publisher: American Chemical Society.

Xie, L., Song, W., Ge, J., Tang, B., Zhang, X., Wu, T., and Ge, Z. (2021). Recent progress of organic photovoltaics for indoor energy harvesting. *Nano Energy*, 82.

Yang, F., Thangarajan, A. S., Michiels, S., Joosen, W., and Hughes, D. (2021). Morphy: Software Defined Charge Storage for the IoT. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, pages 248–260, New York, NY, USA. Association for Computing Machinery.

Zeadally, S., Shaikh, F. K., Talpur, A., and Sheng, Q. Z. (2020). Design architectures for energy harvesting in the Internet of Things. *Renewable and Sustainable Energy Reviews*, 128.