Circuit Support for Practical and Performant Batteryless Systems

Harrison R. Williams

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Matthew Hicks, Chair
Xun Jian
Angelos Stavrou
Ali Butt
Changhee Jung

April 25, 2024
Blacksburg, Virginia

Keywords: Embedded, Sensing, Batteryless, Circuits, Architecture

Circuit Support for Practical and Performant Batteryless Systems

Harrison R. Williams

(ABSTRACT)

Tiny, ultra-low-power embedded processors enable sophisticated computing deployments in a myriad of areas previously off limits to computing power, ranging from intelligent medical implants to massive scale 'smart dust'-type sensing deployments. While today's computing and sensing hardware is well-suited for these next generation deployments, the batteries powering them are not: the size and weight of today's mobile and Internet-of-Things devices are dominated by their batteries, which also limit systems' lifespans and potential for deployment in sensitive contexts. Academic efforts have demonstrated the feasibility of harvesting energy on-demand from the environment as a practical alternative to classical battery power, instead buffering harvested energy in a capacitor to power intermittent bursts of operation. Energy harvesting circuits are miniaturizable, inexpensive, and enable effectively indefinite operation when compared to batteries—but introduce new problems stemming from the lack of a reliable power source. Unfortunately, these problems have so far confined batteryless systems to small-scale research deployments.

The central design challenge for effective batteryless operation is efficiently using scarce input power from the energy harvesting frontend. Despite advances in both harvester and processor efficiency, digital systems often consume orders of magnitude more power than can be supplied by harvesting circuits—forcing systems to operate in short bursts punctuated by power failure and a long recharge period. Today's batteryless systems pay a steep price to sustain operation across these common-case power losses: current platforms depend on high-performance non-volatile memory to quickly and efficiently checkpoint program state before power loss, limiting batteryless operation to a small selection of devices which inte-

grate these novel memory technologies. Choosing exactly when to checkpoint to non-volatile memory represents a challenge in itself: the hardware required to *detect* impending power failure often represents a large proportion of the system's overall energy consumption, forcing designers to choose between the energy overhead of voltage monitoring or the runtime overhead of 'energy-oblivious' checkpointing models. Finally, the choice of buffer capacitor *size* has a large impact on overall energy efficiency—but the optimal choice depends on runtime energy dynamics which are difficult to predict at design time, leaving designers to make at best educated guesses about future environmental conditions. This work approaches energy harvesting system design from a circuits perspective, answering the following research questions towards practical and performant batteryless operation:

1. Can the emergent properties of today's low-power systems be used to enable efficient intermittent operation on new classes of devices?
2. What compromises can we make in voltage monitor design to minimize power consumption while maintaining *just enough* functionality for batteryless operation?
3. How can we buffer harvested energy in a way that maximizes energy efficiency despite unpredictable system-level power dynamics?

This work answers the following questions by producing the following research artifacts:

1. The first *non-volatile memory invariant* system to enable intermittent operation on embedded devices lacking high-performance memory (Chapter 2).
2. The first voltage monitoring circuit *designed for batteryless systems* to enable energy-aware operation without sacrificing efficiency (Chapter 3).
3. The first highly efficient *power-adaptive energy buffer* to store harvested energy without compromising on efficiency or performance (Chapter 4).

Circuit Support for Practical and Performant Batteryless Systems

Harrison R. Williams

(GENERAL AUDIENCE ABSTRACT)

Tiny, ultra-low-power embedded processors enable sophisticated computing deployments in a myriad of areas previously off limits to computing power, ranging from intelligent medical implants to massive scale 'smart dust'-type sensing deployments. While today's computing and sensing hardware is well-suited for these next generation deployments, the batteries powering them are not: the size and weight of today's mobile and Internet-of-Things devices are dominated by their batteries, which also limit systems' lifespans and potential for deployment in sensitive contexts. Academic efforts have demonstrated the feasibility of harvesting energy on-demand from the environment as a practical alternative to classical battery power, instead buffering harvested energy in a short-term energy store (i.e., a capacitor) to power intermittent bursts of operation. Energy harvesting circuits are miniaturizable, inexpensive, and enable effectively indefinite operation when compared to batteries—but introduce new problems stemming from the lack of a reliable power source. Unfortunately, these problems have so far confined batteryless systems to small-scale research deployments.

The central design challenge for effective batteryless operation is efficiently using scarce input power from the energy harvesting frontend. Despite advances in both harvester and processor efficiency, digital systems often consume orders of magnitude more power than can be supplied by harvesting circuits—forcing systems to operate in short bursts punctuated by power failure and a long recharge period. Today's batteryless systems pay a steep price to sustain operation across these common-case power losses: current platforms depend on high-performance non-volatile memory (which retains state without power) to quickly and efficiently checkpoint program state before power loss, limiting batteryless operation to a small

selection of devices which integrate these novel memory technologies. Choosing exactly when to checkpoint to non-volatile memory represents a challenge in itself: the hardware required to *detect* impending power failure often represents a large proportion of the system's overall energy consumption, forcing designers to choose between the energy overhead of voltage monitoring or the runtime overhead of 'energy-oblivious' checkpointing models. Finally, the choice of energy buffer *size* has a large impact on overall energy efficiency—but the optimal choice depends on runtime energy dynamics which are difficult to predict at design time, leaving designers to make at best educated guesses about future environmental conditions. This work approaches energy harvesting system design from a circuits perspective, answering the following research questions towards practical and performant batteryless operation:

1. Can the emergent properties of today's low-power systems be used to enable efficient intermittent operation on new classes of devices?
2. What compromises can we make in voltage monitor design to minimize power consumption while maintaining *just enough* functionality for batteryless operation?
3. How can we buffer harvested energy in a way that maximizes energy efficiency despite unpredictable system-level power dynamics?

This work answers the following questions by producing the following research artifacts:

1. The first *non-volatile memory invariant* system to enable intermittent operation on embedded devices lacking high-performance memory (Chapter 2).
2. The first energy monitoring circuit *designed for batteryless systems* to enable energy-aware operation without sacrificing efficiency (Chapter 3).
3. The first highly efficient *power-adaptive energy buffer* to store harvested energy without compromising on efficiency or performance (Chapter 4).

Dedication

To my dog Tiger for unconditional love.

Acknowledgments

I owe a great deal of gratitude to the many people who inspired, supported, and motivated me over the course of my time at Virginia Tech. I am forever grateful for my parents Julie Lang and Rick Williams for their patience, sacrifice, and support, without which I never would have had the motivation or opportunity to pursue graduate school.

I want to thank to my friends and labmates in and beyond Blacksburg; Dr. Michael Moukarzel, Dr. Stefan Nagy, Jubayer Mahmod, Prakhar Sah, Nandita Singh, Zeezoo Ryu, Daniel Chiba, and Cameron Garcia all deserve special thanks for their part in my time here. I'll look back fondly on the hours we spent celebrating successes, commiserating on the (many) trials of research, and discussing anything ranging from job hunts to relationships. Thank you too to all of the members of the FoRTE Research group I overlapped with during my time here for making the lab a lively and exciting place to work.

I owe the most thanks to my advisor Dr. Matthew Hicks. I joined Matt's lab as one of his first students in September of 2017, as an undergraduate with no interest in an academic career; it is safe to say he dramatically changed my trajectory for the better by convincing me to stay for a PhD. Matt has been a role model for imagining, building, and evaluating systems, running a lab, and leading students, and I am confident the lessons I have learned in his lab will serve me well into the future. It has been a true privilege to work with and learn from him over these last seven years.

I also thank my committee Xun Jian, Angelos Stavrou, Ali Butt, and Changhee Jung for their guidance and support.

Publications from this Thesis

This dissertation is derived from the following coauthored publications:

1. **Harrison Williams**, Xun Jian, and Matthew Hicks. *Forget Failure: Exploiting SRAM Data Remanence for Low-overhead Intermittent Computation.* In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and operating Systems, (**ASPLOS '20**). March 2020.

2. **Harrison Williams**, Michael Moukarzel, and Matthew Hicks. *Failure Sentinels: Ubiquitous Just-in-time Intermittent Computation via hardware support for continuous, low-cost, fine-grain voltage monitoring.* In Proceedings of the 2021 International Symposium on Computer Architecture (**ISCA '21**). June 2021.

3. **Harrison Williams** and Matthew Hicks. *Energy-Adaptive Buffering for Efficient, Responsive, and Persistent Batteryless Systems.* In Proceedings of the Twenty-Ninth International Conference on Architectural Support for Programming Languages and operating Systems, (**ASPLOS '24**). April 2024.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Decades of sustained transistor downscaling, alongside improvements in processor architecture and low-power system design, have produced tiny and inexpensive yet feature-rich computing systems that operate on only micro-watts of power. These highly efficient systems form the backbone of the Internet-of-Things (IoT)—bringing connectivity and intelligence to areas previously off limits to computing power in domains ranging from sophisticated in-body healthcare [105, 138, 139] and space deployments [30] to massive-scale infrastructure or environmental sensing networks [2, 56, 123]. As sensing, computation, and communication continue to scale down to smart dust [83] dimensions, system designers face both new challenges and opportunities as a result of this extreme miniaturization.

One such challenge is the growing disparity between processor performance and battery energy density. While Moore's Law has driven exponential growth in on-chip processing power, no such analog exists for batteries, which have historically scaled at a modest linear rate [125]. As a result, batteries are now the limiting factor in further miniaturizing today's mobile systems: meeting typical operational lifespans (e.g., 10 years [12]) requires integrating a battery that is often orders of magnitude larger than the processor it powers, preventing practical deployment in size/weight-constrained scenarios as well as in sensitive contexts where the volatility of a battery is unacceptable. As long as mobile processors remain tethered to batteries, truly ubiquitous computing will remain out of reach.

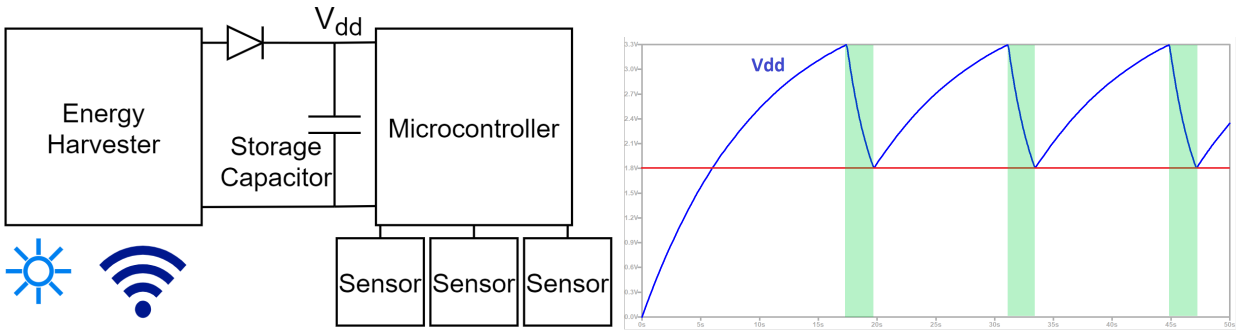
Energy harvesting systems, which operate exclusively using energy scavenged from the en-

vironment in which they are deployed, offer a solution. Environmental energy is abundant and available in many forms: research has demonstrated practical energy harvesters based on both mature (e.g., solar [40], RF [14]) and novel technologies (e.g., vibration [127], chemical [105], heat gradients [122]) suitable for a variety of deployments. Energy harvesters solve the primary problems associated with battery-powered operation: many harvesting circuits can be miniaturized down closer to the scale of the processor [138], while the renewable nature of environmental energy enables effectively indefinite operation.

Despite the system-level potential of batteryless energy harvesting systems, foregoing batteries introduces a myriad of lower-level performance and practicality barriers that prevent widespread deployment of batteryless systems beyond the lab. Extreme energy scarcity—most harvesters output a small trickle of energy relative to the power consumption of even a small mobile system—forces batteryless systems to operate at very low duty cycles, in short bursts of operation punctuated by long recharge times. The highly volatile nature of environmental energy means that the timing and duration of these short bursts of operation are typically impossible to predict, leaving developers with little information to reason about the behavior of their system. Building sophisticated systems on batteryless platforms requires first that designers find ways to extend program execution *across* these unpredictable yet frequent power cycles, an area of study researchers have termed *intermittent computation*.

1.1 An Overview of Intermittent Computation

Figure 1.1 illustrates the basic hardware design and operating behavior of a computational energy harvesting system. The harvester stores energy in an intermediate buffer capacitor until the capacitor voltage reaches some predefined enable level, at which point the microcontroller and associated peripherals begin operation. The system drains energy from the



(a) Energy harvesting block diagram

(b) Idealized supply voltage for energy harvester microcontroller. Red line indicates the minimum voltage for code execution.

Figure 1.1: Energy harvesting device block diagram and idealized power supply (V_{dd}). In a real system, the charge/discharge rate depends on the harvester's yield, the microcontroller's current draw, and the capacitor size. The green highlighted regions indicate when the microcontroller computes.

---

storage capacitor to do work until the capacitor voltage reaches some disable point (often the brown-out voltage of the microcontroller), and the charge/discharge cycle repeats. Because system size, cost, and charge time constraints limit the size of the storage capacitor, the length of a continuous operating period is often on the scale of hundreds of milliseconds. Because program progress and intermediate state is normally stored in volatile memory—which loses its contents upon power loss—unmodified code will never complete under these circumstances, losing all progress and beginning anew on each power cycle.

There are two basic requirements to support intermittent computation following a power loss: (1) recover the program's pre-power-cycle state and (2) ensure the recovered program state is consistent. Figure 1.2 gives a high-level overview of the different approaches explored in the literature. Each approach has its own set of tradeoffs, giving designers a wide range of options when building an intermittent system.

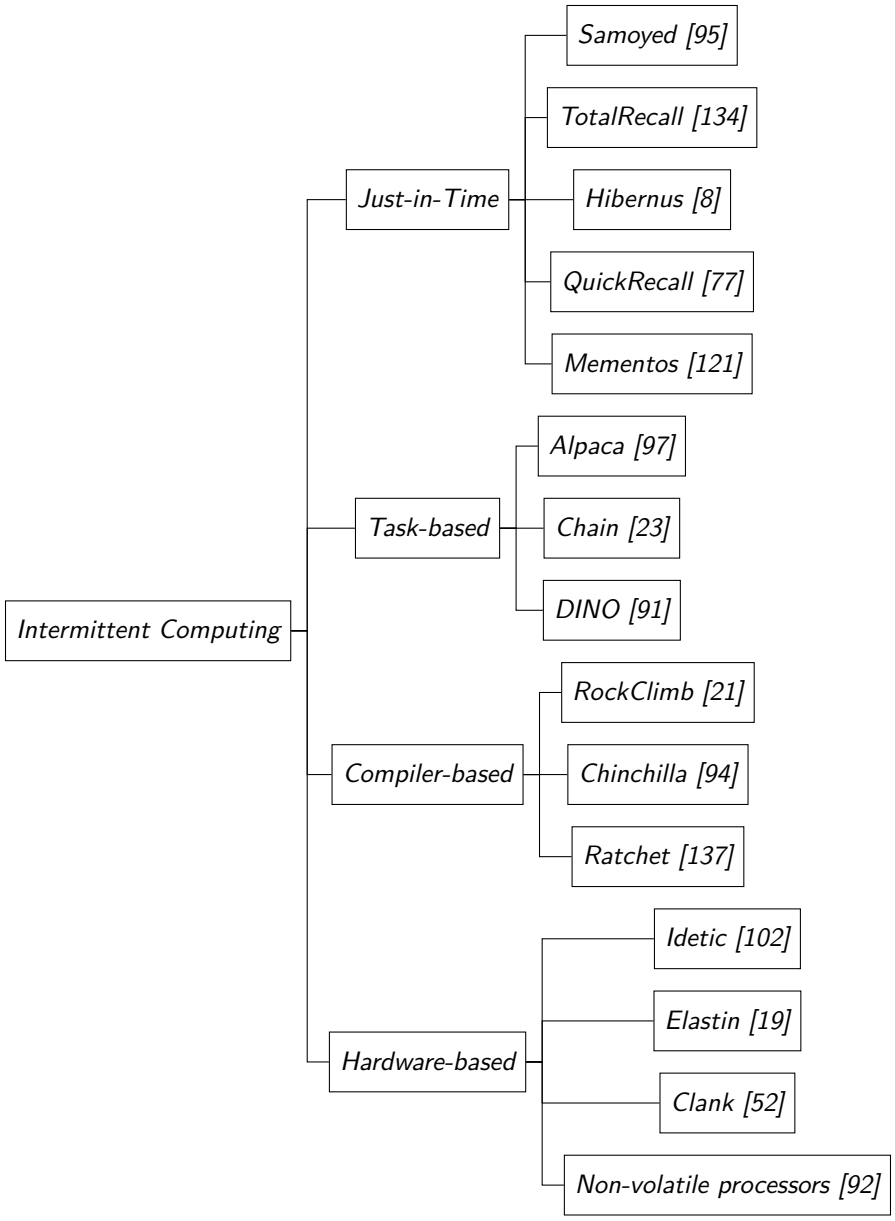


Figure 1.2: A taxonomy of checkpointing techniques for intermittent computation.

Just-In-Time (JIT) Checkpointing: JIT checkpointing systems use a combined hardware/software approach to save and restore state across power cycles. These systems use analog voltage measurement circuits to monitor the remaining energy in the supply capacitor and configure an interrupt to trigger when capacitor voltage falls, indicating power loss is imminent. This interrupt commits all volatile program state (i.e., program data stored in volatile memory and architectural registers) to Non-Volatile Memory (NVM) to be restored on the following power-up. JIT checkpointing schemes minimize both software overhead and programming effort—only strictly-necessary checkpoints are taken, and the software component may be integrated at link-time into unmodified application code—but incur additional hardware and energy overhead in the form of quiescent current for the voltage monitor. One notable limitation of JIT approaches is that execution must stop between recording and recovering from a checkpoint, as re-executing uncheckpointed work can violate program semantics when the re-executed work is *non-idempotent* [120]. While the energy overhead of voltage monitoring systems can be substantial, JIT approaches using low-power voltage monitors typically outperform other checkpointing approaches on pure software benchmarks [95, 135].

Task-based Checkpointing: JIT checkpointing systems face problems when software contains atomic operations that cannot be split across power cycles (e.g., interactions with peripherals), as checkpoints partway through an atomic operation will cause a silent failure when software assumes a failed atomic operation actually completed. Task-based systems address this problem: instead of automatically checkpointing in a 'program-oblivious' manner as in a JIT system, programmers logically separate code into a series of 'tasks' rendered repeatable and atomic by the checkpointing system. Checkpoints are only recorded between tasks—ensuring no atomic operation is ever split—and state is transparently "rolled back" when a task is interrupted by power failure, completely re-executing the task until the system

can reach the next checkpoint. Task-based systems ensure correctness and do not require hardware support in the form of a voltage monitor, but require the programmer to refactor their code and reason about power conditions when deciding on the length of tasks. Task-based systems incur more software overhead than JIT systems because their programming models require checkpoints *regardless* of power conditions, introducing unnecessary checkpoints which are never needed to recover. More recent work combining efficient, energy-aware JIT checkpointing with the rollback mechanisms developed for task-based models blends the performance of JIT approaches with the correctness of task-based ones [95].

Compiler-based Checkpointing: Task-based systems ensure correctness at the cost of increased programmer effort. Compiler-based checkpointing techniques offer an alternative, identifying through static analysis the non-idempotent sections that threaten program correctness and automatically inserting checkpoints of the minimal state required to render them idempotent. Such systems minimize design effort and hardware overhead by eliminating the need for voltage monitors or task-based programming models at the cost of increased software overhead continuously checkpointing state throughout the program.

Hardware-based Checkpointing: While the above techniques are suitable for commercially available low-power systems, enabling intermittent operation by modifying the *hardware* is another viable approach. Hardware-based approaches range from intermittency-aware memory management units [52] to fully-non-volatile processors [92] which eliminate the need for any software mitigation of power failures. While such techniques offer a promising avenue for simple and efficient intermittent computation, the need for custom hardware remains a major barrier in their widespread adoption.

1.2 Circuit- and Device-level Challenges

While extending program execution across power cycles is a central component of effective batteryless systems, considerations beyond software execution contribute to the functional gap between batteryless systems and their battery-powered counterparts. One major obstacle is current checkpointing systems' dependence on high-performance non-volatile memories such as Ferroelectric RAM (FRAM) and Magnetoresistive RAM (MRAM); while these memories offer non-volatility with performance characteristics comparable to typical volatile RAM (i.e., Static RAM, or SRAM), they are integrated into only a tiny fraction of currently available systems. The most widespread NVM—Flash—is unsuitable for the high write rates of existing checkpointing systems, leaving batteryless operation beyond the reach of the vast majority of existing systems. Worse, current iterations of these next-generation non-volatile RAMs underperform typical Flash and SRAM in terms of access frequency and energy—forcing intermittent systems to pay a steep common-case performance price to enable current checkpointing strategies.

Other challenges arise as a result of the basic architecture of batteryless systems. Typical batteryless systems buffer energy in a single bulk capacitor, the size of which has a profound impact on the behavior of the system. Capacitor size exists on a trade space: large capacitors buffer more energy, enabling longer continuous bursts of operation which support longer atomic operations and reduce the overhead incurred recovering from power loss. These large capacitors, however, force the system to buffer more energy to begin operation *at all*—reducing a system's ability to quickly turn on and react to incoming events even if those events require relatively little energy to address. Small capacitors face the opposite problem: the system charges quickly, but cannot buffer enough energy to reliably support long-running operations (and in the worst case may even need to burn off energy as heat to

prevent overvoltage if they reach capacity). This responsiveness-capacity tradeoff prevents current batteryless systems from reliably and flexibly addressing different kinds of events using a single hardware platform.

1.3 Contributions of This Work

My work approaches batteryless operation from a circuits perspective to overcome applicability and performance barriers holding back today's energy harvesting systems. Specifically, I introduce the research artifacts detailed below:

1. *TotalRecall* (Chapter 2): A Just-In-Time checkpointing library that exploits SRAM's *time-dependent non-volatility* to enable practical and performant *NVM-agnostic* intermittent computation, bringing batteryless operation to the cheapest, most widespread, and most performant low-power systems available today.

2. *Failure Sentinels* (Chapter 3): An *ultra-low-power, all-digital voltage monitor* optimized for batteryless systems designed to eliminate the hardware energy, size, and cost overheads associated with JIT checkpointing.

3. *REACT* (Chapter 4): An *energy-adaptive buffer circuit* designed to close the responsiveness-capacity trade space holding back batteryless systems by matching buffer capacitance to immediate power supply/demand dynamics.

Chapter 2

Exploiting SRAM Data Remanence for Intermittent Computation

2.1 Introduction

As modern microcontrollers become smaller and more energy efficient, system designers are finding novel applications for these devices in a variety of areas. The advent of wearable devices [139], RFID smart tags [147], and smart dust [83] represents potential for designers to leverage new low-power chips towards more ubiquitous computing and a greatly expanded Internet-of-Things (IoT). The limiting factor is not microcontrollers themselves, but the batteries powering such devices—today's battery-powered devices are not limited in size by the density of devices on silicon, but by energy demands dictating a battery that often makes up the bulk of the space and cost of the product. Both the economic and engineering feasibility of shrinking IoT and other ubiquitous computing devices depend on foregoing batteries.

A necessary component of a batteryless future is energy harvesting circuits, which draw power from the environment from sources such as RFID readers [84] or ambient vibration [127]. Many energy harvesters cannot output enough energy to continuously power microcontrollers; they trickle charge into a capacitor, providing enough power for brief computation, until the capacitor empties sufficiently, and the cycle repeats.

To enable reliable intermittent computation in the presence of frequent power failures, prior work proposes saving program state to Non-Volatile Memory (NVM) (e.g., Flash [121] and Ferrorelectric RAM (FRAM) [63]), preserving program state that is otherwise lost without power. They propose checkpointing volatile program state (e.g., registers and stack) to NVM, either before the next power failure [7, 8, 77, 121] or periodically at compiler-dictated points [23, 91, 94, 97, 137]. The most ubiquitous, highest performance, and lowest cost NVM is Flash. Unfortunately, while many aspects of Flash memory are great for energy harvesting systems, **Flash writes are particularly ill-suited for intermittent computation**. Flash writes are much slower and more energy intensive than writes to volatile memory [131]. Additionally, Flash's limited write endurance results in existing checkpointing schemes killing it in hours to a year of operation [70]. This is why recent work targets the more esoteric FRAM-based devices despite the many advantages of Flash-based devices [7, 8, 77, 94, 137].

I observe that Flash retains program state for years, but **intermittent computation only requires retention during short power-off times** (e.g., $<1s$). This work answers the question, **"Is there a way to avoid paying the price for retention guarantees that intermittent computation does *not* benefit from?"** I answer this question affirmatively by exploiting the time-dependent volatility of Static Random-Access Memory (SRAM). The driving observation is that when a microcontroller hits its brown-out voltage (e.g., $1.8V$) and ceases computation, significant charge remains in the system; this remaining charge leaks away slowly, resulting in a gradual transition from the brown-out voltage to $0V$. In a process known as data remanence, SRAM retains its state as long as voltage is above its retention voltage (e.g., $0.4V$) [118]. My experiments show that SRAM retains data for almost an hour after the microcontroller turns off (§2.3.2)—3000 times greater than common intermittent computation off times. Thus, **for the short off times common to intermittent computing, SRAM acts as a NVM—without Flash's write penalties**.

To demonstrate the ability of SRAM to serve as a low-overhead NVM for intermittent computation, I design and implement a library-level, lightweight, in-situ, one-time checkpointing approach called TotalRecall.[1] TotalRecall checkpoints are in-situ: SRAM data remains in place, while registers are checkpointed to SRAM. To handle worst-case off-times that expose SRAM's volatility, TotalRecall calculates a Cyclic Redundancy Check (CRC) over SRAM and adds it to the in-SRAM checkpoint. Upon recovery, TotalRecall verifies the CRC, falling back to an existing checkpointing approach in the event of a mismatch. I implement TotalRecall on both Flash- and FRAM-based microcontrollers common to intermittent computation. In experiments with benchmarks from Texas Instruments, **TotalRecall provides better-than-FRAM performance on Flash devices**—with over 99.999% worst-case reliability.[2] TotalRecall improves on the performance of state-of-the-art Flash-based one-time checkpointing between 230% and 37000%.

This work makes four technical contributions:

- I show that SRAM data remanence can safely store program state for stretching computation across short power cycles; previously, it had been shown to be a way to exfiltrate secret data [41, 45] and keep track of time [108, 118].
- I design TotalRecall, a library-level checkpointing technique that exploits SRAM data remanence. TotalRecall is NVM agnostic and supports existing software without modification (beyond linking against my library).
- I implement TotalRecall on both Flash- and FRAM-based MSP430 microcontrollers. My evaluation using benchmarks from Texas Instruments shows correct operation even with up to 5-minute off times. TotalRecall boosts performance of Flash-based devices

[1] TotalRecall stems from SRAM's ability to remember it's power-on state perfectly given short off times.
[2] My experiments suggest that Flash-based checkpointing will cause Flash writes to fail (i.e., device failure) before a silent data corruption occurs with TotalRecall.

up to and beyond FRAM-based systems.

- I explore the performance/reliability trade space that TOTALRECALL exposes to designers for hardware and software CRC implementations as well as 16- and 32-bit CRC implementations. My results show that hardware support for CRC maximizes TOTALRECALL's performance and reliability.

2.2 Background

There are several classes of checkpointing approaches in intermittent systems (§2.7), each ensuring consistent recovery in a different way. Because the more checkpoint heavy continuous checkpointing approaches are ill-suited for Flash devices due to its limitations (§2.2.2), I focus on one-time checkpointing approaches. One-time checkpointing approaches [7, 8, 77, 121] assume some voltage monitoring capability to detect when the energy storage capacitor has *just* enough energy to write a checkpoint before the microcontroller turns off. The microcontroller then stops computation to maintain recovery consistency. *This work focuses on reducing the overhead of one-time checkpointing as used in intermittent computation.*

2.2.1 Microcontroller Memory Hierarchy

The microcontrollers deployed in energy harvesting devices [107, 123, 147] are scalar in-order processors. They employ a flattened memory hierarchy[3] consisting of two types of memory:

- **Static Random-Access Memory (SRAM)**: main memory: holds transient program data (e.g., heap and stack). Positives include fast reads and writes, low energy,

[3]FRAM devices support a hierarchical memory model where SRAM acts as a cache for FRAM [68]. This reduces volatile state to architected state. This organization decreases checkpointing overhead, but sacrifices common-case performance [78]. Thus, the default is a Flash-like, flattened, model.

	Flash [65]	**FRAM** [66]	**SRAM**
NVM size	256 KB	256 KB	
Cost	$6.23	$6.33	
SRAM	16 KB	8 KB	
Max freq.	25 MHz	16 MHz	
Dhrystone	8.59 ms	13.52 ms	
Released	2001	2013	
Read freq.	25 MHz	8 MHz	$>$ 25 MHz
Write freq.	.036 MHz	8 MHz	$>$ 25 MHz
Endurance	10^4	10^{15}	∞
Min V.	2.2 V	1.8 V	$<$ .9 V [59]
Byte erase/program	No	Yes	Yes

Table 2.1: Comparison of Flash and FRAM microcontrollers from Texas Instruments, targeting the same NVM size and price point.

and byte addressable. Negatives include small size (e.g., 512B) and volatility, i.e., it *eventually* loses state without power.

- **Non-Volatile Memory (NVM)**: permanent data store: holds stable program data (e.g., code and constants). The positive is its large size (e.g., 256KB). Negatives include being slower and higher energy than SRAM. Note that different NVMs exacerbate or diminish these trade offs.

2.2.2 NVM Choices

Given that checkpointing is a fundamental part of intermittent computation, NVM selection greatly impacts intermittent computation overhead. Commodity microcontrollers offer two choices of NVM: either Flash or FRAM. To compare and contrast Flash and FRAM with respect to energy harvesting, I select representative microcontrollers from a popular online electronics distributor. I select Texas Instruments MSP430 microcontrollers to facilitate cross-device performance comparison and because that is the microcontroller used by deployed energy harvesting platforms [107, 123, 147]. Since NVM size is the primary driver of cost and technology, I fix NVM size at 256KB. From the remaining microcontrollers, I

select the wider voltage range (1.8–3.6V) as that enables longer on-times. From there, I limit the results to in-stock parts available in small quantities. For each NVM type, I select the cheapest device as the representative microcontroller. Table 2.1 highlights the tradeoffs between the representative microcontrollers, along with a comparison to SRAM.

Flash benefits: The green cells in Table 2.1 highlight where Flash outperforms FRAM. The most important benefit of Flash is its ubiquity: Flash-based MSP430s have a decade lead over FRAM-based MSP430s. Flash devices continue to outsell FRAM devices today as only one vendor sells FRAM-based devices, while dozens sell Flash-based variants. For example, the online distributor lists over 64,000 Flash-based microcontrollers, while only 777 options exist for FRAM variants. In addition to availability, Flash devices also have a performance advantage over FRAM-based devices. Specifically, Flash reads are 3x faster, provide 2x the SRAM, and a 62% higher clock frequency. These individual advantages add up to a 57% improvement in the Flash device's Dhrystone score over the FRAM device's. Thus, while one-time checkpointing approaches mesh well with FRAM, **ignoring high-performance solutions for Flash-based devices leaves most current and future deployed systems unserved**.

Flash drawbacks: Despite the availability and performance advantages of Flash, it has limitations that cause recent energy harvesting systems to move to FRAM-based devices [94, 107, 137, 147]. Flash's key disadvantage centers on writes. Programming Flash (i.e., writing) is a uni-directional, energy, and time-intensive process called hole punching. In hole punching, you increase the voltage of a Flash cell to force charge across a dielectric, changing the state of the cell from a 1 to a 0. This requires a higher starting voltage (e.g., 2.2V) and sufficient time—and energy—to pump the voltage up to the required level (e.g., 12V). Changing any cell's value back to a 1 requires an erase—which only occurs at segment gran-

ularity (e.g., 512B): updating a single word in Flash requires copying the entire segment to SRAM, erasing the segment in Flash, then writing the modified segment back to Flash. This makes Flash writes incredibly costly as Table 2.1 indicates that writing alone is 200x slower than FRAM.

Another write-related problem often overlooked in previous work is Flash's limited write endurance [70]. Each program/erase cycle ages the Flash cell, making it harder to program/erase in the future. This means that the frequent checkpoints required to support intermittent computation quickly kill Flash—taking the microcontroller with it. Thus, it is clear that without an alternative to checkpointing to Flash, the vast majority of microcontrollers will not support intermittent computation and the most advanced continuous checkpointing approaches are a non-starter. *The goal of this work is to achieve FRAM-like checkpointing performance and lifetime on the more ubiquitous and performant Flash-based devices.*[4]

2.3 Motivation

To enable low-overhead and long-term intermittent computation on Flash-based devices, I exploit SRAM's data remanence to allow it to serve as a NVM. This approach stems from two observations: (1) the off times of intermittent computation are short[5] and (2) SRAM on microcontrollers retains data for a relatively long time. Given these observations, I see an opportunity to tradeoff unnecessarily long data retention guarantees for a increase in

[4]As emerging NVM technologies mature and approach Flash-based devices in terms of frequency, read latency, features, and availability, the whole-memory non-volatility guarantee provided by emerging NVM technologies is preferable to the added complexity of the mixed-volatility offered by Flash+TotalRecall.

[5]While many energy harvesting devices will go long periods without power (e.g., smart cards), I note that these long off times demarcate individual, unrelated, computations; I differentiate these workloads from those targeted by intermittent computation. Previous work makes a similar observation about temporal locality of results and intermittent computation [37].

Energy Source	Source	Max Off Time (s)	SRAM Suitable $10\mu F$, $47\mu F$
RF	[49, 92, 121]	10	$<$ 80C, ✓
Thermal	[92]	14	$<$ 75C, ✓
Piezoelectric	[92, 127]	2	✓, ✓
Solar*	[49, 92]	300	$<$ 25C, $<$ 55C

Table 2.2: Off times in various energy harvesting systems along with the maximum ambient temperature that affords 100% SRAM data retention. *Solar-powered systems experience longer off times at night, but this is (predictable) power loss—not intermittent computation. ✓ represents that the temperature is above the device's maximum operating temperature of 85C.

performance and lifespan for Flash-based intermittent computing devices.

2.3.1 Intermittent Off Times are Short

The first observation driving my approach is that the off times common to intermittent computation are short. To validate this observation I explore the off times of energy harvesting platforms across a range of energy sources. This task is complicated by the fact that previous work focuses on on-times due to its reliance on the long-term data retention guarantees of NVMs and because of the on-time's impact on checkpointing overhead. Fortunately, in providing on-time data, they also provide enough data to approximate off times. Table 2.2 presents a summary of the off times from different energy sources. From this summary, I make two observations: (1) intermittent off times tend to be of the same magnitude as on times—i.e., short—and (2) many long off times are predictable and incongruent with the goal of intermittent computation due to temporal locality. Thus the data retention guarantee of traditional NVMs is overkill for intermittent computation.

2.3.2 SRAM has Time-dependent Volatility

The second observation driving my approach is that after a system ceases computation due to power failure, the remanent charge keeps the device's voltage higher than SRAM's Data Retention Voltage (DRV) for some time. This is because SRAM DRV, which is ~$0.4V$ [55, 116], is much lower than the operational voltage of the microcontroller (e.g., $1.6V$ for the representative MSP430s).[6] When supply voltage falls below the minimum operating voltage of the microcontroller, the power consumption drops drastically as transistors stop switching; power drain is now dominated by leakage current in the MCU and surrounding circuitry. While this leakage eventually reduces the supply voltage below the SRAM DRV, both extrinsic (i.e., decoupling capacitors and PCB trace capacitance) and intrinsic (e.g., transistor and trace capacitance) sources of capacitance prevent the supply voltage from dropping instantaneously. The energy storage capacitor described in §2.2 dominates the charge remanence effect; previous work on SRAM remanence supports this observation [118].

To empirically determine how long the remanent charge in microcontrollers allows SRAM to retain state after a power failure, I perform real-system experiments using a Flash-based MSP430. For this experiment, I first initialize all SRAM cells to a known value, then disconnect power for a set time, and finally, read back the SRAM data, checking for bit flips. Because SRAM fails bi-directionally—some cells tend to fail into a 0 state, while others fail into a 1 state—I run two trials at each off time, writing all 1's then all 0's [53]. I perform a binary search to determine the maximal off time where full SRAM state retention occurs.

Figure 2.1 shows the retention time of my MSP430 microcontroller across a range of temperatures and the two energy storage capacitor sizes common to deployed energy harvesting devices [107, 123]. I vary temperature (using a Test Equity 123H thermal chamber) and

[6] Past work on low-power embedded systems has explored using this effect to reduce leakage in sleep modes by clamping supply voltage down to SRAM's DRV [76], which immediately minimizes leakage at the cost of discharging the capacitance that enables SRAM's time-dependent non-volatility.

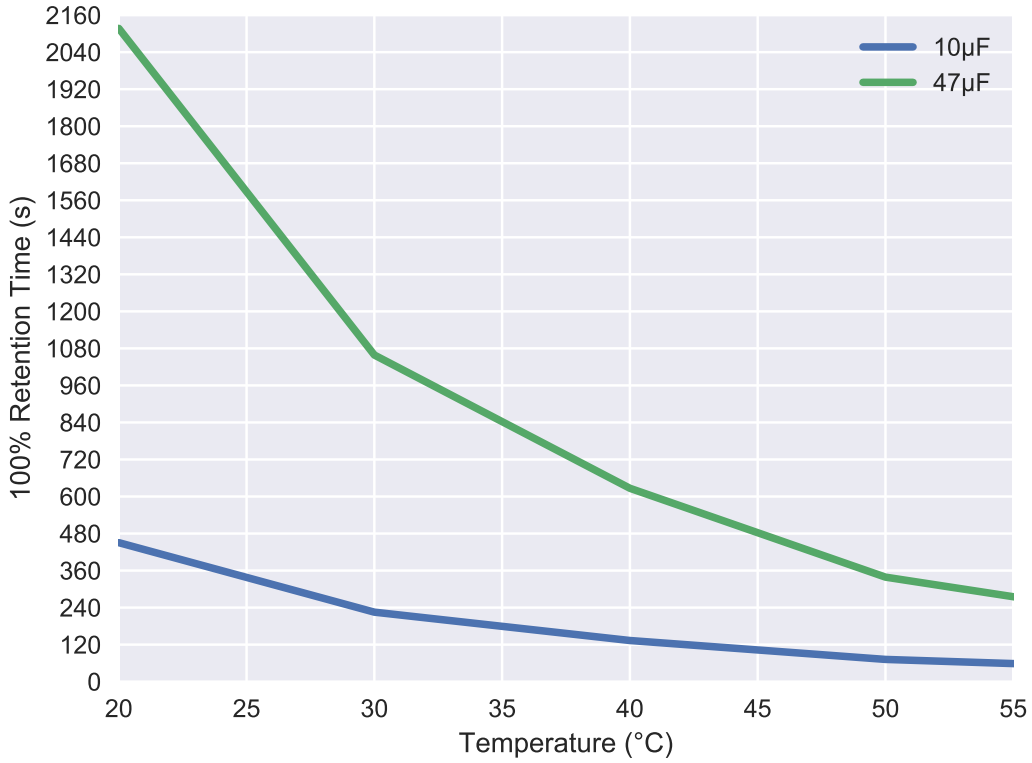


Figure 2.1: The time before the first SRAM cell in a microcontroller loses state varies with ambient temperature and capacitor size.

---

capacitance (via power rail decoupling capacitors) as those two variables dominate charge leakage and by extension SRAM data retention time. To contextualize these results, I add the maximum supported temperature for a given energy source and capacitor size combination in Table 2.2. In this work, **I design and implement a system that reliably uses SRAM as a low overhead, long lifetime, NVM for the short off times common to intermittent computing**, falling back to existing checkpointing to support longer, power off, scenarios.

2.4 TotalRecall Design

I design TotalRecall, an efficient checkpointing technique for intermittent computing on energy harvesting devices. TotalRecall eliminates costly NVM checkpoints by reliably retaining program state in SRAM. The pervasiveness of SRAM makes TotalRecall deployable regardless of NVM technology. The library-based design of TotalRecall makes it deployable to current and future commercial off-the-shelf microcontrollers, without software or hardware modification. When deployed on Flash microcontrollers, TotalRecall enables system designers to take advantage of the benefits of Flash (Table 2.1), without its checkpointing-induced high write/erase overheads and limited lifetime. Much of TotalRecall follows from previous one-time checkpointing systems [7, 8, 77, 121], except that SRAM data remains in place. Keeping SRAM data in place creates the central challenge that I address in this section: how does TotalRecall know when SRAM acted as a NVM while the power was off?

2.4.1 Challenge: Detecting Volatility

My experiments with SRAM remanence (§2.3.2) show that SRAM acts as a NVM given normal off times, but becomes volatile memory given sufficiently long off times. While experiments show that SRAM gradually transitions from completely non-volatile to completely volatile as the time off increases, I assume that any bit loss constitutes complete volatility. Thus, my goal is to design a mechanism to detect if any bit has changed during power loss. Below I explore the SRAM volatility detection design space.

Do nothing: The highest performance, but lowest reliability, solution is to assume SRAM retains all data. This incurs near-zero overhead (only registers are copied to SRAM) for

off times that stay within SRAM's non-volatile range; but leads to program failure for unexpectedly long off times.

Canary: To add some cheap error detection, I could add a canary value to my in-SRAM checkpoint. By checking if the canary value changed at power on, I would know if SRAM lost state. Unfortunately, the SRAM cells that fail first and the direction of failure is dictated by manufacturing-time process variation—hence each device is different [53, 54]. Thus, a canary only detects SRAM state loss—not retention.

Enrollment: A viable solution that has a low run time overhead and is reliable is to pre-characterize devices to determine where to place canary values in each device's SRAM. Characterization follows the same binary search procedure of my remanence experiments (§2.3.2), but at a finer granularity (to tease-out the specific SRAM cell that fails first). Characterization works because SRAM cells lose state in a predictable order that is robust against temperature and voltage changes [51]. I do not choose this solution because it limits the generality of TOTALRECALL; otherwise, this is the optimal solution.

Redundancy: Another approach that takes advantage of the unique failure pattern of SRAM is dual modular redundancy: duplicate all words in SRAM and after powering on, check for disagreement. While this provides stronger reliability guarantees than doing nothing or using canaries without enrollment, redundancy does not provide significant guarantees in the face of frequent power failures. Additionally, this approach has high memory overhead.

ECC: Error Correcting Codes (ECCs) are commonly used to protect DRAM memory. ECC is a poor fit for the bursty errors encountered by SRAM volatility. For example, the MSP430's word size is 16-bits; adding a parity bit to each data word incurs 6.25% overhead,

		$10\mu F$		$47\mu F$	
	Temp. (C)	16-bit	32-bit	16-bit	32-bit
Office	20	3.7	239K	17.2	1,126K
Death Valley	55	0.8	54K	3.9	256K

Table 2.3: Worst-case time, in years, until a silent data corruption.

but can only detect a single bit flip. Upgrading to 5-bits, guarantees two-bit error detection, but incurs a 31.15% overhead. In the average case, my design requires 4-bits per word of detection.

Hash function: The final solution that I rule out is employing a hash function. Using a hash function trades a huge performance loss for near-perfect reliability. To use a hash function, you would feed it the entire contents of SRAM and store the resulting digest in SRAM, as part of a checkpoint. Upon power on, you recalculate the digest and verify against the stored digest. The likelihood that the two digests match, but the SRAM lost state is less than 1 in 100 trillion. This reliability is overkill for the duty cycle of intermittent computation and expensive in terms of run-time overhead.

2.4.2 My Solution: Cyclic Redundancy Checks (CRC)

My goal is to maximize system performance without (practically) compromising data integrity. To this end, I find a solution that handles error bursts like hash functions, is designed with random errors in mind like ECC, but balances performance and integrity. Ideally, I want the cheapest solution that provides *just enough* data integrity. The solution that meets these requirements is a Cyclic Redundancy Check (CRC); CRCs are simple, fast, and provide tunable data integrity that allows system designers to trade performance for integrity. Another benefit of CRC is that, due to its prevalence (e.g., the WISP platform makes heavy use of CRC [123]), many microcontrollers provide hardware support for it.

Cyclic Redundancy Checks (CRCs) are error-detecting codes common in many communication and data storage applications to provide a high degree of confidence that a data packet is error-free. To verify a data packet, the CRC algorithm divides the data packet by a pre-determined generator polynomial using repeated shift and `XOR` operations. The sender stores/transmits the data packet along with the remainder of this division; when the data must be verified, the message is divided by the generator polynomial again. If the calculated remainder matches the stored remainder, the receiver can assume with a high degree of confidence that the message is correct. If the remainders do not match, the data is assumed to be corrupted. The shift and `XOR` operations used to calculate CRCs are simple to implement in both software and hardware.

Besides being efficient, CRCs have high error detection capability. A CRC is guaranteed to detect G bits of errors, where G depends (roughly) on the bit-width of the CRC; a 16-bit CRC guarantees detection of up to three flipped bits anywhere within the data, while a 32-bit CRC guarantees detection of up to five flipped bits [86]. For errors corrupting more bits than these, CRCs provide probabilistic error detection of $1/2^m$, where m is the width of the CRC. Such a multi-bit error is undetected only if the checksum of the corrupted and uncorrupted data match exactly. The probability of undetected corruption is further reduced because CRC guarantees detection of an odd number of bit flips and there is a 50% chance that SRAM fails in the direction of the value stored in the cell. Assuming each power failure is just long enough to corrupt data (see Figure 2.1), I calculate CRC's expected time to first undetected corruption for several energy storage capacitor sizes and CRC bit-widths. Table 2.3 presents the results of this calculation.

Algorithm 1 In-situ SRAM checkpoint routine.

```
1: ...Copy all CPU registers...
2: SRAM_Ptr = SRAM_BOTTOM
3: // CRC everything in SRAM but CRC value
4: while SRAM_Ptr ≠ SRAM_TOP do
5:     // Next input to CRC engine
6:     CRC_Input ← *SRAM_Ptr
7:     SRAM_Ptr = SRAM_Ptr + 2
8: end while
9: // Save CRC result to top of SRAM
10: *SRAM_TOP ← CRC_Result
11: ...Power off...
```

2.4.3 TotalRecall Overview

Like previous one-time checkpointing approaches [7, 8, 77, 121], TotalRecall creates a checkpoint immediately before power loss. TotalRecall performs all checkpointing actions in software by implementing them in an interrupt service routine associated with an interrupt-enabled voltage supervisor that monitors the system's energy storage capacitor. The checkpoint contains all architected state (e.g., general-purpose registers) and a CRC checksum computed from all other SRAM data. Note that the registers themselves are implemented using SRAM, thus they may retain data after power loss; however, TotalRecall still copies them as part of the checkpoint for generality, as some systems tie registers to hardware resets to ensure a known startup state. Unlike previous one-time checkpointing approaches, TotalRecall's checkpoints leave program data in-place in SRAM—eschewing costly Flash writes. After completing the checkpoint, the microcontroller powers-off to avoid inconsistent recovery [8, 120].

Upon power-on, TotalRecall's recovery routine gets invoked. It first recomputes the CRC over SRAM and compares it to the recorded checksum. In the common case when the old and new checksums match, TotalRecall restores the checkpointed register values from SRAM and then resumes program execution. In the uncommon case that the checksums do not match, TotalRecall restarts the user program from the beginning or from a conventional

Algorithm 2 In-situ SRAM checkpoint recovery routine.

```
1: SRAM_Ptr = SRAM_BOTTOM
2: while SRAM_Ptr ≠ SRAM_TOP do
3:     // Next input to CRC engine
4:     CRC_Input ← *SRAM_Ptr
5:     SRAM_Ptr = SRAM_Ptr + 2
6: end while
7:
8: if *SRAM_TOP ≠ CRC_Result then
9:     // SRAM corruption detected
10:    ...restart or load NVM checkpoint...
11: end if
12:
13: ...Restore all CPU registers...
14: Restore stack pointer
15: ...execute initialization callback, if necessary...
16: Restore status register
17: Restore program counter
```

checkpoint in NVM. While I envision more sophisticated ways of handling long off times, that is not the focus of this work.

2.4.4 Checkpoint Layout and Creation

TotalRecall's checkpointing procedure stores the checkpoint in a static location reserved at the top of SRAM. This simplifies CRC processing, makes the checkpoint and restore code more efficient, and enables TotalRecall to seamlessly integrate with existing programs at link time. This reserved address range is small (e.g., 40 bytes) because TotalRecall's checkpoint only contains register values, which are few in number, and a checksum. To prevent the checkpoint content from being inadvertently overwritten by the user program, I modify the memory map used by the linker script to start the stack just after my checkpoint location. By doing this, TotalRecall is able to integrate with existing programs as late as the linker. Algorithm 1 provides the details of TotalRecall's checkpointing routine.

2.4.5 Restoring from Checkpoints

On power up, TotalRecall's recovery routine executes before any platform initialization or user code executes. Algorithm 2 details TotalRecall's recovery routine. TotalRecall first re-calculates the checksum over SRAM (excluding the CRC value) and compares it to the checksum stored at the top of the in-SRAM checkpoint. When the checksums match, TotalRecall repopulates the register file, restores the stack pointer, and executes any peripheral initialization function that the user program registered (by writing its function pointer to a reserved space in the checkpoint area), finally passing control to where the program left off (by restoring the status register and the program counter).

Checksum mismatches result from two situations: (1) no checkpoint exists (i.e., this is the first power on event) or (2) the data in SRAM was lost during an a unexpectedly[7] long power cycle. In the first case, TotalRecall starts execution from the beginning of the program. In the second case, TotalRecall looks for a checkpoint in Flash, restoring it if it exists, otherwise restarting the program.

2.5 TotalRecall Implementation

To validate the applicability of TotalRecall to energy harvesting devices and intermittent computation, I implement and evaluate it on two Texas Instruments microcontrollers: MSP430G2553 and MSP430FR6989. Table 2.4 summarizes the relevant aspects of each device. From a high level, these devices represent existing energy harvesting devices [107, 123, 147] and cover both Flash and FRAM NVM options. From a low level, I select these specific devices because they are available as part of development boards.

[7]For expected long off times (e.g., sunset with a photovoltaic), write a checkpoint to Flash instead.

	Flash	FRAM
Read freq.	**25 MHz**	**8 MHz**
Write freq.	**.036 MHz**	**8 MHz**
Endurance	10^4	10^{15}
Min V.	**2.2 V**	**1.8 V**
Byte erase/program	**No**	**Yes**

Table 2.4: Microcontrollers that I implement TOTALRECALL on.

Power control board: The development boards have the same debug interface, which enables me to use the same power control board across both boards. I approximate energy harvesting conditions using a custom daughter board compatible with both MSP430 development boards, shown in Figure 2.2. Testing intermittent computation systems is difficult because energy harvesting environments are, by nature, unpredictable and difficult to replicate [49]. The power control board allows me to test TOTALRECALL and baseline systems in a consistent, repeatable environment. A microcontroller on the daughter board controls a Digital-to-Analog Converter (DAC) capable of powering the MSP430-under-test and transistors to isolate the MSP430-under-test when power is disconnected to better imitate energy harvesting circuit behavior.

Baseline implementation: To serve as a baseline for my evaluation, I implement the state-of-the-art one-time checkpointing approach (i.e., Hibernus [8]) on both microcontrollers. Being one-time approaches, the systems take a checkpoint when triggered by the interrupt indicating imminent power loss. A checkpoint consists of writing all registers and SRAM data to a static, reserved, location in NVM. Compare this to TOTALRECALL's checkpoints, which leave SRAM data in place, and only copies registers.

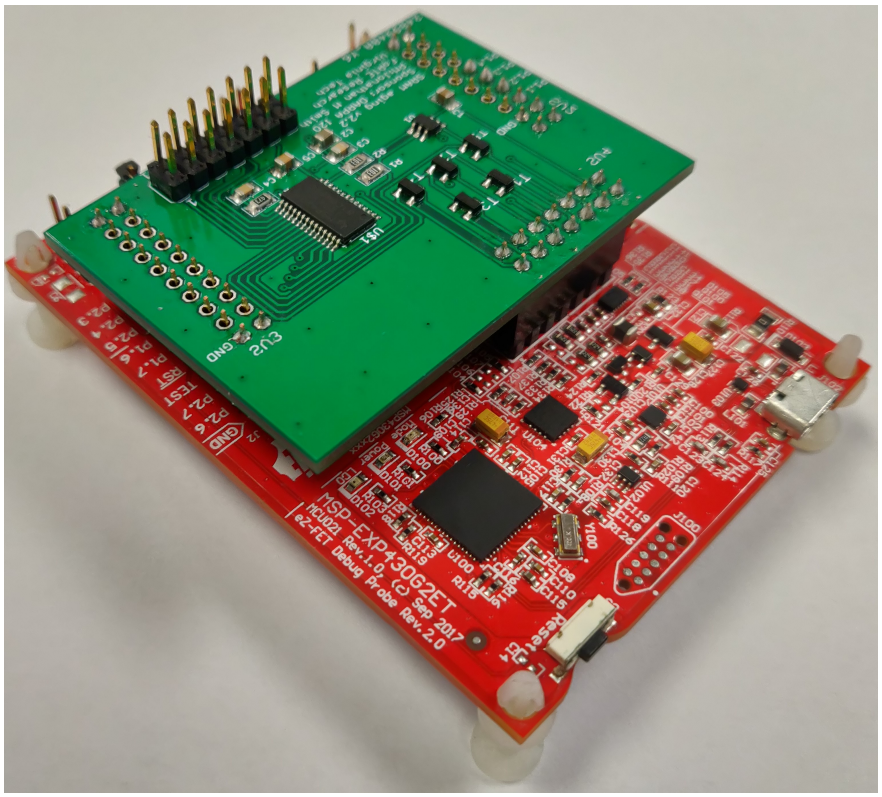


Figure 2.2: The experimental setup with power-control daughter board (top) and the MSP430G2553 Launchpad (bottom).

2.5.1 CRC Implementation

The main question that I answer in the evaluation (§2.6) is whether taking a CRC is more efficient than copying SRAM to NVM. TotalRecall incurs near-zero data transfer overhead because it does not access slower NVM and only needs to copy register data (40 bytes) to SRAM. Instead, the primary source of overhead is calculating the CRC checksum of SRAM data. I evaluate four different implementations of the SRAM integrity check: two table-based software routines applicable to any device, and two routines taking advantage of hardware CRC support common to modern microcontrollers.

Software CRC16/32: The software CRC approach is independent of hardware support, thus applicable to any system. My software CRC implementation is optimized for speed based on a memoization of CRC divisions; platforms with limited code space could sacrifice checkpoint and recovery speed for reduced code size by removing the CRC table and directly calculating the CRC remainder. Algorithm 3 shows the software CRC16 algorithm, adapted from Texas Instruments example code [69]. Not shown is a software implementation of CRC32. Software CRC32 is similar to CRC16, but with added operations to deal with 32-bit numbers on a 16-bit microcontroller. Experiments show that CRC32 takes roughly $1.6x$ the time of CRC16—but increases reliability (Table 2.3).

Hardware CRC16/32: Many low-power microcontrollers which transmit or receive data include hardware dedicated to calculating CRC checksums. The programmer writes data to the input register of the CRC engine, sequentially, until all data is processed; the updated checksum is available the next CPU cycle. I implement TOTALRECALL using both the CRC16 and CRC32 engines available on the MSP430FR6989 to measure the overhead improvement possible when CRC hardware is available. My results show that hardware acceleration increases checkpoint/recovery speed by 342% when compared to the software implementation of the CRC16 and 561% compared to the software CRC32.

2.5.2 Additional Challenges

I encountered a number of challenges implementing my design, primarily related to circumventing embedded system startup routines that assume no valuable data is in SRAM immediately after power-on. Startup routines such as `crt0` (responsible for setting up the C runtime environment) make function calls and allocate variables at the top of the stack because they run before anything else on the system. Because the system writes checkpoints

Algorithm 3 Software CRC16 routine.

```
 1: MASK ← 0xFF00
 2: CRC ← 0x0
 3: P_TABLE ← &CRC_TABLE
 4: P_SRAM ← SRAM_BOTTOM
 5: while P_SRAM ≠ SRAM_TOP-2 do
 6:    INDEX ← CRC[7:0]
 7:    INDEX ← *P_SRAM xor INDEX
 8:    P_SRAM ← P_SRAM + 1
 9:    INDEX ← INDEX + INDEX
10:    INDEX ← INDEX + P_TABLE
11:    CRC ← CRC and MASK
12:    CRC ← *INDEX xor CRC
13: end while
14: return CRC
```

to a static location at the top of the stack, TotalRecall must take care to avoid overwriting `crt0` and other runtime data during the checkpointing process.

Rather than re-run the runtime initialization code on every startup (which increases checkpoint recovery overhead) or re-instrument startup code to accomodate SRAM checkpoints (which increases time to deployment because startup routines are platform-specific), I modify the linker script to make the space used for storing checkpoint data unavailable to the compiler. This reduces the total available RAM space by the size of the checkpoint even if no checkpoints are taken; I consider this an acceptable penalty because register file checkpoints are small.

2.6 Evaluation

To validate TotalRecall's effectiveness and compare against the state-of-the-art one-time checkpointing approach, I evaluate TotalRecall and Flash-based one-time checkpointing using a set of benchmarks written for the MSP430 by Texas Instruments [58]. These benchmarks, shown in Table 2.5, include common embedded system applications and CPU per-

Benchmark	Size (B)	SRAM Usage (B)	Approximation?
FIR Filter	3572	254	Yes
Dhrystone	1285	474	No
Whetstone	16136	468	Yes
Quicksort	864	362	No
Factorial	1532	44	No
Matrix Mult.	1536	122	Yes

Table 2.5: Memory usage information for several benchmarks on the MSP430FR6989.

formance benchmarks. I also include `quicksort` and `factorial` benchmarks to test against stack-intensive programs. Table 2.5 includes the memory usage of the benchmarks, because one potential optimization for TOTALRECALL is to only compute the CRC over SRAM in use. This optimization is especially useful for programs that do not use dynamic memory. Table 2.5 also details which benchmarks are amenable to approximate computing techniques as I foresee that TOTALRECALL's use of SRAM as best-effort approximate storage has potential synergy with approximate computing systems, although I leave this line of research for future work.

I compile all benchmarks using the open-source MSP430 GCC toolchain developed by Texas Instruments [72] with -Os (optimization for size) enabled. I use the results of this evaluation to answer the following questions:

1. Does TOTALRECALL correctly stretch program execution across short yet frequent power cycles?

2. Can TOTALRECALL detect and handle when unexpectedly long power cycles expose SRAM's volatility?

3. How does TOTALRECALL's run-time overhead compare to existing one-time checkpointing approaches?

2.6.1 Correctness

Figure 2.3 shows a test of the system working in a typical energy harvesting environment. The supply voltage to the microcontroller varies as TotalRecall extends the `quicksort` benchmark across five power cycles on the MSP430G2553. Figures 2.3 and 2.4 illustrate three stages of the power failure the microcontroller experiences; these traces are taken on hardware using the Picoscope 2208B Mixed Signal Oscilloscope (MSO) [129]. Before the device disconnects from the power source, the supply voltage is steady at $3.3V$. Immediately after disconnecting power, the microcontroller continues execution, but rapidly drains the energy storage capacitor until V_{dd} reaches the brown-out voltage ($1.5V$). With the microcontroller not executing, V_{dd} drops slowly due to charge leakage from the microcontroller and surrounding development board. The rate of V_{dd} drop can be unpredictable depending on the behavior of power-hungry peripherals such as radios that share the power supply with the MCU. Hardware-oriented solutions such as energy federation [50] can mitigate this problem by intelligently choosing when to connect peripheral supply capacitors to the MCU power rail, reducing their impact on SRAM remanence time. This gradual drop in supply voltage allows data to remain in SRAM as it never goes below SRAM's data retention voltage. Thus, in the case of Figure 2.3, `quicksort` successfully completes execution after five power cycles. I perform similar experiments with all benchmarks, validating TotalRecall's ability to stretch execution across frequent power cycles. I conduct all experiments at 20C to reduce the impact of operating temperature variation on SRAM retention time.

What about when longer power cycles cause data loss? To show that TotalRecall can detect and handle cases where SRAM becomes volatile, I create data loss by driving V_{dd} below SRAM's data retention voltage. This causes some SRAM cells to lose their state. I perform variations of this experiment down to $0V$ and by disconnecting from power for a long time, instead of directly driving V_{dd}. In all cases, the recovery routine detects the data

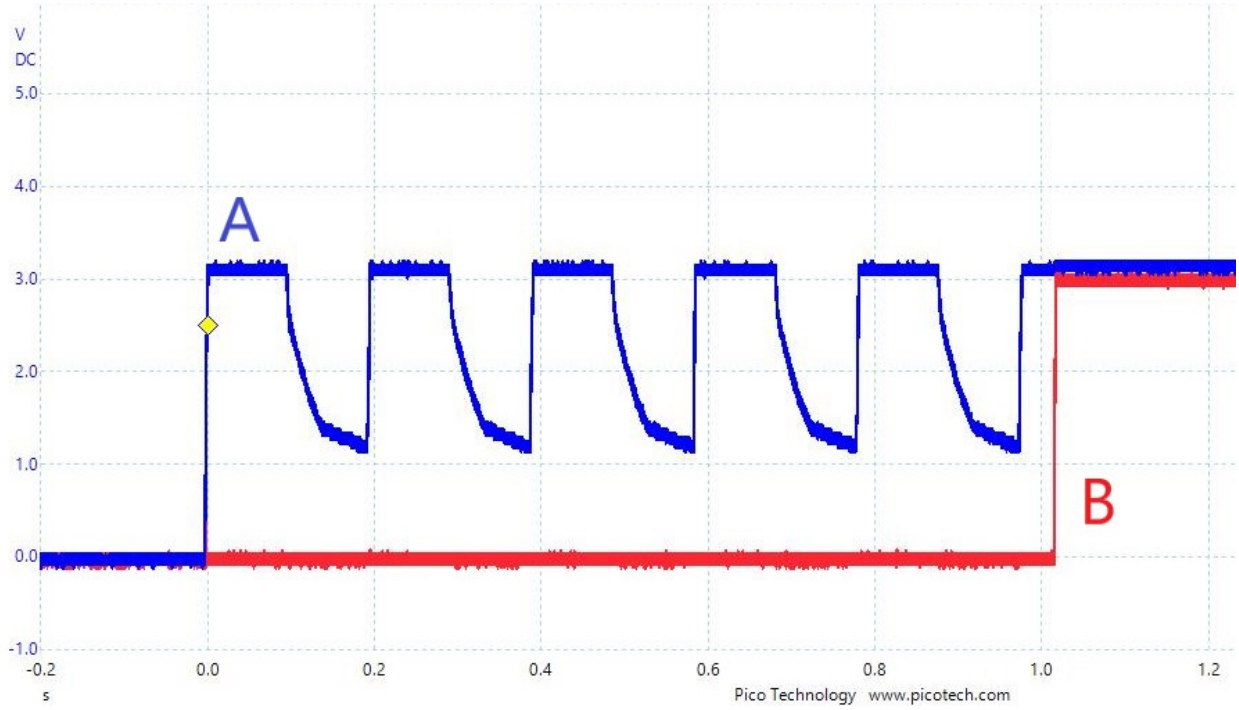


Figure 2.3: Extending the `quicksort` benchmark across five power cycles. Channel A shows supply voltage and channel B shows the status of a GPIO pin that is driven high when the program completes with correct output.

corruption and restarts the program from the beginning rather than continuing with faulty data.

2.6.2 TotalRecall's Overhead

Execution-time Overhead: TotalRecall correctly extends program execution across power cycles, but how does its performance compare to state-of-the-art Flash-based checkpointing solutions? Does it approach FRAM's overhead? To compare TotalRecall's overhead to Flash- and FRAM-based checkpointing systems, I time each checkpoint recording/recovery routine on the experimental hardware setup described in §2.5. Figure 2.5 shows these results as percentages of the total on-time spent in the checkpointing and recovery rou-

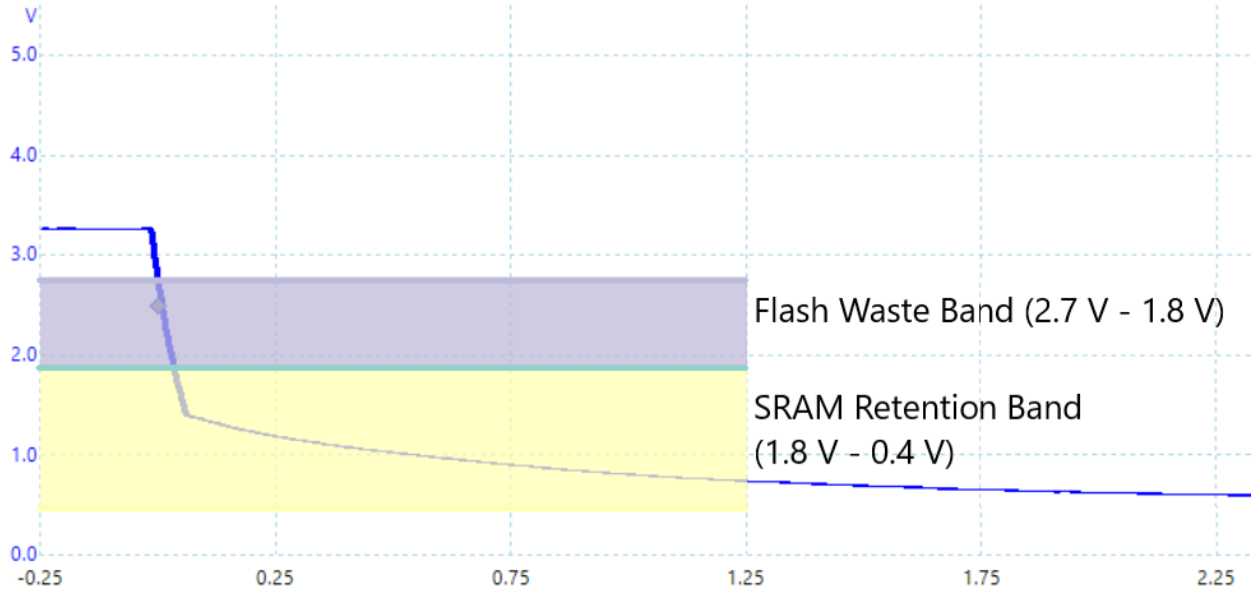


Figure 2.4: MSP430G2553 V_{dd} decay after power is disconnected. Flash checkpointing systems must begin writing a checkpoint well above the minimum Flash write voltage, wasting energy. TotalRecall allows computation to continue until just before V_{dd} reaches the brown-out voltage and uses the remaining energy in the system to retain SRAM state.

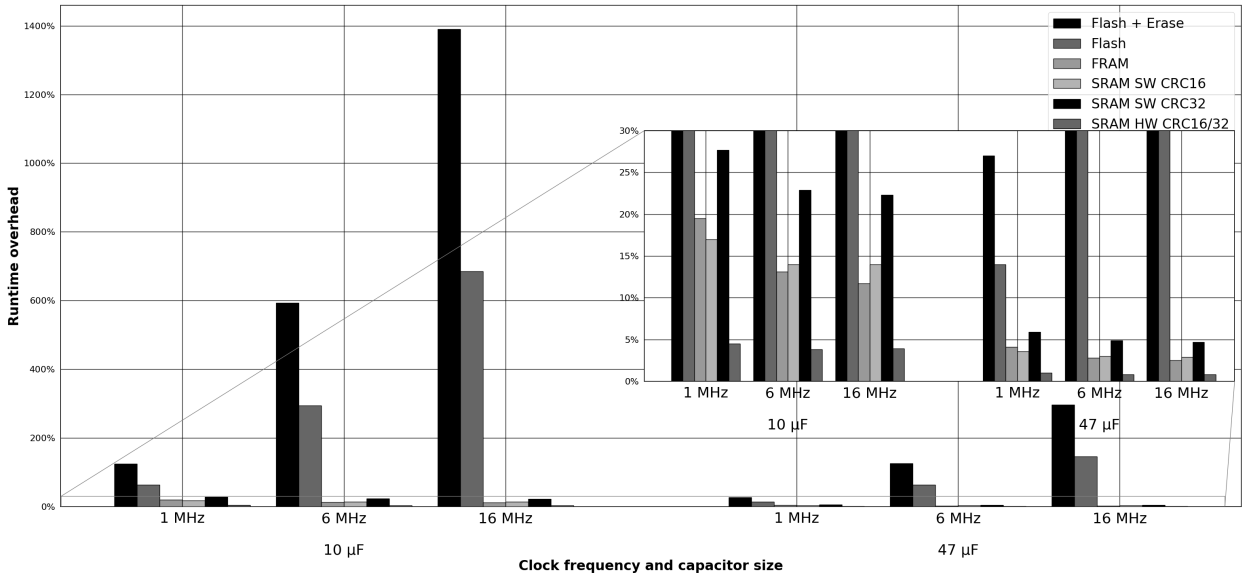


Figure 2.5: Comparison of one-time checkpointing system overheads on platforms with different energy storage capacitor sizes and clock frequencies. Flash+Erase represents overhead when the Flash page must be erased before writing the checkpoint; SRAM-based checkpoint systems are divided by CRC implementation (hardware or software) and bit-width.

tines at each capacitor, frequency, and device combination. An overhead result >=100% indicates that the checkpointing system does not support useful computation in the given configuration, because the time needed to record and recover is greater than the available time in a single power cycle. In a system with a 47μF capacitor operating at $1MHz$, the hardware CRC implementation of TOTALRECALL extends program execution with an average overhead of 1.0% compared to the 27% overhead of the `Flash+Erase`[8] system under the same conditions—a 96% reduction in overhead. TOTALRECALL also scales better with clock speed, supporting $16MHz$ execution with an overhead of 0.8%, while the fixed-time Flash write+erase incurs nearly 300% overhead. In summary, not only does TOTALRECALL outperform Flash-based checkpointing, **it enables intermittent computation on a wider array of system configurations**.

Checkpoint Voltage Guard Bands: In addition to faster checkpoint and recovery routines, TOTALRECALL reduces overhead by increasing the total charge available for useful calculations. One-time checkpointing systems use the concept of a voltage "guard band", which designates the minimum supply voltage at which the system must stop useful calculations and begin writing the checkpoint to ensure that the checkpoint is completely written before power is lost. Figure 2.4 shows a trace of the MSP430G2553 supply voltage during a power failure. Flash checkpointing systems must finish writing a checkpoint before V_{dd} reaches $2.2V$, the minimum voltage required to guarantee successful Flash writes on MSP430-family devices [61]. Because Flash writes are slow, the Flash checkpoint routine must begin well before V_{dd} reaches $2.2V$. For example, on similar hardware, the Flash-based approach Mementos [121] sets the beginning of the checkpoint guard band between

[8]Flash-based systems must spend time and energy to erase data. They can do it every power cycle if there is sufficient energy (`Flash+Erase`) or they devote an entire power cycle to an erase operation when there is insufficient energy to both erase and write in a single power cycle. Thus, real deployments of Flash-based checkpointing systems are not able to achieve the overheads shown in `Flash` due to the need to eventually erase.

$2.8V$ and $2.6V$. Given that the MSP430G2553 guarantees code execution between $3.6V$ and $1.8V$, Flash checkpointing systems waste 45% to 55% of the otherwise usable charge held by the capacitor.

TotalRecall maximizes energy use because it is not limited by the $2.2V$ Flash write minimum supply voltage; it exploits the microcontroller's full voltage range. Based on my test platform of the MSP430G2553 with the stock 11μF capacitor running at $1MHz$, I use Equation 2.1 to determine that TotalRecall can delay taking a checkpoint until V_{dd} reaches $1.837V$. This leaves 98% of the total useful capacitor charge available to the system for useful computation.

$$V_{guard} = (\overbrace{\Delta t}^{\text{Checkpoint time}} * \underbrace{I}_{\text{Active current}} / \overbrace{C}^{\text{Total capacitance}}) + \underbrace{V_{low}}_{\text{Min. voltage}} \tag{2.1}$$

TotalRecall on FRAM Platforms: FRAM and other emerging NVM technologies such as Spin-Transfer Torque Magnetoresistive RAM (STT-MRAM) are potential alternatives to Flash on intermittent computing platforms due to their lower write currents, higher write endurance, and lower write latency. The state of the art for one time checkpointing systems uses FRAM-based microcontrollers that enable NVM-based checkpoints with approximately 15-20% time and energy overhead [7, 8]. I implement TotalRecall on the FRAM-based MSP430FR6989 to compare its performance to FRAM-based checkpoints. My evaluation in Figure 2.5 shows that TotalRecall, with a software-based CRC implementation, enables checkpointing performance competitive with state-of-the-art FRAM checkpointing. On a system with a 10μF capacitor operating at $1MHz$, TotalRecall using a software CRC32 implementation introduces 27.7% overhead compared to FRAM's 19.5% overhead. However, TotalRecall's capacity for hardware acceleration allows it to outperform FRAM-based checkpoints on devices that include hardware CRC support: on the same system, but using hardware CRC support, TotalRecall incurs 4.5% overhead—a 77% reduction. These

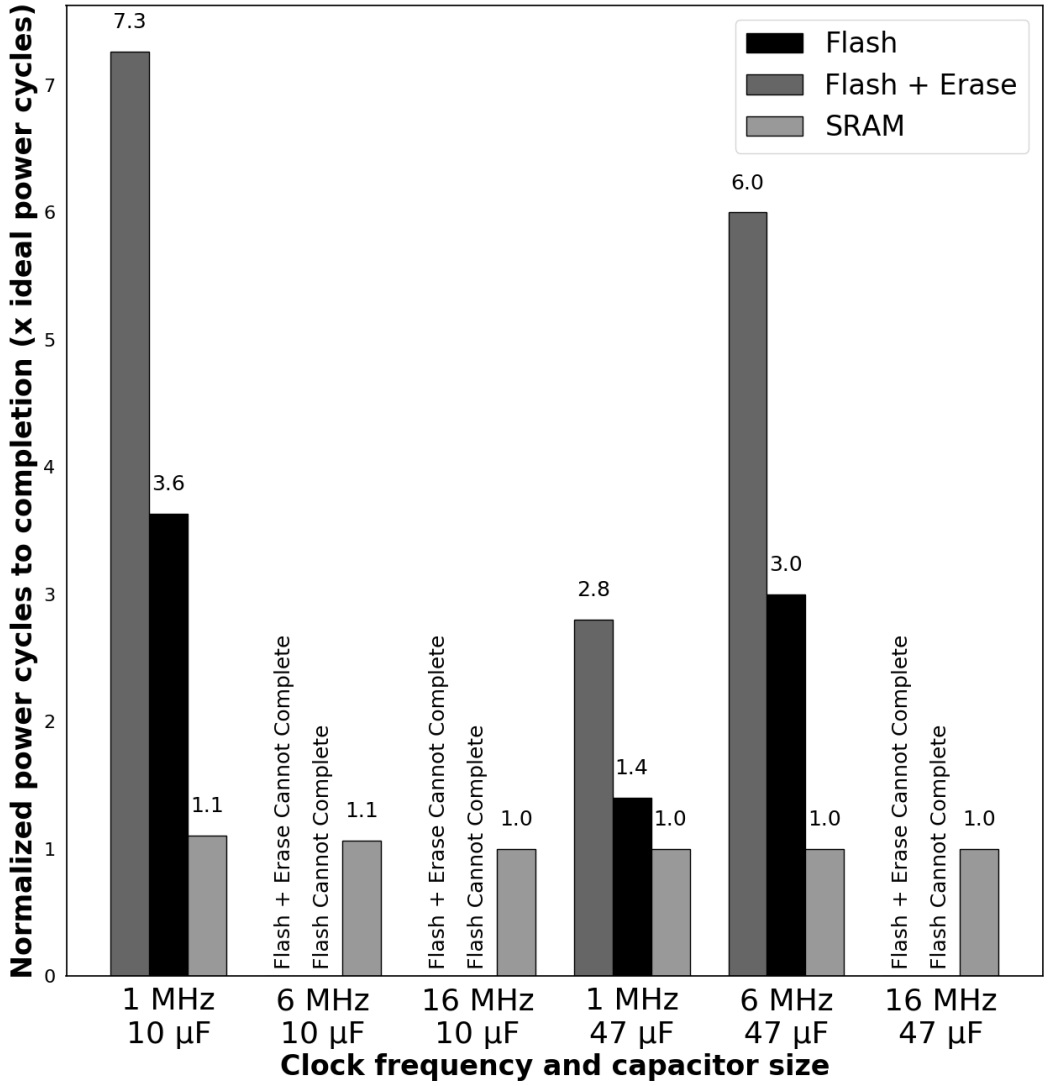


Figure 2.6: Power cycles required to complete the `FIR` benchmark normalized to continuous execution. `Flash+Erase` represents erasing every other power cycle.

---

results highlight TotalRecall's potential for high speed, NVM-agnostic checkpoints.

2.6.3 TotalRecall Practical Considerations

To gauge TotalRecall's practical impact on intermittently executed programs, I determine the number of power cycles required to complete a benchmark program running on the

MSP430G2553. I model the total active time available as the time required to drain a capacitor from the maximum operating voltage ($3.6V$) to the minimum voltage ($2.2V$ for Flash and $1.8V$ for SRAM) while drawing the typical active mode current at the specified clock frequency. Figure 2.6 depicts the number of power cycles required to complete the benchmark, normalized to the power cycles required if there were no checkpointing overhead (i.e., 100% of the active time was dedicated to running the user program). Operating at $1MHz$ with a 10μF capacitor, TotalRecall reduces the number of power cycles (and thus the total energy) required to complete the benchmark by over $7x$ compared to Flash-based checkpointing, while enabling intermittent computation at higher clock speeds and with smaller energy storage capacitors.

2.7 Related Work

TotalRecall represents a shift in how intermittent computation systems maintain state across power cycles. All previous work, because it assumes instant loss of volatile state, involves storing, in the form of a checkpoint, volatile state (e.g., registers and SRAM) to non-volatile memory (e.g., Flash and FRAM). Two classes of checkpointing approaches exist: one-time checkpointing approaches that store all volatile state just before power loss and continuous checkpointing approaches that are power-failure-agnostic and make many, smaller checkpoints that ensure consistent recovery. In this section, I discuss the progression of advancement in each approach class, including a look at work that extends intermittent computation beyond the microcontroller.

2.7.1 One-time Checkpointing

Mementos [121] is the first system to tackle the problem of stretching computation across frequent power cycles. Mementos relies on periodic measurements of the system's energy storage capacitor, coupled with an energy model, to estimate how much energy remains. The goal is to commit a checkpoint just before power failure. While Mementos works well when programs treat non-volatile memory as read-only, follow-on work exposes state inconsistency when programs modify non-volatile state during post-checkpoint execution [120]. The problem is that Mementos allows for uncheckpointed work to occur. If uncheckpointed work updates both volatile and non-volatile memory, only the non-volatile memory updates persist, creating an inconsistent state upon recovery.

QUICKRECALL [77] addresses Mementos's correctness issue by storing all program data in non-volatile memory; effective, but expensive due to non-volatile memory's speed limitations. Hibernus [7, 8] solves the problem in a different way: through the introduction of guard bands and hibernation. A guard band is a voltage threshold that represents the amount of energy required to store the largest possible checkpoint to non-volatile memory in the worst-case device and environmental conditions. Execution occurs only when voltage is above the guardband threshold. Beyond correctness, Hibernus also improves upon Mementos in terms of performance. To avoid the overhead of polling the voltage of the energy storage capacitor's voltage and the risk of energy estimation, Hibernus leverages the Analog-to-Digital Converter's (ADC) interrupt mechanism. Hibernus configures the ADC to fire an interrupt when voltage dips below a the guard band threshold. More recently, ReplayCache [146] transparently extends one-time checkpointing systems to support volatile data caches supporting NVM by replaying potentially unpersisted stores following a power loss. While ReplayCache is implemented at the software level and targets existing volatile data caches, WL-Cache [22] modifies the cache at the architectural level to automatically

flush dirty cache lines to NVM and tune cache behavior to energy availability.

As Hibernus represents the most correct and highest performance one-time checkpointing system suitable for our uplatforms, TotalRecall uses it as the baseline for comparison. As my evaluation shows (§2.6.2), for the FRAM-based systems targeted by Hibernus and QuickRecall, guardbands are reasonable due to the low time and energy cost of FRAM-based checkpoints. But, the time and energy costs of Flash-based checkpoints make adapting Hibernus's approach to the Flash-based systems targeted by Mementos prohibitively expensive. TotalRecall, by keeping checkpoints in-place in SRAM, enables it to provide improved performance—with smaller guardbands—for both Flash- and FRAM-based intermittent computation systems.

2.7.2 Continuous Checkpointing

The more recent continuous checkpointing systems eschew taking a single, large, checkpoint, for many, smaller, checkpoints. The driving observation that underlies such approaches is that the short on-times of intermittent computation limits the amount of state changed during a power cycle. Thus, the checkpoint needed to track such a change is also small. The challenge is avoiding the incorrectness of Mementos that occurs due to post-checkpoint execution [120]. To avoid state inconsistency during recovery, continuous checkpointing systems must track program execution at a fine-grain, inserting checkpoints where consistency demands. Researchers approach this problem from two directions: compiler tracking and hardware tracking. Note that no matter the execution tracking approach, continuous checkpointing is incompatible with Flash-based devices due to the time and energy overheads of Flash writes/erases and Flash's limited write endurance.[9]

[9]Even with systematically elided checkpoints [94], continuous checkpointing induces Flash write failure in less than a week [70].

Compiler-directed

DINO [91], is a software-only, programmer-driven checkpointing scheme for intermittent computation. With DINO, programmers write programs using annotations that define a series of independent tasks, backed by volatile data versioning (i.e., checkpointing) to achieve recovery consistency. By doing this, DINO provides the notion of task-based atomicity. Chain [23] extends Dino with a more well-defined data passing interface between tasks that reduces overhead through checkpoint size reduction at the expense of requiring more complex programmer reasoning about possible data interfaces. Alpaca [97] extends DINO and Chain through the dynamic privatization of statically-identified inter-task data. Alpaca detects shared data using idempotence analysis and copies only the identified data into a private per-task buffer. This further reduces log/checkpoint size to just the buffered data—ignoring all volatile state.

Task-based intermittent systems afford a low overhead, but required programmers to reason correctly and statically about the effects of power loss. An automatic alternative is Ratchet [137]. Ratchet is a compiler that decomposes unmodified code into a series of checkpoint-connected idempotent computations. Tracking idempotency allows Ratchet to add overhead only on the subset of non-volatile memory writes that are critical for recovery consistency. Chinchilla [94] improves upon Ratchet with guidance from a smart timer and basic-block-level energy estimation. Like Ratchet, Chinchilla's compiler inserts the checkpoints required to make correct forward progress with very short on-times. Like Mementos, Chinchilla includes a timer-based runtime component that deactivates checkpoints when there is enough energy to make it to the next checkpoint. In many cases, this eliminates 99% of Ratchet's checkpoints, while maintaining correct execution. Sweepcache [148] extends compiler-based intermittent computation to systems with volatile caches between the processor and NVM, flushing dirty cache lines to NVM at compiler-chosen boundaries in

software.

Hardware-directed

Idetic [102] is the first hardware-based system for intermittent computation. Idetic takes applications, creates a hardware circuit from them using existing high-level synthesis tools, and inserts non-volatile checkpoints in the resulting circuit's control-and-data-flow graph. While Idetic works for simple applications and single-function hardware, it is not general purpose. Non-volatile processors [92, 93] generalize the Idetic approach by incorporating non-volatility into existing processor pipelines (e.g., via non-volatile flip-flops). Recent work mixes-in approximation to improve performance [37] for applications that benefit from partial results. Lastly, in an approach closer to compiler-directed approaches than to other hardware-directed approaches, Clank [52] inserts in-hardware idempotence monitors in the memory hierarchy to better identify idempotence violations (compared to Ratchet) and to buffer idempotence-breaking writes to maximally delay checkpoints. Elastin [19] extends on the idea of Clank, but at page granularity, which is used by more complex systems.

2.7.3 Checkpointing Beyond MCUs

While the focus of most intermittent computation research targets decreasing checkpointing overhead, other important problems exist. In the first line of work the focus is on peripherals. Most intermittent computation deployments involve sensors, thus depend on peripherals. One hidden problem of peripherals is that initialization often takes longer than a single power cycle [27]. Samoyed addresses this problem with a one-time checkpointing system that runs user-annotated peripheral functions by disabling checkpoints and undo-logging [95]. Samoyed guarantees progress by energy profiling, dynamic peripheral workload

scaling, and a user-provided software fallback routine. The second line of work focuses on extending intermittent computation to x86-class processors. Work in this direction includes architectural support for out-of-order superscalar processors [92] and compiler-based techniques that adapt to x86-class systems [20, 90, 117].

2.8 Conclusion

TOTALRECALL exploits the time-dependent volatility of Static Random-Access Memory (SRAM) to eliminate the need for expensive checkpoints to Flash-based non-volatile memory for the off times common to intermittent computation. To address the central challenge of identifying if SRAM acted as a non-volatile memory during the off time, TOTALRECALL uses a Cyclic Redundancy Check (CRC) before and after a power cycle to validate SRAM data integrity. My evaluation on real hardware with unmodified benchmarks, shows that, compared to the state-of-the-art one-time checkpointing approach applied to Flash-based systems, TOTALRECALL increases performance by up to $370x$, while dramatically increasing system lifetime. On microcontrollers that have hardware support for CRC, performance surpasses FRAM-based checkpointing.

These results show that is is possible to have better than FRAM performance on the more widely deployed, more available, and higher performance Flash-based systems. Beyond validating TOTALRECALL's approach to supporting intermittent computation, these results also open the door to a wave of future approaches that leverage the synergy between the relatively short off-times common to intermittent computing and SRAM's time-dependent volatility.

Chapter 3

Hardware Support for Just-in-Time Intermittent Computation

3.1 Introduction

Continuous advances in the design and manufacture of tiny, low-power computing devices have opened the door for microcontrollers in applications previously limited by size, power, or cost constraints. Today's microcontroller-based sensor motes are small enough to monitor cellular temperature [138] and cheap enough to be deployed in high volumes inside of groceries [14] or with consumer goods to secure supply chains [56]. These advances have also enabled the use of sensor motes in more extreme and inaccessible environments such as space [30], deep underwater [82], or even embedded in concrete [2]. The challenge for today's designers is to build systems that best leverage this rapid down-scaling of computing hardware.

One major hurdle for the widespread deployment of tiny computing platforms is batteries, which have not experienced the same level of continual scaling as transistors. A typical lithium battery measuring 1 cm^3 can supply a low-power microcontroller drawing 300 μW for less than 4 months [119], after which the device is useless without a battery replacement—which is at best costly and at worst infeasible. Batteries also carry a risk of fire or explosion, limiting their use in sensitive applications such as medical implants, space deployments, or

aircraft. The limitations of batteries are driving work in a new direction: energy harvesting platforms, which replace the battery with a transducer to capture energy from the environment and a buffer capacitor to store the gathered energy until it is sufficient to power the on-board devices. The source and amount of available energy depends on the operating environment: many deployed systems are powered by RFID readers [14, 84], while other promising energy harvesters leverage thermal [92], photovoltaic [49] or piezoelectric [127] effects.

While energy harvesting opens up new opportunities for self-sufficient devices, it also poses challenges for the system designer. The unpredictable nature of harvested energy, low power output of transducer circuits, and energy buffer size limitations mean that a microcontroller running on harvested energy may maintain operation for a few hundred milliseconds—too short for many useful software applications to complete before power fails and program state is lost. Past work proposes a variety of techniques to support long-running program execution across numerous and frequent power failures, the most promising approach being *just-in-time checkpointing*: saving a snapshot of the program state to non-volatile memory when power failure is imminent [7, 8, 77, 121, 134].

Unfortunately, the requirement of a voltage monitor limits just-in-time checkpointing approaches. To know when power failure is imminent, just-in-time approaches track the voltage across the buffer capacitor (voltage is a surrogate for energy) using an Analog-to-Digital Converter (ADC) [121] to measured the voltage then comparing the measurement to a user-defined voltage threshold. However, ADCs are among the most power-intensive peripherals available on modern low-power microcontrollers: integrated ADCs typically consume *as much or more power than the processor core itself* (see Table 3.1), reducing useful computation time by over 50% even before considering the software overhead introduced by checkpoints. Recent just-in-time approaches replace ADCs with their lighter weight cousins,

the analog voltage comparator [7, 8, 77, 95]. This decision trades lower voltage resolution for marginally decreased current consumption. No matter the mechanism to monitor voltage, recent advancements in just-in-time checkpointing have exposed the voltage monitor as the primary source of run-time overhead—over an order of magnitude more than checkpointing [134].

The key issue is that ADCs and analog comparators are not optimized to support intermittent computation. For intermittent computation use cases, energy efficiency is paramount as long as resolution and sample rate are sufficient. Existing voltage monitors have been optimized in the opposite direction: performance first, then energy. The key to unlocking the promise of just-in-time approaches is a low-power, all-digital, on-chip supply-voltage monitor with just enough resolution and sample rate to meet the demands of current and future intermittent computation use cases. With such a voltage monitor, it is practical to make energy availability a first-class abstraction provided by the hardware, improving existing intermittent computation systems (Section 3.2.3) and enabling future ones.

In order to enable efficient voltage monitoring on energy harvesting systems, I develop *Failure Sentinels*—a low power, all digital, reference-free, on-chip voltage monitor designed to scale with the rest of the system. *Failure Sentinels* leverages the predictable gate-delay response of digital circuits to a changing supply voltage to inform software decisions about available energy; *Failure Sentinels* works by counting the number of times a signal traverses a self-oscillating feedback loop during a fixed time period as a reference-free indicator of buffer capacitor voltage, which itself indicates available energy. I design *Failure Sentinels* using only CMOS (Complementary Metal-Oxide Semiconductor, the technology of choice for digital integrated circuits) logic, ensuring that it scales along with the rest of the system's digital logic and take advantage of the corresponding price, power, and size benefits. *Failure Sentinels* exposes a broad design space to system designers to allow them to tune a

variety of performance parameters such as resolution and sample rate to find the balance of performance and energy consumption that is just-right for their use case.

I implement and evaluate *Failure Sentinels* in SPICE and on a RISC-V-based system-on-chip running on an Artix-7 Field-Programmable Gate Array (FPGA) [140]. I use the SPICE implementation to explore *Failure Sentinels*'s trade space across process nodes and voltages. I use the FPGA implementation to validate the SPICE results and to provide a real-world demonstration of *Failure Sentinels*'s performance. Finally, I evaluate *Failure Sentinels* against analog alternatives on energy harvesting power traces to explore the system-level impact. My evaluation indicates that *Failure Sentinels* reduces runtime overhead by 24%–70% compared to existing solutions, provides a flexible and scalable design space, and enables a variety of system designs previously limited by voltage monitor options.

This work makes the following three contributions:

- I evaluate existing voltage monitors and identify the power and scalability as the primary hurdles for their use in current and future intermittent computation systems (Section 3.2.2).
- I design *Failure Sentinels*, an on-chip voltage monitor (Section 3.3). *Failure Sentinels* leverages the power and space scaling of wholly-digital circuits and enables designers to build in *just enough* resolution and sample rate to meet at near-zero additional power and area (Section 3.4.2).
- I build *Failure Sentinels* in simulation and on FPGA hardware to explore the power/accuracy trade space that different *Failure Sentinels* implementations expose to designers (Section 3.4). My results show that *Failure Sentinels* improves intermittent system performance by up to 77% by eliminating a major source of power consumption, freeing up energy for useful computation (Section 3.5).

3.2 Background and Related Work

Although the size and power consumption of modern devices continue to decrease, harvested energy is typically too weak and unreliable to guarantee enough power to continuously support current microcontrollers [49, 77, 121, 134]. Instead, energy harvesting circuits slowly feed power into a buffer capacitor until enough energy is available to support a short burst of computation. Once computation starts, the microcontroller and peripherals rapidly drain the capacitor until the system reaches the minimum operating voltage, and the charge-discharge cycle repeats. The limitations of the harvesting circuit mean that devices running on harvested energy can restart dozens of times per second [15, 121]. Given that programs and programmers alike are not prepared for such operating conditions, previous work proposes a variety of strategies to stretch long-running computation across frequent power cycles, referred to as intermittent computation.

3.2.1 Supporting Intermittent Computation

Most current systems to support software on intermittently-powered platforms fall broadly into one of two categories; while each commits some portion of volatile memory (typically architectural registers, main memory, and any peripheral registers) to non-volatile memory, they can do so *just-in-time* before power failure [7, 8, 77, 95, 121, 134] by measuring available energy or *continuously* [23, 52, 91, 97, 137] throughout execution. The choice of checkpointing strategy is the primary determinant of system performance. Just-in-time systems theoretically maximize performance by only recording one checkpoint per power cycle and simplify software's interface as existing software is supported by linking against a library-level interrupt handler, but they depend on a voltage monitor attached to the buffer capacitor that interrupts computation to store a checkpoint when voltage falls below

Platform	MSP430FR5969 [67]	PIC16LF15386 [99]
Core I_{in} (μA/MHz)	110	90
ADC I_{in} (μA)	265	295
Comp. I_{in} (μA)	35	75
Core V_{min} (V)	1.8	1.8
Ref. V_{min} (V)	1.8	2.5

Table 3.1: Core versus ADC/comparator power requirements of sensor-mote-class microcontrollers, including voltage reference draw.

a threshold value (indicating imminent power loss). Unfortunately, practical considerations limit the applicability and performance of just-in-time approaches as existing voltage monitoring solutions are ill-suited for the voltage monitoring use case. The key to unlocking the promise of just-in-time approaches is a low power, scalable, on-chip supply-voltage monitor with just enough resolution and sample rate.

3.2.2 Monitoring Supply Voltage

Modern low-power microcontrollers include two components suitable for supply voltage monitoring: an Analog-to-Digital Converter (ADC) and an analog comparator. Unfortunately, the signal-processing focus on resolution and sample rates driving ADC design makes them unsuitable for supporting intermittent computation because of their relatively high power consumption: Table 3.1 shows that each component (and supporting circuitry)[1] requires current on-par with the processor itself.[2] This means that over half of the energy harvested is wasted on checking for imminent power failure—as opposed to computation. The wasted energy will only increase for future systems due to the discrepancy in scaling between digital logic and ADCs: performance/Watt for processors tends to scale at 2x every 1.57 years [31], while performance/Watt scales at 2x approximately every 2.6 years [80].

[1]Both components require a reference voltage to compare against the measured signal, typically provided by a diode [60] or internal bandgap reference generator [38, 67].

[2]While discrete low-power ADCs exist [64], their cost is on par with the microcontroller itself and their standalone nature adds size and complexity to the system.

To avoid the power-hungry nature of full-fledged ADCs, recent intermittent computation systems employ single-bit analog comparators [7, 8, 95]. While single-bit analog comparators improve on ADCs, they still waste 21%–45% of harvested energy on reference voltage generation. Single-bit solutions also limit utility as many current and emerging intermittent computation systems demand dynamic, fine-grain, and poll-able voltage monitoring; essentially, the ideal solution is making available energy a first-class abstraction provided by the hardware to software at near-zero energy cost.

3.2.3 Enabling Future Intermittent Systems with Practical Voltage Monitoring

The first systems to address intermittent computing on small batteryless devices focus on *enabling* long-running intermittent programs [23, 91, 121, 134, 137]; more recent ones focus on *optimizing* it [7, 39, 94, 97]. Since checkpoints are one of the drivers of run time overhead (on-par with voltage monitoring), one way to improve performance is to eliminate superfluous checkpoints. Chinchilla [94] is a timer-augmented continuous checkpointing system that improves performance through energy-guided checkpointing. Chinchilla dynamically tunes a timer to the expected on time and skips checkpoints that occur before the timer expires. Despite the challenges of representing energy in a timer value, Chinchilla yields a 2x–4x performance boost over similar checkpointing systems. Chinchilla must be overly pessimistic on available energy and energy usage to maintain correctness. With a practical voltage monitoring solution, Chinchilla is able dynamically query available energy and remove its guard bands; this increases performance, while also increasing system reliability.

Work beyond checkpointing presents a variety of energy-efficient techniques tailored for energy harvesting. PHASE [32] makes the case for single-workload heterogeneous architectures,

switching between high-performance and high-efficiency systems for the same workload depending on the availability of ambient power. HarvOS [9] profiles the energy requirements of each section of code and schedules software execution or non-volatile checkpoints accordingly. Dewdrop [15] similarly balances task execution and sleeping depending on available energy to make the most of changing ambient energy conditions. These systems all promise significant performance boosts but depend principally on *low cost, on-demand measurements of remaining energy.* While ADCs can fulfill this role, their high resolution and high sample rate are overkill and steal up to 50% of energy from software. The goal of this work is to *enable these and other technologies* through a low power, all digital (i.e., scalable), on-chip voltage monitor that provides just enough resolution and sample rate; in doing so, I make energy availability a first-class hardware abstraction.

3.3 *Failure Sentinels* Design

I design *Failure Sentinels*: a low power, fully digital, software-programmable voltage monitor optimized to minimize power consumption while meeting the resolution and sample rate needs of current and future intermittent computation systems. Two intermittent-computation-focused design goals drive my approach:

- ***Failure Sentinels* must minimize power consumption and provide *just enough resolution and sample rate* to serve software's needs.** Section 3.2 shows that existing voltage monitors fail at this, whereas *Failure Sentinels* provides enough resolution to support frequent, voltage measurements, while avoiding the design and power concerns associated with DSP-focused ADCs.

- ***Failure Sentinels* must be *compact and fully-digital* to enable ubiquity and**

scalability: As energy harvesters find their way into micro-healthcare and smart dust applications, miniaturization of every component in the system is a primary concern. Implementing *Failure Sentinels* with solely the CMOS gates used for digital logic ensures that it can be incorporated into any device and scales with process technology.

3.3.1 Ring Oscillators

Digital systems operate correctly at a wide range of voltages using well-chosen clock frequency guard bands, which hide the extreme sensitivity of the underlying circuits to voltage changes. Removing these guard bands from an otherwise digital circuit reveals analog-domain latency changes, which in turn reveal the system voltage. Desktop-class systems use this effect by measuring the propagation of a signal through lines of digital delay elements to support dynamic voltage and frequency scaling [16, 42, 110], but their narrow voltage range—beyond which the input either propagates entirely or not at all—makes them ill-matched for the voltage monitoring required by intermittent systems. Feeding the output of the delay line into the input such that that the output changes each time it passes through the entire delay line (i.e., it is self-oscillating) forms a Ring Oscillator (RO) with an output frequency that is primarily a function of supply voltage and a dynamic range covering nearly the entire voltage at which the RO oscillates. **This work leverages the voltage-dependent nature of RO frequency to measure supply voltage**.

The RO is a common circuit with applications in clock generation [43], process tuning and characterization [11, 46, 47], and performance monitoring [18]. Ring oscillators are attractive options for these applications for their ease of integration into IC designs, low power consumption, and electrical tunability [98]. The basic RO structure is an odd-numbered ring of digital inverters as shown at the bottom of Figure 3.2. Because the output of an

odd-numbered chain of inverters is the inverse of the input, feeding the chain output back to the input produces a circuit that oscillates as long as power is applied. RO length is largely application-dependent, but is typically prime to reduce potential harmonic oscillations [10].

The frequency of oscillation depends on the length n of the chain and the gate delay of each inverter τ_d as shown in Equation 3.1 [98].

$$f_o = \frac{1}{2n\tau_d} \tag{3.1}$$

With a constant chain length, the RO output frequency is entirely dependent on average gate delay. Several factors affect the gate delay: the designer tunes gate delay by changing the transistor size or supply voltage, while temperature and manufacturing variations also play a role. Among these, voltage is the dominant factor [81, 142].[3]

3.3.2 Voltage-Frequency Relationship

To characterize supply voltage's effect on RO frequency, I run a comprehensive set of SPICE simulations on ROs of varying length, operating at a range of supply voltages using the Predictive Technology Models [17] for the 130nm, 90nm, and 65nm technology nodes. I choose these feature sizes because they are representative of the technology currently used on energy harvesting platforms [136] as well as the logical next feature size for future systems. I sweep the supply voltages from 0.2 V (below which the rings do not oscillate) in 100 mV steps up to 3.6 V, the maximum supply voltage for typical energy-harvesting-class devices [60, 67, 99].

[3]Ring oscillators also have potential as fully-fledged ADCs [26, 124], but the signal-processing focus of these designs precludes them for efficient supply monitoring. However, research in this area to linearize the RO frequency-voltage curve and reduce sensitivity to process/temperature variations [79, 115] has potential to improve *Failure Sentinels*.

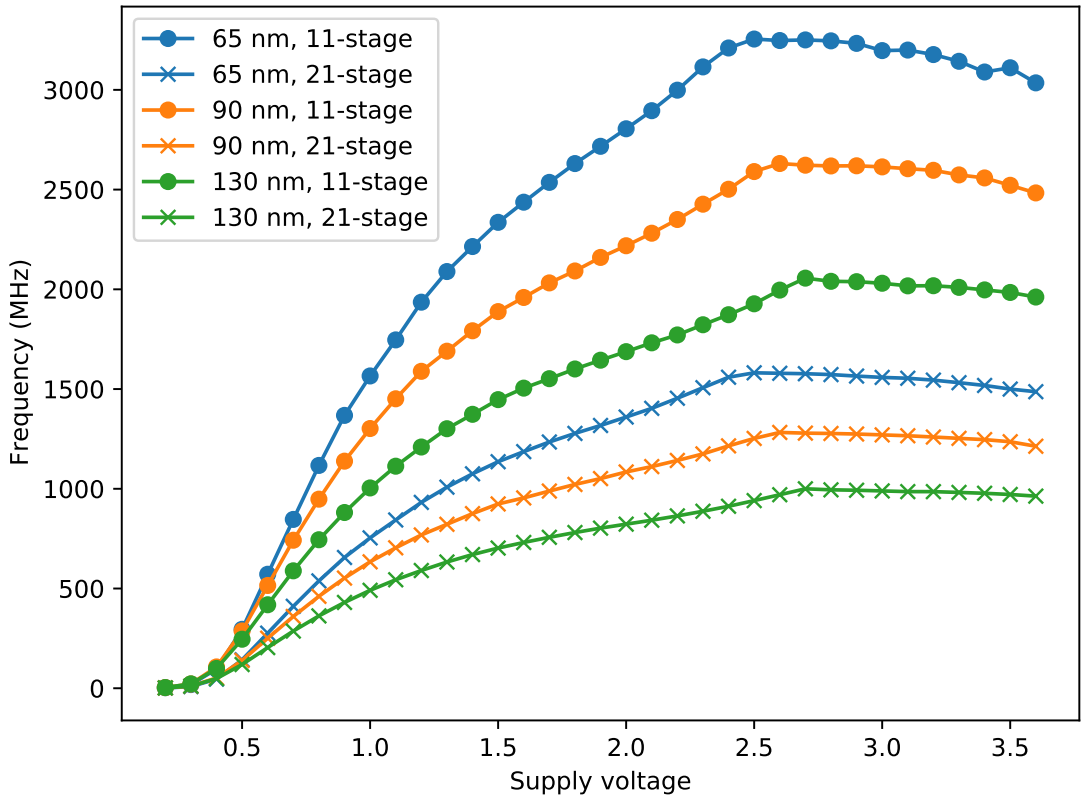


Figure 3.1: RO frequency vs. supply voltage at different feature sizes.

Figure 3.1 illustrates the results of these simulations using 11- and 21-stage ROs in each technology. I use these results to make three key observations motivating and informing the design of *Failure Sentinels*:

- The high sensitivity of frequency to voltage makes ROs viable supply voltage sensor, and the sensitivity increase from moving to smaller processes means that *Failure Sentinels* improves as technology scales.

- Decreasing RO chain length magnifies the effects of supply voltage changes, increasing sensitivity.[4]

[4]Decreasing RO length also increases the frequency and therefore current consumption of supporting circuitry; I explore the tradeoffs of different RO lengths in more detail in Section 3.5.1.

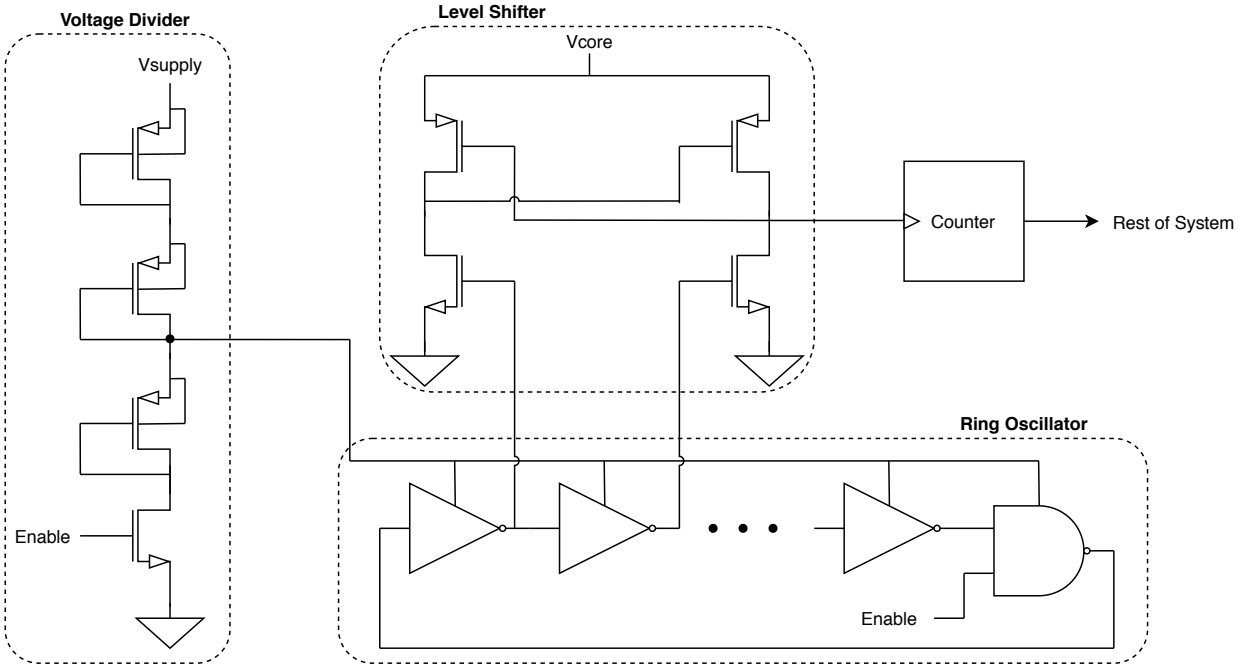


Figure 3.2: *Failure Sentinels* block diagram with a divide-by-3 voltage divider. Not shown is a level shifter for interfacing the enable signal to the RO.

- Regardless of RO length or feature size, the output frequency becomes less sensitive as supply voltage increases, eventually decreasing at higher supply voltages. The RO must operate in the low-voltage, high-sensitivity region to reduce error.

3.3.3 System Overview

Figure 3.2 shows the high-level organization of *Failure Sentinels*. The voltage divider sets the operating range for the RO, allowing me to tune the RO to operate in the most-sensitive voltage region. The level shifter makes the output signal from the RO compatible with the voltage level used by digital logic, reducing power consumption and ensuring reliable operation. The enable signal drives both an input to the NAND gate closing the RO loop and an N-type MOS device (NMOS) at the bottom of the voltage divider, allowing the designer

to change the duty cycle of the RO, reducing dynamic power consumption. Breaking the RO chain with an enable sets each gate to a known state before it begins oscillating to prevent higher harmonic output frequencies [10]. Finally, the counter makes the output of the RO available to the rest of the system in the form of an edge count accumulated during the sampling period. Software maps the resulting counter values to supply voltage values using enrollment data stored in the NVM.

3.3.4 Choosing RO Length

Per Equation 3.1, *Failure Sentinels*'s voltage sensitivity scales proportionally to $1/n$ where n is the length of the RO—a given change in supply voltage produces a corresponding frequency change that is larger in shorter ring oscillators because a smaller n magnifies the impact of a change in the gate delay τ_d on the oscillation frequency. For ROs implemented in the same technology a given voltage change produces the same *proportional* frequency change regardless of the number of RO stages, but *Failure Sentinels* measures the *absolute* change in frequency. A higher change in frequency requires a shorter enable period to detect; a shorter enable period allows *Failure Sentinels* to run either at a lower duty cycle (consuming less power, because the ring spends less time enabled) or at a higher sampling rate. I distinguish between the *enable period*—the amount of time the RO is powered to produce a single sample—and the *sample period*—the time between distinct samples—and discuss their impact on *Failure Sentinels* in more detail in Section 3.3.5.

Note that the dynamic power consumed by an RO is not dependent on its length, as only one inverter is ever switching at a time.[5] Increasing the size of the RO increases area overhead and static power; however, my evaluation in Section 3.4.2 shows that *Failure Sentinels* consumes

[5] *Failure Sentinels*'s total dynamic power is weakly dependent on RO size because the counter and level shifter power draw increase with frequency. However, the RO consumes the majority of *Failure Sentinels*'s power (Section 3.5.1).

negligible power and area compared to the rest of the microcontroller. However, an RO that oscillates too fast for a given sampling period will overflow the counter. Thus, the counter bit-width, sampling period, and RO length are interconnected, a design space that I explore in Section 3.5. From these constraints, I analyze the RO length primarily to the extent that it affects accuracy and power draw by setting a minimum duty cycle.

3.3.5 Duty Cycling

The accuracy of *Failure Sentinels* depends largely on the sampling rate and duty cycle $D = T_{en}/T_{sample} \leq 1$, where T_{en} is the time per sample during which *Failure Sentinels* is enabled and T_{sample} is the sampling period. A higher T_{en} enables *Failure Sentinels* to discriminate between finer RO frequency, and thus voltage, changes. The output of *Failure Sentinels* is in the form of the count $C = f_{ro} * T_{en}$; the edge-sensitive nature of the counter means that decimal values of C are effectively truncated. Therefore, the minimum detectable RO frequency change is $1/T_{en}$. The bit-width n of the counter limits the maximum value of C to $2^n - 1$; all possible values of $f_{ro} * T_{en}$ must be below this maximum to prevent counter overflow. Increasing T_{en} increases both accuracy and power consumption, which scales directly with duty cycle: given that low-resolution and low-frequency (relative to ADCs) measurements of the supply voltage are sufficient for current and near-future energy harvesters, operating *Failure Sentinels* with a low duty cycle enables significant power savings at little practical cost. A sufficiently low duty cycle also reduces counter size and its power. I evaluate the relationship between duty cycle, power, and accuracy in more detail in Section 3.5.1.

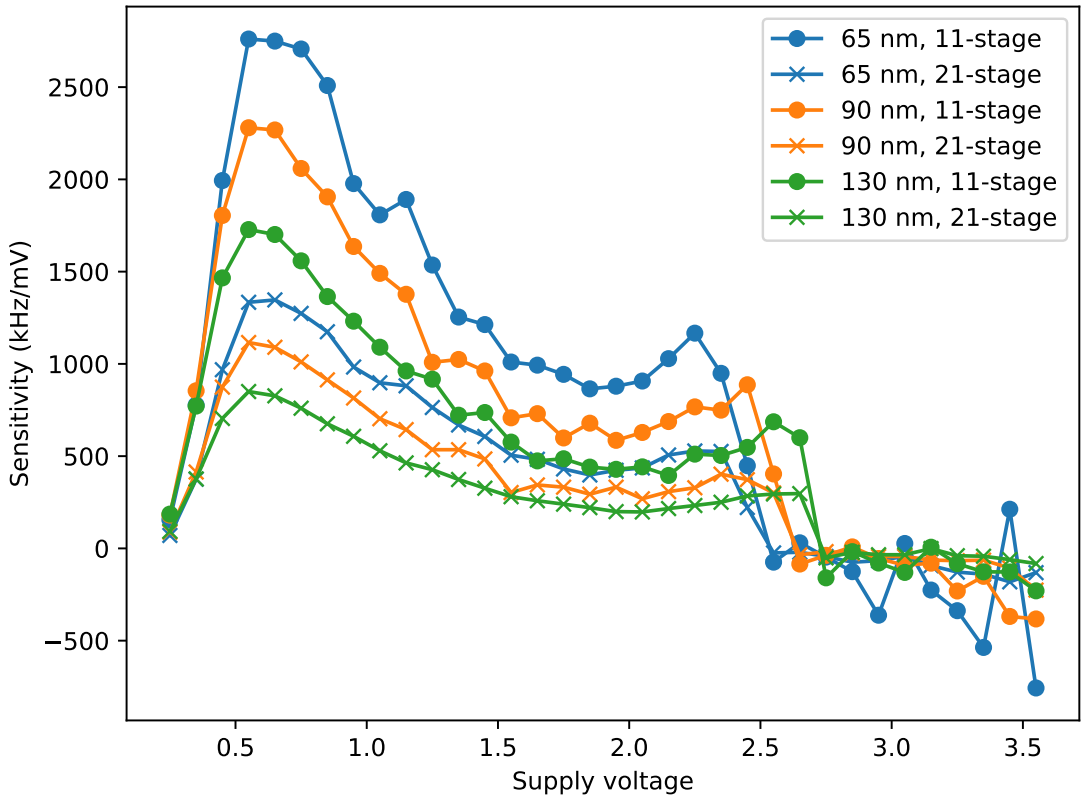


Figure 3.3: Frequency-voltage sensitivity for ROs across length and technology.

3.3.6 Maximizing Voltage Sensitivity

Inverter cell choice Past work provides a variety of options for the design of the inverter used to build the RO. Most ROs used in communications, clock generation, and other applications are current-starved [87]: the charge/discharge time of each inverter is limited by a voltage-controlled current source using a separate variable biasing voltage. An important property of the current-starved RO for these applications is that the current source *isolates the inverter from supply voltage noise*, minimizing uncontrollable variation and enabling the designer to produce a frequency output that is primarily a function of only the bias voltage. The crucial difference in *Failure Sentinels* is that the change in the supply voltage is

the quantity of interest. Instead, I maximize sensitivity to changes in the supply voltage by using the simplest inverter available consisting of single PMOS and NMOS transistors connected directly to the supply voltage and ground, respectively. This basic inverter design has additional benefits, as it reduces the total transistor count and is implementable using digital-only standard cell libraries.

RO operating voltage Figure 3.1 shows that the frequency-voltage curve for each RO is steepest at lower voltages, leveling off around 2.5 V and decreasing at higher voltages. The recommended operating voltage for microcontrollers used in recent energy harvesting work is 1.8V–3.6V [67, 73, 99]; for these platforms, connecting the ROs directly to the supply voltage means that they would operate primarily in the less-sensitive region. Furthermore, the voltage-frequency relationship at high voltages is non-monotonic—complicating the mapping in software from RO frequency back to supply voltage. To maximize *Failure Sentinels*'s sensitivity to supply voltage changes and keep the voltage-frequency relationship monotonic, the RO operates at a reduced voltage produced by the transistor-based voltage divider shown in Figure 3.2. This has the added benefit of reducing power consumption. The trade-off is that reducing the RO operating voltage adds complexity to the design because the output must be integrated back into the digital system, which Section 3.3.7 explores in detail.

Assuming a standard n-well process, which exposes the bulk connection of PMOS transistors,[6] the voltage divider consists of diode-connected PMOS devices with the bulk terminal connected to the source to ensure that each device is biased identically even as the gate voltage with respect to ground of successive transistors drops. V_{gs} for each individual transistor is small, limiting the current draw of the divider. This design parallels a resistive voltage divider, but the use of transistors makes it applicable to wholly-digital ICs. The drawback of

[6] In a p-well process, the voltage divider consists of NMOS devices and works equally well.

the transistor version is that they become non-linear at extremely low and high voltages, but these voltages are well beyond the specified operational voltage range of microcontrollers.

The RO draws power from a node n PMOS transistors away from ground; in a divider consisting of m diode-connected devices, the RO supply voltage is $V_{ro} = V_{supply} * n/m$. The best voltage division ratio depends on the sensitivity curve of the RO, shown in Figure 3.3 for several RO lengths and technologies. Reducing the voltage seen by the RO tends to increase sensitivity; however, it also reduces the voltage change seen by the RO for a corresponding change at the supply rail. I define the *sensitivity gain* G using Equation 3.2.

$$G = \frac{\overline{S_{new}}}{\overline{S_{old}}} * \frac{n}{m} \tag{3.2}$$

The $\overline{S_{new}}$ and $\overline{S_{old}}$ terms reflect the average sensitivity in the new and old operating regions, respectively. The best division ratio is the one that maximizes G and is technology-dependent. I find that the best ratios implementable in a small number of transistors are $n/m = 1/3$ or $1/2$; each of these division ratios produces a sensitivity gain of $G \approx 2$. Between division ratios that produce the same sensitivity gain for a given process, the smallest one reduces power consumption by reducing the operating voltage of the RO. Thus, I select $n/m = 1/3$.

Assuming the transistors are well-matched, the unloaded output of the voltage divider is a reliable fractional value of the supply voltage. However, enabling the RO to draw power from the voltage divider reduces the effective resistance between the divider output and ground—resulting in a voltage drop and a V_{ro} below the nominal value. I compensate for this voltage drop by increasing the width of certain transistors. I widen the transistors between the voltage divider output and V_{supply} to increase current delivered to the RO and reduce the magnitude of the voltage drop. Appropriate transistor sizing reduces, but does

not eliminate, the voltage drop seen at V_{ro} because the proportional error depends on the value of V_{supply}. However, the enrollment process described in Section 3.3.8 accounts for any remaining error, because the voltage offset is predictable at each supply voltage.

3.3.7 Logic Interfacing

Digital CMOS gates depend on well-defined input signals to achieve high speed and low power: an input must either be close to the supply voltage or close to ground to fully and rapidly switch the component transistors. Operating the RO at a fraction of the system's supply voltage increases sensitivity and decreases power consumption, but means that applying the logical 1 output of the RO directly to the counter input (operating at the normal supply voltage) violates this fundamental assumption of digital CMOS logic. The low-voltage RO logical 1 at best leaves little margin for noise and at worst is consistently below the core's logical 1 level, producing a signal that is unrecognizable to the core. Even if the RO output is reliably interpreted as a logical 1 by the core, driving CMOS gates with a low-voltage 1 increases power consumption due to ohmic losses from partially-on transistors and current in the low-impedance path to ground. I resolve the voltage difference using the level shifter shown in Figure 3.2, a self-reinforcing circuit leveraging the common ground of both voltage domains to boost the RO output voltage to the core voltage.

Ultimately, software needs to measure the frequency of the RO to make decisions based on supply voltage. I measure the output of the RO using a digital counter configured as shown in Figure 3.2 to increment on every positive edge of the level shifter output. The measurement is sent to a digital comparator for interrupt generation and made available to software by the addition of an instruction to the microcontroller's ISA, making energy availability a first-class hardware abstraction.

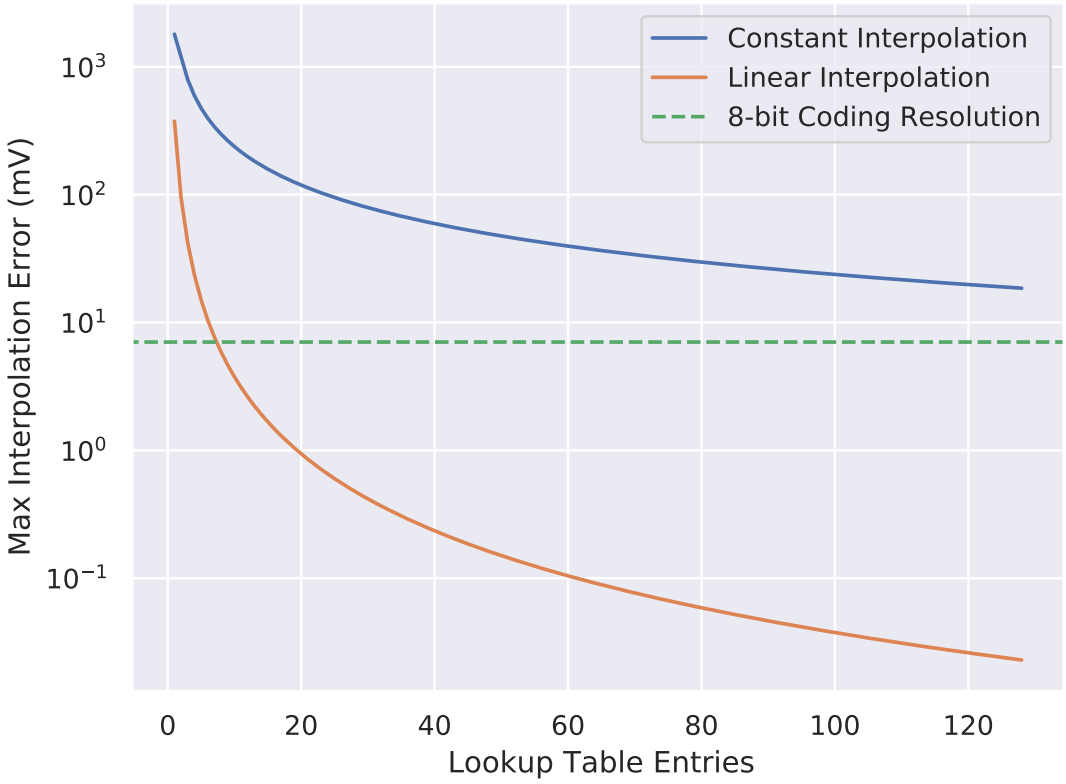


Figure 3.4: Maximum interpolation error for a 21-stage RO in 130nm. The dashed line indicates minimum error possible using 8-bit calibration table entries.

3.3.8 Voltage-Frequency Memoization

The counter maps RO frequency to a count value; the final step is mapping the count value to supply voltage. While the slope of the frequency-voltage relationship is predictable across all ROs, manufacturing-time process variation mean that identical ROs on different chips produce different frequencies under the same conditions. Microcontroller manufacturers already address process variation in sensitive circuits such as clock oscillators and sensors [60, 73, 99] using a post-manufacture enrollment step, testing the device with known inputs and writing device-specific calibration data to the Flash/ROM before deployment. I extend this enrollment process to increase *Failure Sentinels*'s precision by recording the RO frequency

using several known supply voltages. [7] Once deployed, software uses these calibration values to determine supply voltage with reduced error.

The choice of both what and how much data to store is important. In general, designers can increase run-time performance by increasing memory consumption and enrollment effort. I identify and evaluate several enrollment strategies that occupy different points in that trade space:

- **Full enrollment:** A simple but impractical solution is to store a voltage value for every possible *Failure Sentinels* output; this maximizes accuracy (the voltage-count curve is fully characterized and stored) and speed (mapping a count to a voltage is a simple indexing operation). However, it also maximizes memory overhead and enrollment effort for each device.

- **Piecewise-constant interpolation:** Instead of storing every possible counter output, I can trade accuracy for memory overhead by reducing the number of data points stored in NVM. When the counter produces a value not stored in the lookup table, *Failure Sentinels* pessimistically assumes the supply voltage is at whatever level is associated with the closest stored count value below the measured value. Designers can tune *Failure Sentinels*'s accuracy by changing the number of stored data points, while a runtime count-voltage conversion in this case is slightly slower than with a full table (requiring a comparison followed by indexing).

- **Piecewise-linear interpolation:** Piecewise-linear interpolation enables the same accuracy-memory tradeoff as the piecewise-constant design but instead calculates a linear interpolation between the nearest two points when a count value is not stored.

[7] On devices with ADCs, an alternative to manufacture-time enrollment is a one-time characterization of the RO frequency-supply voltage relationship using the ADC for ground truth.

This increases accuracy for the same memory footprint at the cost of increased runtime overhead evaluating the interpolation function.

- **Polynomial interpolation:** To minimize memory overhead, the enrollment system can characterize *Failure Sentinels* at a few supply voltage points and place coefficients for an arbitrary-degree polynomial regression function in the device's memory. This makes space overhead negligible at the cost of runtime performance—evaluating the polynomial function requires numerous floating-point multiplication operations, which can be both time- and energy-intensive on typical energy harvesting hardware.

I explore the piecewise-constant and piecewise-linear interpolation designs in more detail because they are the most flexible and best suited to the performance and NVM constraints of current energy harvesters. For a continuous function $f(x)$ with lower and upper bounds a and b, respectively, Equations 3.3 and 3.4 describe the respective maximum error for piecewise-linear and piecewise-constant interpolation [145].

$$E_{const} \leq h * \max_{x \in [a,b]} \left| \frac{df(x)}{dx} \right| \tag{3.3}$$

$$E_{lin} \leq \frac{h^2}{8} * \max_{x \in [a,b]} \left| \frac{d^2 f(x)}{dx^2} \right| \tag{3.4}$$

$f(x)$ is the mapping from frequency to voltage for a given RO, the inverse of the relationship shown in Figure 3.1. h is the distance between known frequency datapoints and decreases with higher NVM consumption; for the frequency-voltage transfer function with minimum frequency L, maximum frequency H, and c evenly-spaced datapoints[8], $h = (H - L)/c$.

Figure 3.4 shows the maximum error introduced by both types of interpolation as a function of NVM overhead, assuming that each voltage entry in the table is stored in a single byte.

[8]One way to increase interpolation accuracy is to locally reduce h by taking more data points in areas where $\left|\frac{df(x)}{dx}\right|$ or $\left|\frac{d^2 f(x)}{dx^2}\right|$ are highest, but for simplicity I use evenly spaced points.

By operating the RO at a low voltage using the divider described in Section 3.3.6, I maximize the linearity of the voltage-frequency transfer function and enable highly accurate interpolation with a relatively small NVM footprint. Linear interpolation scales better than constant interpolation with increasing NVM overhead, but both eventually achieve diminishing returns as other sources of error such as temperature begin to dominate *Failure Sentinels*'s total error. The precision of the recorded data points also limits interpolation accuracy, as shown in Figure 3.4: assuming a 1.8 V supply range, interpolating between 8-bit values cannot reduce the total error below $\frac{1.8V}{2^8} \approx 7mV$.

3.4 *Failure Sentinels* Implementation

I evaluate *Failure Sentinels* using two implementations, each targeting different aspects of the design: (1) a SPICE implementation and (2) a FPGA implementation. I use SPICE to drive my design space exploration and evaluate the effects of supply voltage, feature size, and the analog circuit components on *Failure Sentinels*'s performance. To explore the effects of run time variation such as temperature and to demonstrate *Failure Sentinels* on real hardware, I integrate *Failure Sentinels* into a RISC-V processor running on a FPGA.

3.4.1 SPICE Modeling

I model *Failure Sentinels* using LTspice [35] to explore its behavior at a wide variety of supply voltages across different feature sizes. This enables me to practically explore *Failure Sentinels*'s design space. To match deployed and near-future real-world energy harvesting microcontrollers, I implement each RO using the 130nm, 90nm, and 65nm process Predictive Technology Model (PTM) SPICE cards [17]. I also include the provided parasitic resistance

and capacitance estimates for local interconnects in those technologies between components. These SPICE simulations also offer insight into the effects of the analog circuitry (the voltage divider and level shifters), which is not available on the FPGA. Finally, SPICE includes power consumption information for each component of the design—enabling a direct comparison between *Failure Sentinels* and currently available alternatives such as ADCs.

3.4.2 FPGA Implementation

While the SPICE PTM models make it possible to perform a design-space exploration across process nodes and voltages, they do not accurately model the effects of thermal variation [88] and do not capture the ability to incorporate *Failure Sentinels* into a full system. To validate the SPICE-based design space exploration, understand temperature's impact on *Failure Sentinels*, and to show how architects can add *Failure Sentinels* to an existing System-on-Chip (SoC) to make energy availability a first-class hardware abstraction, I implement *Failure Sentinels* inside a RISC-V RocketChip SoC [5] on top of a Xilinx Artix-7 FPGA [141]. On top of this SoC, I run software that communicates with *Failure Sentinels* via two instructions added to the ISA: (1) an instruction that stores a 64-bit value representing the available energy to a user-specified destination register and (2) an instruction that the library-level recovery routine uses to enable *Failure Sentinels* as well as set the energy interrupt threshold. Similar to previous work that requires ADC support [134], I link unmodified software against a library-level interrupt handler that saves software state as a checkpoint when *Failure Sentinels*'s interrupt fires.

Following the design goals of the ideal voltage monitor, Table 3.2 shows that adding *Failure Sentinels* to an existing SoC is low cost: *Failure Sentinels* maintains the maximum frequency of the SoC, minimally increases area, and, in accordance with the SPICE evaluation (Sec-

	area (LUTs)	timing (MHz)	power (W)
Base SoC	53664	30	1.105
+*Failure Sentinels*	53687 (+0.04%)	30 (+0.0%)	1.104 (-.09%)

Table 3.2: *Failure Sentinels* hardware overheads when added to a RISC-V SoC [5]. Note that power is within the noise margin of the tools.

tion 3.5.1), steals very little energy from software computation. The implemented variant of *Failure Sentinels* has a 21-stage RO and an 8-bit counter. The fixed nature of the FPGA fabric precludes implementing the transistor-based voltage divider and level shifter, but those minimally contribute to *Failure Sentinels* area and removing them actually increases *Failure Sentinels*'s power.

3.5 Evaluation

I take a two-level approach in evaluating *Failure Sentinels* in order to support comparison to currently available alternatives. I first perform a comprehensive evaluation of the *Failure Sentinels* design space using SPICE simulations to explore the trade space between different design parameters and their effect on *Failure Sentinels*. Then I demonstrate *Failure Sentinels* on real hardware and evaluate the impact of thermal variation by implementing *Failure Sentinels* as part of a RISC-V System-on-Chip using a FPGA. The results of this evaluation answer the following questions:

1. How does building *Failure Sentinels* to satisfy certain design constraints impact its performance in other areas?

2. How do typical sources of run-time variation such as temperature affect *Failure Sentinels*?

3. How well does *Failure Sentinels* fit the needs of energy harvesting applications com-

Design			Performance		
Parameter	Min.	Max.	Parameter	Min.	Max.
RO Length	3	73	Mean Current (μA)	0	5
F_s (kHz)	1	10	F_s (kHz)	1	10
Counter Size (bits)	1	16	Granularity (mV)	0	50
Enable Time	1 μs	1 ms	NVM Overhead (B)	0	128
NVM Entries	1	128	Transistor Count	0	1000
Entry Size (bits)	1	16			

Table 3.3: *Failure Sentinels* design and performance parameters bounding my exploration.

pared to existing alternatives?

3.5.1 *Failure Sentinels* Design Space

Failure Sentinels is designed for flexibility; each application places unique demands on the system in terms of power consumption, resolution, and other parameters. Rather than hand-design and evaluate one *Failure Sentinels* implementation for a given deployment, I explore the *Failure Sentinels* design space by finding a set of Pareto-optimal implementations within performance constraints suitable for a range of energy-harvester deployments. I model *Failure Sentinels* design as an optimization problem mapping six design parameters to five performance parameters as shown in Table 3.3. I set the design parameter bounds to ease integration with today's energy harvesters—for example, I limit the counter size to 16 bits to improve performance on the 16-bit architectures common to currently deployed energy harvesters [67]. Similarly, the 1 μs minimum enable-time stems from the minimum period of the fastest (1 MHz) clock available on similar systems without increasing current consumption [67].

I explore the resulting design space using Pymoo [13], a Python optimization library. I

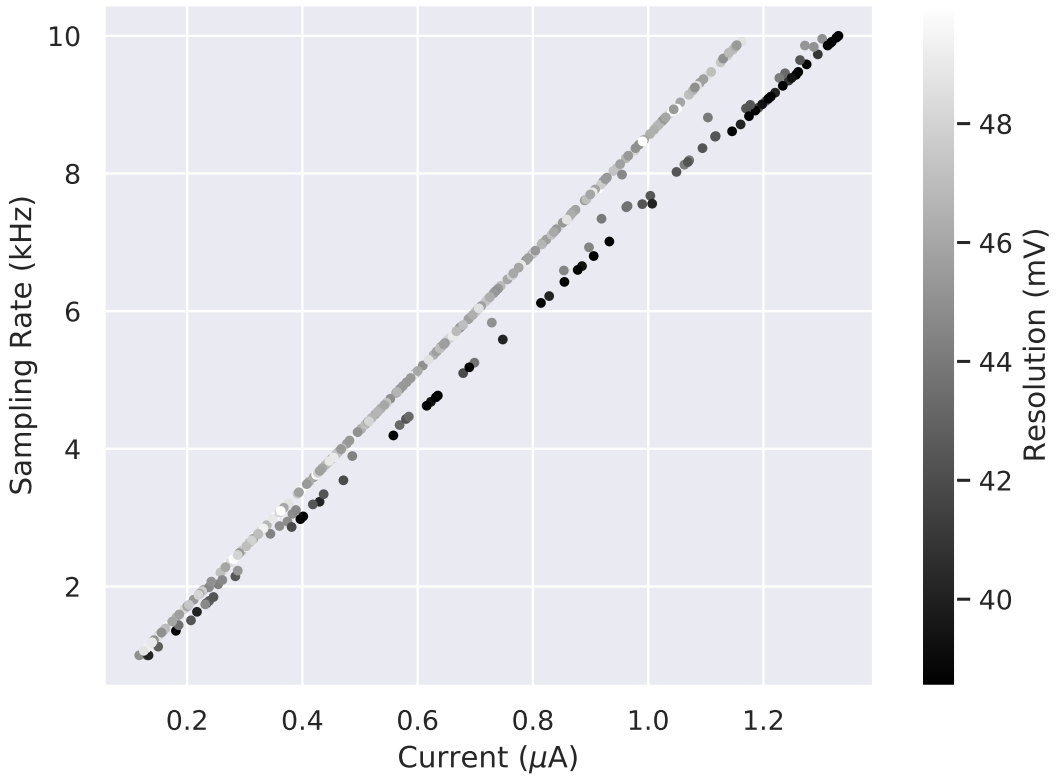


Figure 3.5: Objective space exploration for *Failure Sentinels* in 90nm.

implement *Failure Sentinels* in LTspice [35] at several design points spanning the limits shown in Table 3.3 in each of the process nodes described in Section 3.3.2 and evaluate each implementation across the typical 1.8V–3.6V operating range, in 100 mV steps. The results from these simulations form the basis for an analytical model of *Failure Sentinels*'s performance that I use to drive the optimization function. However, the SPICE simulations do not fully reflect several design choices beyond the core *Failure Sentinels* hardware. In order to accurately represent a real *Failure Sentinels* implementation, I augment the analytical model with several elements beyond the SPICE results:

- I model the number and size of NVM lookup table entries to fulfill the NVM overhead constraint, and factor in their effect on *Failure Sentinels*'s accuracy using the piecewise-

linear interpolation strategy described in Section 3.3.8.

- I include temperature as another limiting factor on *Failure Sentinels*'s accuracy and assume a maximum temperature-induced RO frequency deviation of 2% according to the FPGA-based experiments in Section 3.5.3.

- I add a rejection filter to ensure the resulting configuration is realizable and correct (e.g., the RO is never enabled long enough to overflow the counter).

In general, the resulting Pareto frontier is five-dimensional in each of the performance parameters. Given that NVM and die space consumption have minimal impact on operational performance (as long as the code/calibration data still fit in the NVM and *Failure Sentinels* fits on the chip), I expect typical *Failure Sentinels* deployments to be constrained primarily by sampling frequency, power, or resolution. For visualization, I reduce the dimensionality of the frontier by only plotting the first three performance parameters in Table 3.3 with the knowledge that each solution satisfies the limit on NVM overhead and transistor count. Figure 3.5 shows the trade space for *Failure Sentinels* in 90nm technology; each point denotes the performance of a different Pareto-optimal configuration.

Failure Sentinels's flexibility enables designers to precisely tune performance to the needs of their specific application by compromising on each of the three performance parameters shown. Sampling frequency is the primary driver of current consumption in the design space I explore because temperature variations rather than current consumption set the limit on *Failure Sentinels*'s resolution (see Section 3.5.3). Figure 3.5 shows the current-resolution-sample rate trade space accounting for the temperature-induced limit; reducing sampling granularity (e.g., from 38 mV to 48 mV) reduces mean current consumption by 14% at the highest sampling rate of 10 kHz. This tradeoff becomes more favorable at both lower sampling rates and smaller process nodes; at a 10 kHz sample rate, there is an 8% difference

in current consumption between the finest (27 mV) and coarsest (50 mV) granularities for the 65nm implementation of *Failure Sentinels*. For all configurations and all technologies, the RO represents over 90% of *Failure Sentinels*'s total current consumption. Given that RO length only affects the power consumption of the supporting components (see Section 3.3.4), this indicates that duty cycle—a function of sampling frequency and enable-time—is the primary determinant of current consumption.

In each case, *Failure Sentinels* dramatically reduces the power budget required for voltage monitoring hardware and provides the resolution and speed performance needed to enable the most sophisticated energy harvesting runtimes available. Assuming a 1.8V dynamic range, Figure 3.6 shows that *Failure Sentinels* offers between 5 and 6 bits of resolution depending on feature size while consuming, in total, less than 1 μA—enabling sophisticated energy harvesting systems with negligible power overhead. *Failure Sentinels* eliminates between 59%–77% of the system's energy overhead while enabling the same power-based intermittent runtimes as an ADC. Even compared to single-bit analog comparators supporting a simple just-in-time checkpointing system, *Failure Sentinels* increases energy available to software by 24%–45%.

3.5.2 *Failure Sentinels* Scales with Technology

Failure Sentinels's fully-digital design enables it to scale down with the rest of the processor to maximize performance. First, smaller process nodes enable lower-power operation, all other parameters being equal: switching from 130nm to the 90nm process, I observe a 14% reduction in power consumption—with a similar reduction from 90nm to 65nm. Second, transistor delays in smaller technologies are also more sensitive to supply voltage variations [3]: my experiments show that RO frequency in the 65nm process is approximately 2%

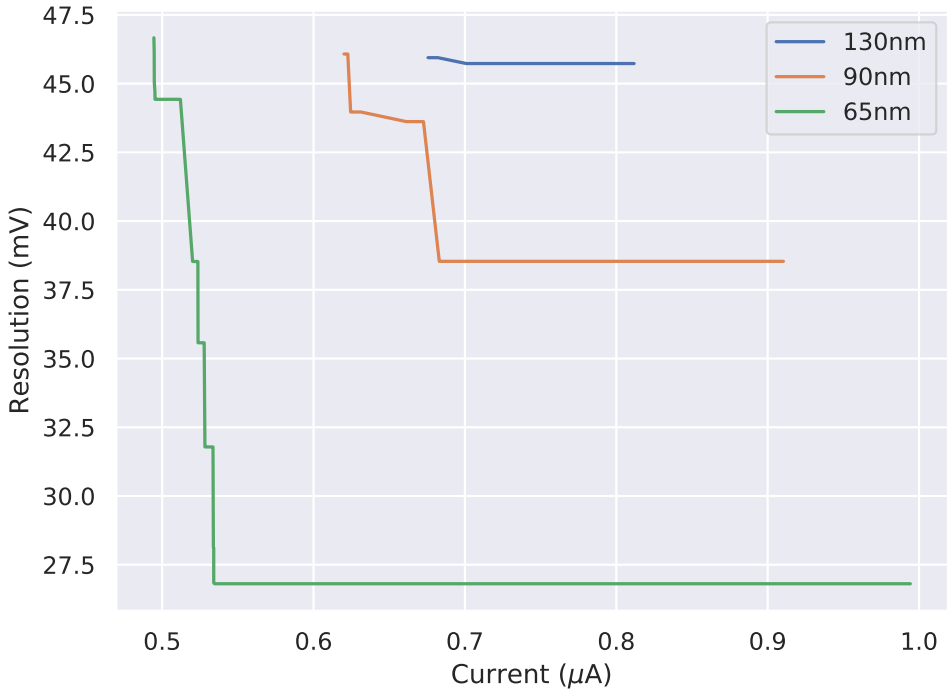


Figure 3.6: Pareto-optimal configurations for each technology with $F_s = 5$ kHz.

more sensitive to supply voltage than in the 90nm process and 14% more sensitive than the 130nm process.

To predict *Failure Sentinels*'s performance trends from current energy harvester feature sizes to near-future ones, I explore the trade space in each technology discussed in Section 3.3.2 around the $F_s = 5kHz$ operating point. Figure 3.6 shows that at the same sample rate, smaller feature sizes enable both lower current and finer resolution operation for *Failure Sentinels*. These results indicate that *Failure Sentinels* effectively removes the power/size bottleneck of highly-analog circuits and enables energy harvesters to better leverage the advantages of transistor scaling.

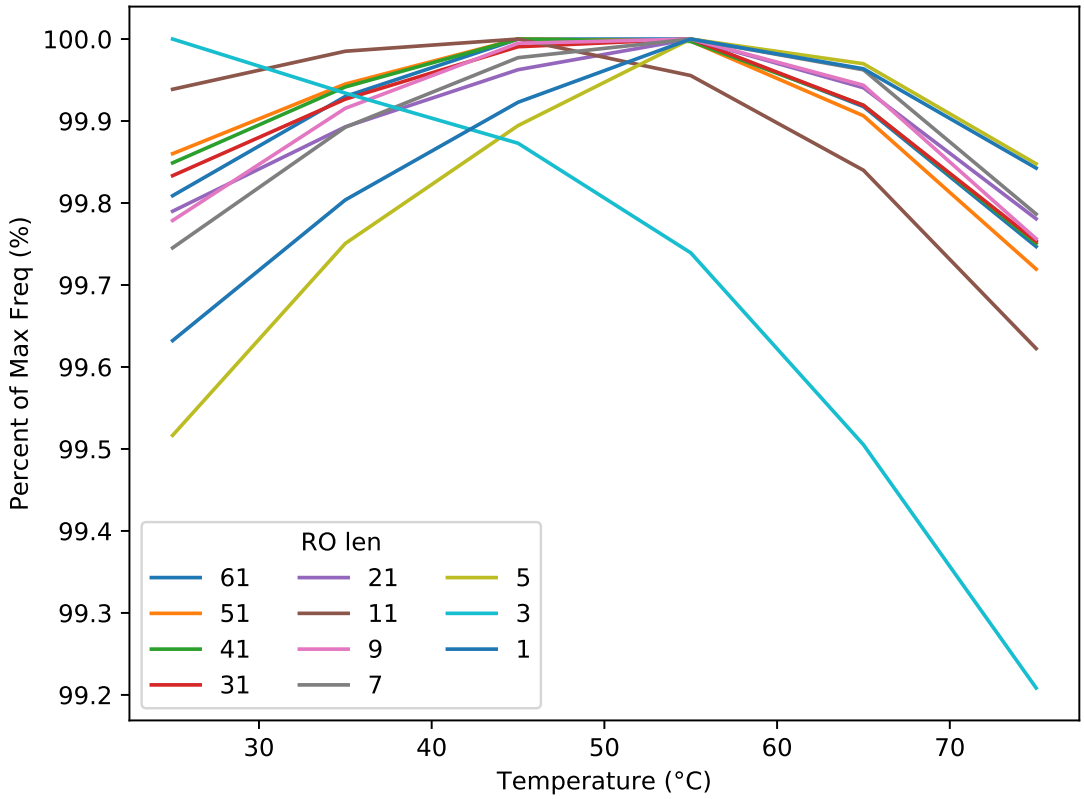


Figure 3.7: RO frequency variation with temperature on Xilinx Virtex-7 FPGA.

3.5.3 Temperature Variation

To build a picture of how operational conditions affect *Failure Sentinels*'s performance, I need to examine the impact of thermal fluctuations on RO frequency. Environments where the temperature changes dramatically have the potential to reduce the system's accuracy because *Failure Sentinels* misinterprets temperature-induced frequency changes as voltage changes. Temperature affects digital circuits by changing gate delay,[9] which in turn affects the frequency of the RO. For *Failure Sentinels*, the circuitry supporting the RO is largely temperature-independent: the voltage divider depends only on the *relative* differences be-

[9]Temperature affects digital gates by reducing carrier mobility (increasing propagation delay) and reducing threshold voltage (decreasing propagation delay) [111]; each of these effects dominates in different circumstances.

tween each device, and temperature affects each device equally. Temperature changes the maximum operating frequency of the level shifter by changing transistor drive strength, but my results indicate that RO frequency is always well below the level shifter's maximum.

Given that the RO is the primary factor in *Failure Sentinels*'s temperature sensitivity, I implement a range of RO sizes on a Xilinx Artix-7 FPGA. Using a TestEquity 123H temperature chamber [130], I vary the operating environment from room temperature (25C) up to 75C to encompass the typical operating range of energy harvesting devices. I let the device stabilize at the target temperature for an hour before measurements. For each configuration and temperature, I report the average of 1000 RO count measurements. Figure 3.7 illustrates the relative change in frequency across temperatures for all implemented ROs. I consider the temperature-induced error to be the largest frequency change between any two frequencies. Much like voltage-induced changes, temperature-induced changes are similar across RO sizes, because only one gate switches at a time. I double the 1% maximum effect shown in Figure 3.7 to create a conservative, worst-case 2% thermal error. This error fits past work measuring RO and delay line sensitivity to temperature [18, 132].

This thermal error serves as an upper bound on *Failure Sentinels*'s resolution. My analytical model indicates that temperature-induced frequency changes approximately double *Failure Sentinels*'s error, motivating future work reducing *Failure Sentinels*'s temperature sensitivity. One potential approach is to increase the interconnect length between each inverter; because transistors are significantly more sensitive than interconnects to temperature changes [149], increasing the RO delay due to interconnect reduces *Failure Sentinels*'s overall temperature sensitivity. Because longer interconnects may affect *Failure Sentinels*'s voltage sensitivity, I leave a detailed exploration of this area for future work.

Monitor	Sys. Current (μA)	Res. (mV)	F_s (kHz)	V_{ckpt} (V)
Ideal	112.3	Infinite	Infinite	1.82
FS (LP)	112.5	50	1	1.87
FS (HP)	113.6	38	10	1.86
Comparator	147.3	30	3030*	1.86
ADC	377.3	0.293	200	1.87

Table 3.4: Voltage monitors I evaluate within a full system. FS (LP) uses a 67-stage RO with a 49-entry LUT of 8-bit values, while FS (HP) uses a 7-stage RO with a 52-entry LUT of 10-bit values. Both versions uses a 6-bit counter and a 1 μs enable time. *Comparator response time is 330 ns.

3.5.4 System-level Impact

In order to determine the impact *Failure Sentinels* has on a typical intermittent system in an energy-scarce environment, I compare it to existing solutions in the context of a simulated solar-powered energy harvester using the EnHANTs irradiance dataset [40] for a pedestrian in New York City at night. Similar to past energy harvesting architectural exploration [32], I use this simulation framework to explore the effect of different *Failure Sentinels* configurations on system performance, measured in time available for executing application code.

Evaluation Parameters I compare *Failure Sentinels* to the analog alternatives on the MSP430FR5969 [67] detailed in Table 3.1. I evaluate one 90nm *Failure Sentinels* implementation optimized for high performance (HP) and one for low power (LP), taken from opposite extremes of the objective space exploration in Figure 3.5. I model a typical energy harvesting sensor using a 5 cm^2, 15% efficient solar panel to charge a 47 μF storage capacitor; when the capacitor reaches the enable voltage of 3.5V, the microcontroller and a peripheral accelerometer [34] begin consuming power. Both devices operate until the supply capacitor reaches a checkpoint voltage detailed below, at which point the microcontroller stops application code and stores a checkpoint in NVM. I model the current consumption of the microcontroller core, accelerometer, and voltage monitor when the device is executing,

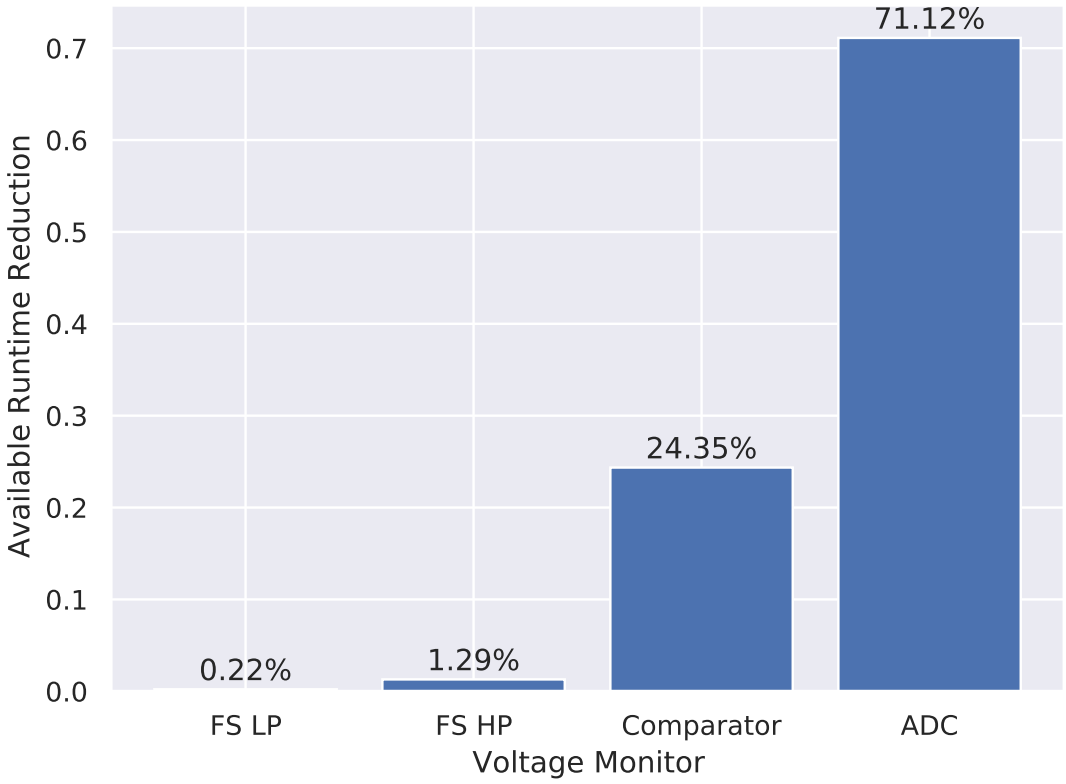


Figure 3.8: Reduction in available time to process application code, normalized to ideal monitoring.

---

and a leakage current of 0.5 μA at all times.

Checkpointing Mechanics I model the worst-case checkpoint behavior as writing all volatile data to non-volatile FRAM, which takes 8.192 ms at a clock frequency of 1 MHz on my microcontroller. This execution time combined with total current draw and supply capacitance sets the ideal minimum voltage at which the microcontroller has just enough energy to complete the checkpoint. However, the limited accuracy of each system prevents us from achieving this minimum voltage. I add the measurement resolution of each device to the theoretical minimum to ensure the checkpoint will always complete despite worst-case measurement error and show the final checkpoint voltage for each system in Table 3.4. The

similar checkpoint voltages across each monitor, despite dramatic differences in resolution, show how current monitors are over-optimized for resolution because the additional energy drawn from the capacitor is consumed by the monitor itself. Finally, I consider the effect of monitor sampling frequency (because capacitor voltage changes over the course of a sample). The effect of sampling frequency on accuracy is small for this scenario—2mV in the worst case using FS (LP)—showing that reducing sampling frequency is an effective way to reduce power consumption without sacrificing performance.

Performance Comparison Table 3.4 and Figure 3.8 illustrate the performance improvement of *Failure Sentinels* over analog-based alternatives for my checkpointing system. I normalize all runtime results to performance using the ideal voltage monitor, representing perfect sampling and zero overhead from monitoring hardware. Both implementations of *Failure Sentinels* achieve near-ideal runtime, compared to the 24% and 70% runtime penalties of the analog solutions—illustrating *Failure Sentinels*'s ability to maximize the time and energy available for application code.

Discussion While I evaluate *Failure Sentinels* here based on a typical batteryless sensor mote, different system-level design choices place different demands on the voltage monitoring hardware. Systems with smaller supply capacitors require a monitor with a higher sampling frequency because the supply capacitor discharges more per unit of time, but designers must balance higher sampling frequency with the corresponding current draw. Conversely, monitor resolution becomes more important as the size of the supply capacitor increases because the voltage offset represents increasingly more energy that could have otherwise been used for computation. Broadly, I expect small sensor motes to favor a low-current, low-resolution implementation of *Failure Sentinels* while platforms with comparatively large supply capacitors and active power draws (e.g., energy harvesting satellites [30]) benefit

more from a high-resolution implementation of *Failure Sentinels* when the additional energy extracted from the capacitor outweighs the increased draw of the monitor itself. Emerging energy-aware systems beyond checkpointing (Section 3.2.3) will further exercise the voltage monitoring trade space I explore, highlighting the value of *Failure Sentinels*'s flexibility.

3.6 Conclusion

Failure Sentinels leverages the voltage-dependent gate delay of CMOS devices to eliminate the need for ill-suited, high-power analog hardware to monitor available energy. I design *Failure Sentinels* to provide *just enough* performance using only the lowest power, most scalable hardware available to designers: the transistor. A focus of my design is identifying and operating at the sweet spot of the transistor delay and voltage relationship, where dynamic power is reduced and sensitivity is most linear. I incorporate *Failure Sentinels* into a RISC-V system-on-chip and provide a software-queriable register for energy availability, making energy availability a first-class abstraction of the hardware. My evaluation shows that *Failure Sentinels* reduces power consumption by between 59% and 77% compared to conventional analog-to-digital converters—without compromising system performance. Replacing one-bit voltage comparators with *Failure Sentinels* reduces power consumption by between 24% and 45%, while also enabling a myriad of new power-responsive techniques to improve whole-system efficiency and performance.

These results show that enabling sophisticated intermittent computing support on even the smallest, lowest-power devices is possible without a substantial increase in price or power consumption. *Failure Sentinels*'s low power and space overhead implies that it could even be integrated onto devices smaller than microcontrollers to support sophisticated intermittently operated peripherals, expanding the horizons for future energy harvesting designs and

deployments.

Chapter 4

Energy-Adaptive Buffering in Batteryless Systems

4.1 Introduction

Ever-shrinking computing and sensing hardware has pushed mobile Internet-of-Things (IoT) type devices beyond the limits of the batteries powering them. A typical low cost/power microcontroller [62] drains a 1 cm^3 battery nearly 14x its size in just over 8 weeks of active operation [109], rendering the system useless without a potentially costly replacement effort. Cost, maintenance, and safety concerns make batteries further incompatible with massive-scale (one million devices per square kilometer [12]) and deeply-deployed (infrastructure [2], healthcare [105]) applications. IoT engineers are turning to batteryless energy harvesting platforms to power low-cost, perpetual systems capable of driving a ubiquitous computing revolution. Increasingly efficient energy harvesting circuits enable batteryless systems across a range of IoT use cases including feature-rich batteryless temperature sensors 500x smaller than a grain of rice [138] and batteryless flow-meters [44] supporting deep-sea drilling or geothermal plants for decades without maintenance.

The energy harvesting design model both enables new deployments previously limited by batteries and places new demands on system developers. Harvested energy is highly unreliable: sensitive environmental factors such as shadows over a photovoltaic cell or shifts in

the orientation of a rectenna produce rapid, outsized changes in the energy scavenged by the harvester. Energy harvesters mitigate this unreliability by charging an energy buffer to a given enable voltage, which the system periodically discharges to supply a useful quantum of work despite potential power loss.

Buffer capacity is a key design element of any batteryless system. Past work [24] explores the tradeoff between buffer sizes: small buffers are highly *reactive*—charging rapidly and quickly enabling the system to address time-sensitive events—but sacrifice *longevity* because they rapidly discharge during operation, guaranteeing only a short burst of uninterrupted operation. Large buffers store more energy at a given voltage, improving longevity by supporting a longer or more energy-intensive burst of operation at the cost of reactivity because they require more energy to enable the system at all. Matching buffer size to projected energy demand is critical to ensuring the system is both reactive enough to address incoming events/deadlines (e.g., periodic sensor readings) and long-lived enough to support uninterruptible operations (e.g., radio transmissions). Designers choose the minimum size necessary to power all atomic operations on the device, maximizing reactivity given a required level of longevity.

In this work, I explore static energy buffer *efficiency* as a third metric for buffer performance and find that it varies dramatically with *net energy input* rather than simple energy demand. Small buffers reach capacity quickly if power input exceeds instantaneous demand—burning off hard-won energy as heat to prevent overvoltage. Large buffers capture all incoming power, but enable slowly and lose more harvested energy to leakage below the minimum system voltage. The volatile nature of harvested power means that fixed-size buffers experience *both* problems over the course of their deployment, discharging energy during a power surplus and losing large portions of energy to leakage during a deficit.

To make the most of incoming energy in all circumstances, I propose *REACT*[1]: a dynamic energy buffering system that varies its capacitance following changes in *net* power. *REACT* maximizes system responsiveness and efficiency using a small static buffer capacitor, quickly enabling the system to monitor events or do other low-power work under even low input power. If input power rises beyond the current system demand and the static buffer approaches capacity, *REACT* connects additional capacitor banks to absorb the surplus, yielding the capacity benefits of large buffers without the responsiveness penalty. When net power is negative, these capacitors hold the system voltage up and extend operation beyond what is possible using the small static capacitor.

While expanding buffer size to follow net power input ensures the system can capture *all* incoming energy, increasing capacitance also increases the amount of unusable charge stored on the capacitor banks—charge which could power useful work if it were on a smaller capacitor and therefore available at a higher voltage. As supply voltage falls and approaches a minimum threshold, *REACT reclaims* this otherwise-unavailable energy by reconfiguring capacitor banks into series, shifting the same amount of charge onto a smaller equivalent capacitance in order to boost the voltage at the buffer output and ensure the system continues operating for as long as possible. *REACT* effectively eliminates the design tradeoff between reactivity and capacity by tuning buffer size within an arbitrarily large capacitance range, only adding capacity when the buffer is already near full. *REACT*'s charge reclamation techniques maximize efficiency by moving charge out of large capacitor banks onto smaller ones when net input power is negative, ensuring all energy is available for useful work.

I integrate a hardware prototype of *REACT* into a full energy harvesting platform to evaluate it against previous work, operating under different input power conditions and with different power consumption profiles. My results indicate that *REACT* provides the "best of both

[1] **R**econfigurable, **E**nergy-**A**daptive **C**apaci**T**ors

worlds" of both small- and large-buffer systems, rapidly reaching the operational voltage under any power conditions while also expanding as necessary to capture all available energy and provide software longevity guarantees as needed. Maximizing buffer capacity and reclaiming charge using *REACT*'s reconfigurable capacitor banks eliminates the efficiency penalties associated with both small and large static capacitor buffers, increasing the portion of harvested energy used for application code by an average 39% over an equally-reactive static buffer and 19% over an equal-capacity one. I also compare *REACT* to recent work exploring dynamic-capacitance batteryless systems [144] using a fully-interconnected capacitor architecture; my evaluation on real-world energy harvesting traces shows that prior work targeting the reactivity-longevity tradeoff *underperforms* baseline static capacitance systems as a result of lossy switching between capacitance modes. *REACT* improves performance by an average 26% over the state of the art owing to its efficient bank-based charge management structure. This work makes the following technical contributions:

- I evaluate the power dynamics of common batteryless systems in real deployments and explore how common-case volatility introduces significant energy waste in static or demand-driven buffers (§ 4.2).

- I design *REACT*, a dynamic buffer system which varies its capacitance according to system needs driven by net input power (§ 4.3). *REACT*'s configurable arrays combine the responsiveness of small buffers with the longevity and capacity of large ones, enables energy reclamation to make the most of harvested power, and avoids the pitfalls of energy waste inherent in other dynamic capacitance designs (§4.3.3).

- I integrate *REACT* into a batteryless system and evaluate its effect on reactivity, longevity, and efficiency under a variety of power conditions and workloads (§4.5). My evaluation indicates that *REACT* eliminates the responsiveness-longevity tradeoff

inherent in static buffer design while increasing overall system efficiency compared to *any* static system.

- I publish my design and evaluation data for *REACT* and baseline systems in an open-source repository at `https://github.com/FoRTE-Research/REACT-Artifact` to enable further evaluation and integration of *REACT* into batteryless systems.

4.2 Background and Related Work

Scaling sensing, computation, and communication down to smart dust [83] dimensions requires harvesting power on-demand rather than packaging energy with each device using a battery. Many batteryless systems use photovoltaic [29, 138] or RF power [121], while other use-cases are better suited for sources such as vibration [127], fluid flow [44], or heat gradients [92]. Commonalities between ambient power sources have inspired researchers to develop general-purpose batteryless systems, regardless of the actual power source: ambient power is *unpredictable*, *dynamic*, and often *scarce* relative to the active power consumption of the system.

Batteryless systems isolate sensing and actuation components from volatile power using a buffer capacitor. The harvester charges the capacitor to a pre-defined enable voltage, after which the system turns on and begins consuming power. Because many environmental sources cannot consistently power continuous execution, systems operate intermittently—draining the capacitor in short bursts of operation punctuated by long recharge periods. This general-purpose intermittent operation model has enabled researchers to abstract away the behavior of incoming power and focus on developing correct and efficient techniques for working under intermittent power [52, 91, 94, 97, 134, 137].

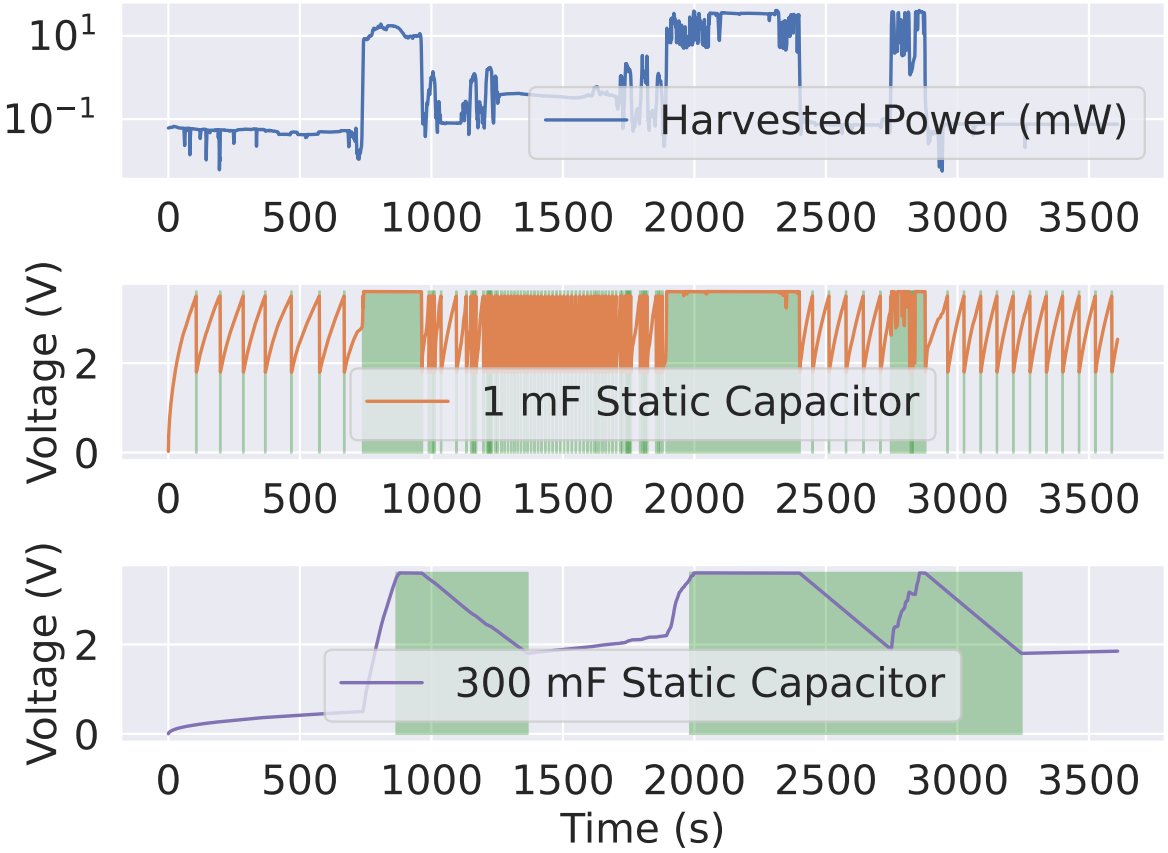


Figure 4.1: Static buffer operation on a simulated solar harvester. Highlighted blocks indicate the system is running.

4.2.1 Choosing Buffer Capacity

Buffer size determines system behavior in several important ways. Supercapacitors provide inexpensive and small-form-factor bulk capacitance [85], enabling designers to choose a capacitor according to performance rather than cost or size concerns. Two metrics motivate past work: *reactivity* refers to the system's ability to rapidly charge to its enable voltage and begin operation. High reactivity ensures a system is online to execute periodic tasks or address unpredictable input events. *Longevity* refers to the energy available for an

uninterrupted period of work with no additional power input; long-lived systems support high-power and long-running uninterruptible operations and reduce the overhead incurred with state recovery after a power loss.

Reactivity and Longevity:

A batteryless system's reactivity and longevity depend primarily on the charge and discharge rate of the buffer capacitor. I illustrate the tradeoff using a simulated solar harvester with a 22% efficient, 5 cm^2 panel, based on a pedestrian trace from the EnHANTs solar dataset [40]. The system runs from 3.6V down to 1.8V and draws 1.5 mA in active mode, representative of a typical deployment [89]. Figure 4.1 illustrates the reactivity-longevity tradeoff inherent in static buffer systems at two design extremes, using a 1 mF and 300 mF capacitor. The 1 mF system charges rapidly and is therefore highly reactive, reaching the enable voltage over $8x$ sooner than the 300 mF version. However, the smaller capacitor also discharges quickly—the mean length of an uninterrupted power cycle using the 1 mF capacitor is 10 seconds versus 880 seconds for the 300 mF capacitor, indicating the 300 mF system is far longer-lived once charged. The relative importance of reactivity and longevity depends on the use case, but often changes over time for a complex system—complicating design further.

Power Volatility and Energy Efficiency:

Buffer capacity is also a major driver of end-to-end energy *efficiency*: using the 300 mF capacitor the system is operational for 49% of the overall power trace, compared to only 27% for the 1 mF platform. This stems from the high volatility of incoming power—82% of the total energy input is collected during short-duration power spikes when harvested power rises above 10 mW, despite the system spending 77% of its time at input powers below 3

mW. A large buffer captures this excess energy to use later while the smaller buffer quickly reaches capacity and discharges energy as heat to avoid overvoltage.

Large buffers, however, are not always more efficient: the energy used to charge the capacitor to the operational voltage cannot power useful work, and is eventually lost to leakage while the system is off. When power input is low, this "cold-start" energy represents a significant portion of total harvested energy. For the system described above powered by a solar panel at night [40], the 1 mF buffer enables a duty cycle of 5.7% versus only 3.3% using a 10 mF buffer. This low power environment highlights another risk of oversized buffers: the system using the 300 mF capacitor never reaches the enable voltage and so never begins operation.

Improvements in harvester efficiency and low-power chip design are closing the gap between harvester output and chip power consumption. Power is increasingly limited by volatile environmental factors rather than scarcity induced by low efficiency; the result is that energy harvesters experience periods of both energy scarcity and surplus. Rapidly changing power conditions place opposing demands on batteryless systems, which must remain responsive with low input power, provide longevity for long-running operations, and maximize efficiency by avoiding energy waste.

4.2.2 Power-Responsive Performance Scaling

One solution to volatile energy input is modulating throughput according to incoming power, increasing execution rate when power is plentiful and decreasing it to maintain availability when power is scarce. Limiting net input power to the buffer by matching power consumption with input enables systems to use small buffer capacitors without reaching the buffer capacity, ensuring no power is wasted with an over-full buffer. Past work realizes power-responsive scaling using heterogeneous architectures [32] or by adapting the rate and

accuracy of software execution [1, 6, 96, 143].

Unfortunately, I find the assumptions underlying power-performance scaling often do not apply to batteryless systems. Increasing energy consumption by accelerating execution only improves systems which *have useful work to do exactly when input power is high*, but many batteryless systems focus on periodic sensing and actuation deadlines which do not correlate with ambient power supply. Further, resource-constrained platforms may have few on-chip operations which can be delayed until power is plentiful; when these operations do exist, they are often not amenable to scaling (e.g., transmitting data to a base station may be delayed but always requires a fixed-cost radio operation). Flexible batteryless systems must capture energy and use it on demand rather than fit operation to unreliable power input.

4.2.3 Multiplexed Energy Storage

Rather than match power consumption to incoming supply, systems may charge multiple static buffers according to projected demand. Capybara [24] switches capacitance using an array of heterogeneous buffers: programmers set capacitance modes throughout the program, using a smaller capacitor to maximize responsiveness for low-power or interruptible tasks and switching to a larger capacitor for high-power atomic operations. UFoP and Flicker [48, 50] assign each peripheral on the system a separate buffer and charging priority, enabling responsive low-power operation while waiting to collect sufficient energy for high-power tasks. These systems increase overall energy capacity by directing excess power to capacitors not currently in use.

Static arrays increase capacity without reducing responsiveness, but waste energy when charge is stored on unused buffers. Reserving energy in secondary capacitors 1) requires error-prone [96] speculation about future energy supply and demand to decide charging

priorities, which can change between when energy is harvested and when it needs to be used; and 2), wastes energy as leakage when secondary buffers are only partially charged, failing to enable associated systems and keeping energy from higher-priority work. To minimize programmer speculation, decouple tasks which compete for buffered energy, and minimize leakage, energy must be *fungible*: the buffer must be capable of directing *all* harvested energy to *any* part of the system on demand.

4.2.4 Unified Dynamic Buffering

Past work has also explored varying the behavior of a single unified buffer to capture the fungibility requirement described above. Dewdrop [15] varies the enable voltage to draw from a single capacitor according to projected needs (e.g., begin operation at 2.2V instead of 3.6V)—providing complete energy fungibility—but still suffers from the reactivity-longevity tradeoff of capacitor size. Morphy [144] replaces static buffers using a set of capacitors in a unified switching network; software can connect and disconnect arbitrary sets of capacitors in series or parallel to produce different equivalent capacitances.

I evaluate *REACT* alongside Morphy because the Morphy architecture also targets the reactivity and longevity challenges discussed in § 4.1, but several key design choices differentiate each system. Morphy's fully-connected capacitor network (Figure 4.4) enables a wide range of equivalent capacitance configurations for a given set of capacitors, particularly for low capacitances: every capacitor in the network can be combined in series for an equivalent capacitance of C/N (assuming N C-sized capacitors), producing a highly-reactive system. This design also enables Morphy to act both as bulk energy storage and a large charge pump, boosting low-voltage inputs from an energy harvester by up to the number of capacitors in the system by alternating between fully-parallel and fully-series. In contrast, *REACT*'s

bank-based design (Figure 4.3) limits the number of potential configurations as all banks are effectively permanently in parallel—instead, *REACT* realizes high reactivity using a small static capacitor (§4.3.2).

The key advantage of *REACT*'s design is *energy efficiency*. As software reconfigures Morphy's fully-connected network to vary capacitance, current flows between capacitors *within* the network to equalize the output voltage of the array. This internal current dissipates a significant portion of stored energy; energy is lost on every reconfiguration of the network, dramatically reducing overall efficiency when faced with the highly volatile power inputs typical of many real-world deployments (§4.2.1) as the system rapidly varies capacitance to match net power input. My evaluation in §4.5 shows that this internal energy dissipation reduces common-case end-to-end performance below that of even static capacitors, making this approach impractical for energy-constrained devices. *REACT* eliminates this loss by dividing sets of capacitors into mutually isolated banks and ensuring no current flows between banks even as software varies capacitance. This energy-focused approach prioritizing minimal energy loss is key to developing intermittent systems that simultaneously maximize reactivity, longevity, and overall efficiency.

4.3 Design

An intelligent energy buffering strategy is key to effective and efficient batteryless systems. Three performance objectives, informed by the advantages and limitations of prior approaches, drive *REACT*'s design:

- **Minimize charge time:** Rapidly reaching the operational voltage, even when buffered energy cannot support complex operation, maximizes reactivity and enables systems to reason about power or sensor events from within low-power sleep modes.

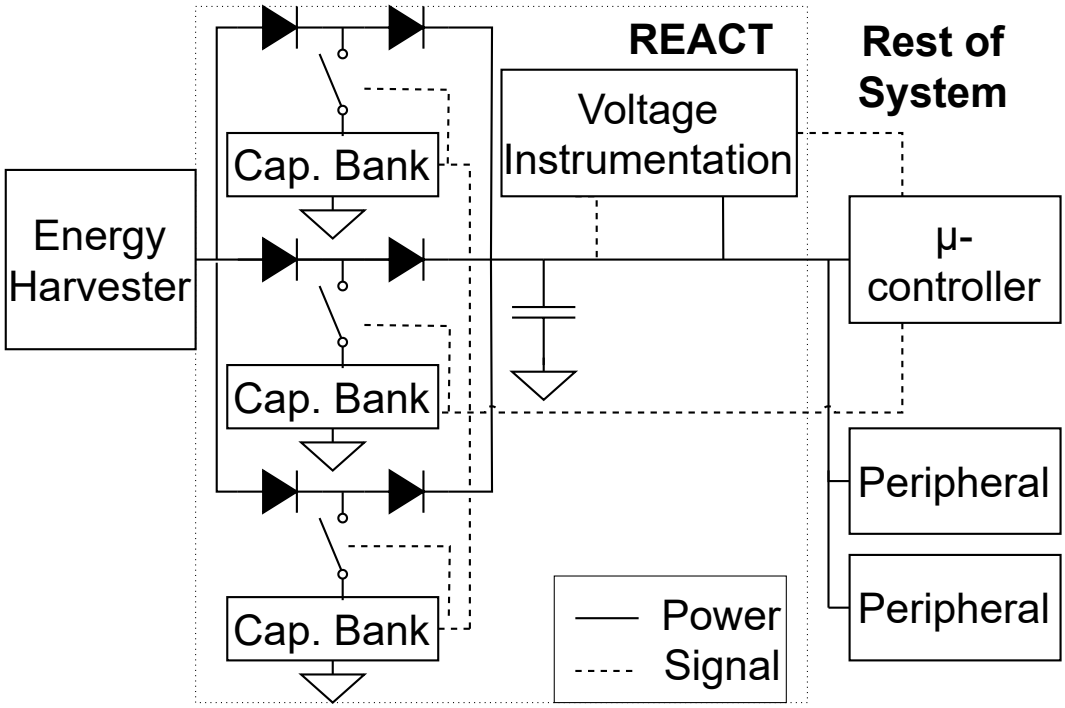


Figure 4.2: *REACT* diagram and signal flow between components.

- **Maximize capacity:** System-wide longevity and efficiency require buffering large amounts of incoming energy when power supply exceeds demand, either to power long-running uninterruptible operations or support future power demand when supply is low.

- **Maximize energy fungibility:** Unpredictable power demand patterns mean that energy cannot be pre-provisioned to specific operations at harvest-time; systems need the ability to draw all harvested energy from the buffer and direct it as needed.

4.3.1 *REACT* Overview

REACT buffers energy using a fabric of reconfigurable capacitor banks that software adjusts as needed. Figure 4.2 shows a high-level overview of *REACT*'s hardware design. I design

REACT's hardware component as a drop-in replacement for a typical buffer between the harvester and the rest of the system, while the buffer management software requires no code modification or programmer input. The only system prerequisite is a set of digital I/O pins to configure capacitor banks and receive voltage monitoring information.

4.3.2 Cold-start Operation and the Last-level Buffer

From a cold start ($E(t) = 0$), *REACT* minimizes overall capacitance in order to rapidly charge to the operational voltage and enable the system with minimum energy input (high reactivity). The minimum capacitance is set by the smallest quantum of useful work available on the system (minimum required longevity), such as a short-lived software operation or an initialization routine that puts the system into a low-power responsive sleep mode. *REACT* provides this rapid charge time using a small static buffer referred to hereafter as the *last-level buffer*. Additional capacitor banks are connected using normally-open switches and only contribute to overall capacitance when configured to do so in software, after the system is able to reason about buffered energy.

The last-level buffer sets the minimum capacitance at power-on when all other banks are disconnected. This enables simple tuning of the energy input required to enable the system (reactivity) and the guaranteed energy level when the system does begin work (minimum longevity). It also smooths voltage fluctuations induced by capacitor bank switching (§ 4.3.3). Finally, the last-level buffer serves as the combination point between the different capacitor banks and the rest of the system. Although energy may be stored in multiple banks of varying capacity at different voltages, combining it at the last-level buffer simplifies system design by presenting harvested power as a unified pool of energy which the system taps as needed (i.e., harvested energy is fungible).

Monitoring Buffered Energy

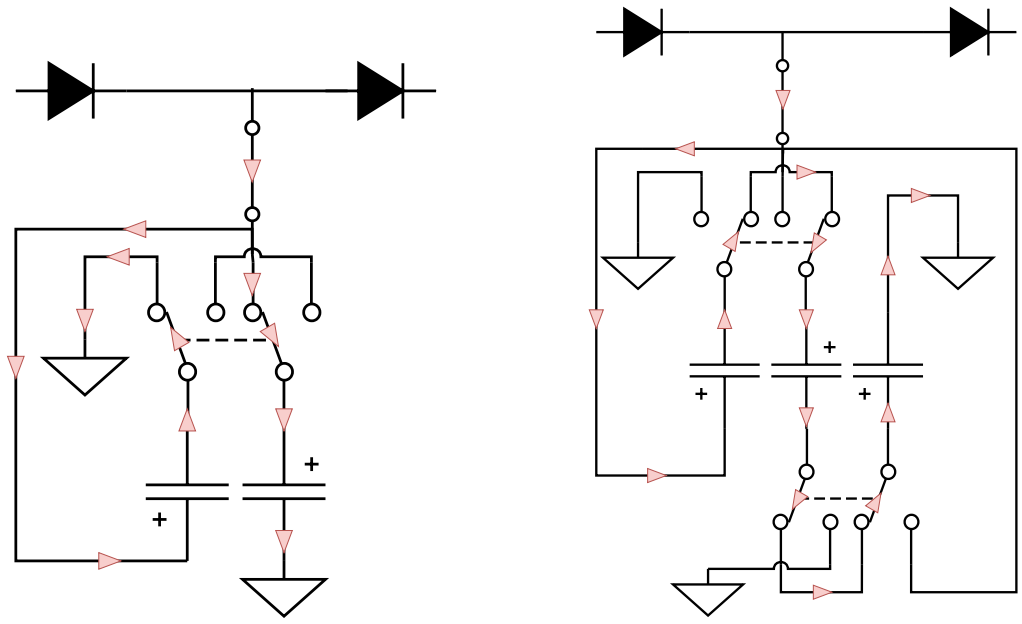
Despite mutual isolation, bank voltages tends to equalize: the last-level buffer pulls energy from the highest-voltage bank first, and current flows from the harvester to the lowest-voltage bank first. This enables *REACT* to measure only the voltage on the last-level buffer as a surrogate for remaining energy capacity. If voltage rises beyond an upper threshold—the buffer is near capacity—*REACT*'s voltage instrumentation hardware signals the software component running on the microcontroller to increase capacitance using the configurable banks. Voltage falling below a lower threshold indicates the buffer is running out of energy and that *REACT* should reconfigure banks to extract additional energy and extend operation. *REACT*'s instrumentation only needs to signal three discrete states—near capacity, near undervoltage, and OK—so two low-power comparators is sufficient for energy estimation.

4.3.3 Dynamic Capacitor Banks

The last-level buffer on its own enables high reactivity and minimizes cold-start energy below the operational minimum, maximizing efficiency during power starvation. However, when net power into the buffer is positive—such as during a period of high input power or low workload—the small last-level buffer rapidly reaches capacity. *REACT* provides the energy capacity required to both maximize efficiency and support long-running operation by connecting configurable capacitor banks when the last-level buffer reaches capacity, as shown in Figure 4.2.

Capacitor Organization

Careful management of the connections between each capacitor is key to maximizing energy efficiency while also presenting a valid operational voltage for the computational backend.



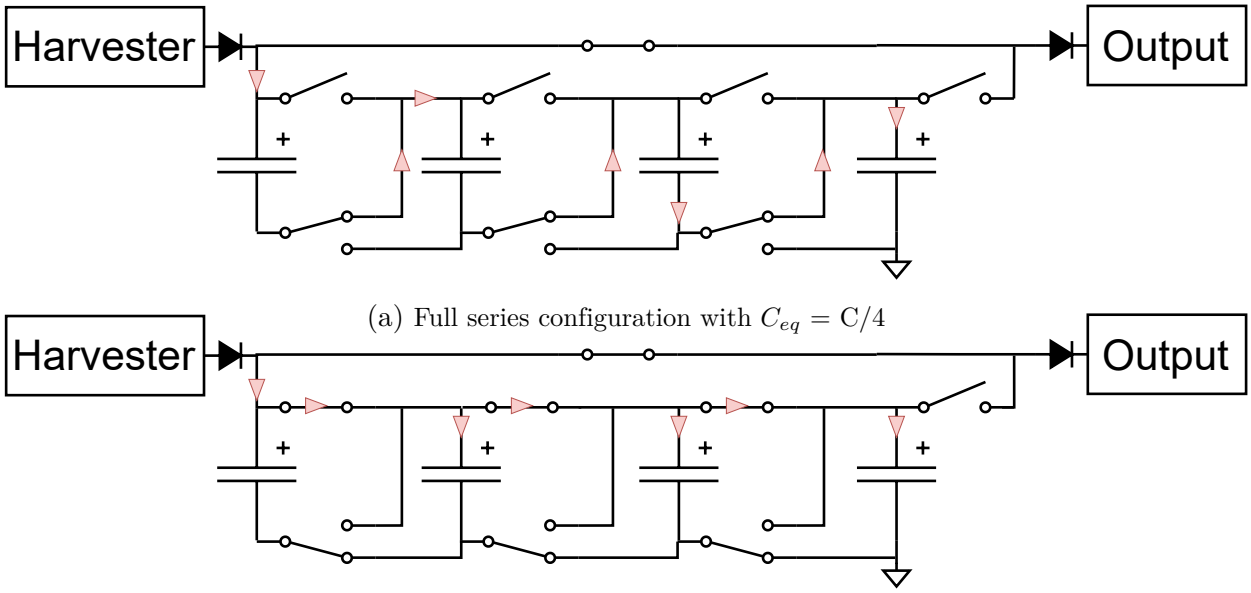
(a) Two capacitors, parallel configuration

(b) Three capacitors, series configuration

Figure 4.3: *REACT* capacitor banks in different bank sizes and configurations. Arrows indicate charging current path.

Morphy [144] presents one approach: by connecting a set of equally-sized capacitors through switches similar to a charge pump, overall buffer capacitance can be varied across a wide range of capacitance values. Different switch configurations produce intermediate buffer sizes between the extremes shown in Figure 4.4; gradually stepping through these configurations smoothly varies capacitance through software control.

A fully interconnected array enables a wide range of equivalent capacitances, but introduces significant waste through dissipative heating when the charged capacitor array is reconfigured. Figure 4.5 illustrates how energy is lost when charged capacitors are connected in a new configuration. Before reconfiguration, the energy contained in the system is $E_{old} = \frac{1}{2}(C/4)V^2$; when a capacitor is taken out of series and placed in parallel with the remaining capacitors to increase equivalent capacitance to $4C/3$, current flows to the



(a) Full series configuration with C_{eq} = C/4

(b) Full parallel configuration with C_{eq} = 4C

Figure 4.4: Structure of the unified approach presented by Yang et al. [144]. Arrows indicate charging current path.

---

lower-voltage newly-parallel capacitor to equalize output voltage. The final output voltage is $3V/8$, and the remaining energy is $E_{new} = \frac{1}{2}(4C/3)(3V/8)^2$. The portion of energy conserved is $E_{new}/E_{old} = 0.75$—i.e., 25% of buffered energy is dissipated by current in the switches during reconfiguration. Larger arrays are increasingly inefficient: the same scenario with an 8-capacitor array wastes 56.25% of its buffered energy transitioning from an 8-parallel to a 7-series-1-parallel configuration. Similar waste occurs when reducing equivalent capacitance by placing capacitors in series.[2] My evaluation in § 4.5.6 indicates that the energy loss caused by switching often outweighs any advantage from dynamic behavior, causing the fully-connected approach to underperform even static buffers.

[2]Charge pumps avoid this waste by never connecting capacitors at different potentials in parallel; in this use case, however, parallel capacitance is always necessary to smooth voltage fluctuations during switching and keep the output voltage within the computational backend's acceptable range.

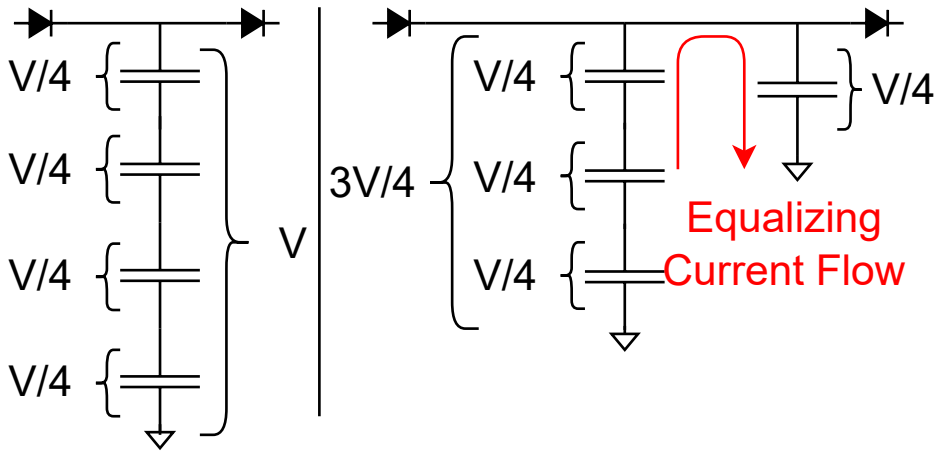


Figure 4.5: Dissipative current flow in a fully-unified buffer during reconfiguration. Energy is dissipated by current spikes after capacitors at different voltages are placed in parallel.

Bank Isolation

The switching loss discussed above stems from charge flowing between capacitors within the power network as they switch into different configurations. *REACT* eliminates unnecessary current flow by organizing capacitors into independent, mutually isolated banks as shown in Figure 4.2. Figure 4.3 illustrates in detail two example capacitor banks in each possible configuration: capacitors within a bank can only be arranged in either full-series (low capacitance) or full-parallel (high capacitance) so that no current flows between capacitors *within* a bank. Isolation diodes on the input and output of each bank prevent current *between* banks: when a charged parallel-configured bank is reconfigured into series (reducing its capacitance and boosting its output voltage), isolation diodes prevent it from charging other banks in the array. Similarly, banks re-configured into parallel cannot draw current from anywhere except the energy harvester. Isolation reduces the number of potential capacitor configurations compared to a fully-connected network, but dramatically increases energy efficiency.

REACT's isolation diodes direct the flow of current: intermediate capacitor arrays are only charged directly from the energy harvester and only discharge to the last-level buffer. This

also means that all current from the harvester flows through two diodes before reaching the system, so minimizing power dissipation in the diodes is essential to maintaining overall system efficiency. To maximize charging efficiency, I design *REACT* using ideal diode circuits incorporating a comparator and pass transistor, rather than typical PN or Schottky diodes. Active ideal diodes are far more efficient at typical currents for batteryless systems: the circuit I use [71] dissipates 0.02% of the power dissipated in a typical Schottky diode [128] at a supply current of 1 mA.

Bank Reconfiguration

The range of buffer sizes depends on the number of capacitor banks and the number of capacitors in each bank. *REACT*'s capacitor banks are effectively connected in parallel, so the overall capacitance is the sum of each bank's contribution. Each *REACT* bank containing N identical capacitors of capacitance C may be configured to contribute no capacitance (disconnected), series capacitance C/N, or parallel capacitance NC.

REACT must increment buffer capacitance in small steps in order to keep voltage within the operational range while capturing all incoming power. A large increase in capacitance pulls output voltage down and introduces cold-start energy loss if net power input is low; for extreme cases, the system may run out of energy and cease execution while the new capacitance charges *even if incoming power would be sufficient to power operation.* *REACT* first connects banks in the series configuration to contribute a small capacitance and avoid large jumps in overall buffer size. If the buffer continues to charge and reaches the upper voltage limit V_{high}, *REACT* further expands capacitance by toggling double-pole-double-throw bank switches to configure the capacitors in parallel. Expanding the buffer by reconfiguring charged capacitors rather than adding new ones reduces the time the system is cut off from input power while current flows exclusively to the new capacitance, because it is already

charged to V_{high}/N. Because no current flows between capacitors or banks, bank reconfiguration changes capacitance seen on the common rail without dissipative loss. *REACT* uses break-before-make switches to ensure no short-circuit current flows during switching; incoming current flows directly to the last-level buffer during the momentary open-circuit in the bank.

Charge Reclamation

Reconfiguring a bank from series to parallel allows *REACT* to efficiently increase capacitance without dropping output voltage. When voltage on the last-level buffer appraches the threshold value V_{low}, indicating net power is leaving the buffer, *REACT* needs to reduce equivalent capacitance to boost voltage and keep the backend running. *REACT* accomplishes this by transitioning charged N-capacitor banks from the parallel to the series configuration, reducing equivalent capacitance from NC to C/N and boosting output voltage from V_{low} to NV_{low}. This boosts voltage on the last-level buffer and extracts more energy from the capacitor bank than would otherwise be available once voltage falls below V_{low}.

The remaining energy unavailable after the parallel→series transition depends on the number N of C_{unit}-size capacitors in the bank. Before switching, the cold-start energy stored on the parallel-mode bank is $E_{par} = \frac{1}{2}NC_{unit}V_{low}^2$. Switching to the series configuration conserves stored energy: $E_{ser} = \frac{1}{2}(C_{unit}/N)(NV_{low})^2 = E_{par}$, but boosts voltage to enable the digital system to continue extracting energy. If net power remains negative, the system eventually drains the series-configuration bank down to V_{low}. This is energetically equivalent to draining the parallel-configuration bank to V_{low}/N, leaving $E_{par} = \frac{1}{2}NC_{unit}(V_{low}/N)^2 = \frac{1}{2}C_{unit}V_{low}^2/N$ unusable; the overall result is that *REACT* reduces energy loss by a factor of N^2 when reducing system capacitance compared to simply disconnecting the capacitor.

Bank Size Constraints

Increasing the number of capacitors N in a bank improves efficiency by reclaiming more energy when switching a bank from parallel to series. However, it also introduces voltage spikes when the bank output voltage is temporarily multiplied by N. Because *REACT* measures overall energy at the last-level buffer, the software component may interpret this voltage spike as a buffer-full signal and incorrectly add capacitance despite low buffered energy. In extreme cases, the voltage spike may exceed component absolute limits.

The size of the last-level buffer C_{last} constrains the number N and size C_{unit} of each capacitor in a bank in order to keep voltage below *REACT*'s buffer-full threshold during a parallel→series transition. A larger C_{unit} contains more energy and thus pulls voltage higher when switched from parallel to series. Equation 4.1 gives the last-level buffer voltage after switching a bank to series at a trigger voltage V_{low}:

$$V_{new} = \frac{(NV_{low})(C_{unit}/N)}{C_{last} + C_{unit}/N} + \frac{V_{low} * C_{last}}{C_{last} + C_{unit}/N} \tag{4.1}$$

Constraining $V_{new} < V_{high}$ and solving for C_{unit} yields the absolute limit for C_{unit} (Equation 4.2). Note that C_{unit} is only constrained if the parallel→series transition at V_{low} produces a voltage above V_{high}:

$$C_{unit} < \frac{NC_{last}(V_{high} - V_{low})}{NV_{low} - V_{high}} \tag{4.2}$$

4.3.4 *REACT* Software Interface

REACT's standalone hardware design means that the software component running on the target microcontroller is minimal. The software subsystem monitors for incoming over- or under-voltage signals from *REACT*'s voltage instrumentation and maintains a state machine for each capacitor bank. Each capacitor bank is disconnected at startup; on an overvoltage

signal from *REACT*'s hardware, the software directs *REACT* to connect a new capacitor bank in the series configuration. A second overvoltage signal[3] causes *REACT* to reconfigure the newly-connected bank to parallel; on the next overvoltage signal, *REACT* connects a second capacitor bank, and so on. *REACT* similarly steps capacitor banks in the opposite direction when an undervoltage signal arrives.

My proof-of-concept *REACT* implementation uses external voltage detectors with hard-wired thresholds to supply over/undervoltage signals to the software component and varies capacitance to keep supply voltage within a range corresponding to the limits for my load microcontroller (2.0-3.4V). One possible extension to *REACT* is to instead use on-chip analog comparators or ADCs and redefine these thresholds in software, enabling developers to dynamically change the target operating voltage at different points in the program. Decoupling supply voltage from buffered energy in this way has both functional benefits (some operations, like writes to Flash memory, require a higher supply voltage) and performance benefits (e.g., keeping the input to a voltage regulator at its most efficient operating point).

Software-Directed Longevity

REACT's software component requires no active programmer intervention or code changes aside from setting voltage thresholds, initializing each bank state machine, and setting the order to connect and disconnect banks. Software does not need to know the details (N, C_{unit}) of each bank, although this information with the state of each bank gives a coarse idea of the current buffered energy. Because *REACT* only changes capacitance when the bank is near-full or near-empty, capacitance level is an effective surrogate for stored energy. Application code can use this feedback to set longevity guarantees through *REACT*'s software interface.

[3] *REACT* polls the over/undervoltage signals using an internal timer rather than edge-sensitive interrupts to handle cases such as a high enough power input that the capacitance step does not pull supply voltage below V_{low}.

Bank	**0**	**1**	**2**	**3**	**4**	**5**
Capacitor Size (μF)	770	220	440	880	880	5000
Capacitor Count	1	3	3	3	3	2

Table 4.1: Bank size and configurations for my *REACT* test implementation. Bank 0 is the last-level buffer.

In preparation for a long-running or high-energy atomic operation, software sets a *minimum capacitance level* corresponding to the amount of energy required and then enters a deep-sleep mode keeping *REACT*'s capacitor polling time active. As the system charges, *REACT* eventually accumulates enough energy to reach the minimum capacitance level—indicating that enough energy is stored to complete the planned operation, and pulling the system out of its deep-sleep with enough energy to complete execution regardless of future power conditions.

4.4 Implementation

I explore *REACT*'s impact on overall efficiency, reactivity, and longevity using a hardware prototype integrated into a real batteryless platform. My testbed is based on the MSP430FR5994 [74], a popular microcontroller for energy harvesters [97, 134, 135]. For each buffer configuration I evaluate, an intermediate circuit power gates the MSP430 to begin operation once the buffer is charged to 3.3V and disconnects it when the buffer voltage reaches 1.8V.

My *REACT* implementation has a range of 770 μF-18.03 mF using a set of 5 dynamic banks, in addition to the last-level buffer, detailed in Table 4.1. I implement the capacitors in banks 0-4 using combinations of 220 μF capacitors with max leakage current of 28 μA at their rated voltage of 6.3V [103]. Bank 5 uses supercapacitors with approximately 0.15 μA

leakage current at 5.5V [104].

4.4.1 Baseline Systems

I evaluate *REACT* against three fixed-size buffers spanning my implementation's capacitance range—770 μF, 10 mF, and 18 mF—to ensure the realized improvement is a result of energy-adaptive behavior rather than simply different buffer capacity. To compare *REACT*'s capacitor architecture to prior work on dynamic energy buffers, I also implement and evaluate Morphy [144] for a similar capacitance range. My Morphy implementation uses eight 2 mF capacitors with leakage current of approximately 25.2 μA at 6.3V [106] (i.e., slightly lower leakage than the capacitors in *REACT*).

Morphy uses a secondary microcontroller powered by a battery or backup capacitor to control the capacitor array; I use a second MSP430FR5994 powered through USB, corresponding to Morphy's battery-powered design. Accordingly, I expect my results to slightly overestimate Morphy's performance in the fully-batteryless case as the system does not have to power the Morphy controller or charge a backup capacitor in my implementation. Seven of the eight capacitors in the array are available to reconfigure, with one task capacitor kept in parallel to smooth voltage fluctuations from switching. I evaluate the same subset of eleven possible configurations for the remaining seven capacitors as is done in the original Morphy work, resulting in a capacitance range for my Morphy implementation of 250 μF-16 mF.

4.4.2 Computational Backend

To explore how *REACT* affects performance across a range of system demands—focusing on diverse reactivity and longevity requirements—I implement four software benchmarks:

- **Data Encryption (DE)**: Continuously perform AES-128 encryptions in software. This application has no reactivity requirements, low persistence requirements, and a predictable power draw; I use it as a baseline to explore *REACT*'s software and power overhead. Figure of Merit: Number of AES encryptions completed, not including encryptions interrupted by power loss.

- **Sense and Compute (SC)**: Exit a deep-sleep mode once every five seconds second to sample and digitally filter readings from a low-power microphone [36]. This benchmark represents systems which value high reactivity and can accept low persistence; individual atomic measurements are low-energy, but the system must be online to take the measurements. Figure of Merit: Number of sensing tasks completed, not including samples or processing interrupted by power loss.

- **Radio Transmission (RT)**: Send buffered data over radio [57, 101] to a base station. Data transmission is an example of an application with high persistence requirements (radio transmissions are atomic and energy-intensive) and low reactivity requirements (transmitting data may be delayed until energy is available). Figure of Merit: Number of transmissions completed, not including transmissions interrupted by power loss.

- **Packet Forwarding (PF)**: Listen for and retransmit incoming data over the radio. Timely operation demands both high persistence and reactivity to successfully receive and retransmit data. Figures of Merit: Packets received and transmitted, not including operations interrupted by power loss.

I emulate the power consumption of the necessary peripherals for each benchmark by toggling a resistor connected to a digital output on the MSP430, with values for each benchmark chosen to match the relevant peripheral. The reactivity-focused benchmarks (SC and PF) have deadlines that may arrive while the system is off; I use a secondary MSP430 to deliver

	Data Encrypt					Sense and Compute					Radio Transmit				
Buffer	**770μ**	**10m**	**17m**	**Morphy**	**REACT**	**770μ**	**10m**	**17m**	**Morphy**	**REACT**	**770μ**	**10m**	**17m**	**Morphy**	**REACT**
RF Cart	1275	1574	1831	1745	1711	50	81	104	77	83	22	53	56	38	48
RF Obs.	666	472	0	357	576	44	28	0	39	49	4	6	0	0	3
RF Mob.	810	1004	645	801	1038	52	50	40	53	84	4	13	12	4	15
Sol. Camp.	6666	7290	7936	8194	9756	330	353	367	398	439	1376	1457	1542	1059	1426
Sol. Comm.	2168	2186	2554	2399	2232	88	110	130	133	154	8	40	48	31	34
Mean	2317	2505	2593	2699	3063	113	124	128	140	162	283	314	332	226	313

Table 4.2: Performance by figure of merit on the DE, SC, and RT benchmarks, across traces and energy buffers.

these events. A deployed system may use remanence-based timekeepers [28] to track internal deadlines despite power failures for the SC benchmark, while incoming packets as in the PF benchmark would arrive from other systems. Although I evaluate each benchmark in isolation, full systems are likely to exercise combinations of each requirement—one platform should support all reactivity, persistence, and efficiency requirements.

4.4.3 Energy Harvesting Frontend

Energy volatility makes repeatable experimentation with batteryless devices difficult; uncontrollable environmental changes often have an outsized effect on energy input and obfuscate differences in actual system performance. I make my experiments repeatable and consistent using a programmable power frontend inspired by the Ekho [49] record-and-replay platform. The power controller supplies the energy buffer using a high-drive Digital-to-Analog Converter (DAC), measures the load voltage and input current using a sense resistor, and tunes the DAC to supply a programmed power level. I evaluate *REACT* emulating both solar (5 cm^2, 22% efficient cell [133]) and RF energy (915 MHz dipole antenna [114]).

Trace	Time (s)	Avg. Pow. (mW)	Power CV*
RF Cart	313	2.12	103%
RF Obstruction	313	0.227	61%
RF Mobile	318	0.5	166%
Solar Campus	3609	5.18	207%
Solar Commute	6030	0.148	333%

Table 4.3: Details of each power trace. *CV = Coefficient of Variation.

4.5 Evaluation

I evaluate *REACT* alongside the baseline buffers running each benchmark under three RF and two solar traces from publicly available repositories [4, 40], representative of power dynamics for small energy harvesting systems. I record the RF traces in an active office environment using a commercial harvester and transmitter [112, 113] and use solar irradiance traces from the Enhants mobile irradiance dataset [40]; Table 4.3 gives a short summary of each trace. These traces show the power variability common for IoT-scale harvesters: environmental changes (e.g., ambient RF levels, time of day) affect *average* input power, while short-term changes such as orientation cause instantaneous variation even if the environment is unchanged. I apply each trace using the power replay system described in § 4.4.3; once the trace is complete, I let the system run until it drains the buffer capacitor.

4.5.1 Software and Energy and Overhead

Figure 4.6 illustrates *REACT*'s behavior through the last-level buffer voltage when varying capacitance; the inset focuses on *REACT*'s voltage output as it expands to capture energy (also shown is the voltage of the comparable Morphy array). From a cold start *REACT* only charges the last-level buffer—rapidly reaching the enable voltage and then the upper

voltage threshold (3.5V). *REACT* then adds a series-configured capacitor bank to capture excess incoming energy. Voltage drops as the system temporarily operates exclusively from the last-level buffer while harvested energy goes towards charging the new capacitance. As power input falls, *REACT*'s output voltage falls below the upper threshold voltage—indicating *REACT* is operating at an efficient capacitance point. At $t \approx 450s$ the last-level buffer is discharged to the lower threshold and *REACT* begins switching banks into series mode to boost their output voltage and charge the last-level buffer, visible in Figure 4.6 as five voltage spikes corresponding to each capacitor bank—sustaining operation until no more energy is available at $t \approx 500s$.

I characterize *REACT*'s software overhead by running the DE benchmark on continuous power for 5 minutes with and without *REACT*'s software component, which periodically interrupts execution to measure the capacitor bank. At a sample rate of 10 Hz, *REACT* adds a 1.8% penalty to software-heavy applications. I measure *REACT*'s power overhead by comparing the execution time of systems running the DE benchmark using *REACT* and the 770 μF buffer after charging each to their enable voltage. Based on this approach I estimate that my implementation of *REACT* introduces a 68 μW power draw, or ~14 μW per bank.

4.5.2 Characterization

Table 4.4 details the cost at scale of *REACT* compared to Morphy for a capacitor network consisting of eight 100 mF supercapacitors. Considering the same supercapacitors and switch ICs as in the original Morphy implementation [144], *REACT* using four two-capacitor banks is approximately 25% less expensive than Morphy owing to its simpler capacitor switching structure: *REACT* requires one Double-Pole-Double-Throw (DPDT) and one Single-Pole-Single-Throw (SPST) switch per *bank*, corresponding to six switch ICs (as each IC contains

Device	1000x Unit Cost	Morphy		REACT	
		Count	Cost	Count	Cost
Supercap. [104]	$1.54	9	$13.87	8	$12.33
Switch IC [33]	$1.53	12	$18.36	6	$9.18
Ideal Diode [71]	$0.27	1	$0.27	8	$2.16
Ceramic Cap. [103]	$0.45	0	$0	4	$1.82
Comparator IC [75]	$0.77	0	$0	1	$0.77
Detector IC [126]	$0.26	0	$0	2	$0.52
Control MCU [100]	$2.37	1	$2.37	0	$0
RTC [25]	$1.08	1	$1.08	0	$0
Platform Cost			**$35.94**		**$26.77**

Table 4.4: Cost comparison between *REACT* and Morphy implementations targeting the same capacitance range.

two SPDT switches which can be combined to form one DPDT switch). The Morphy network requires one DPDT and one SPST switch per *capacitor*, plus an additional SPST switch to fully connect the network. The battery-free version of Morphy requires an additional capacitor and diode to power the Morphy controller; following the original Morphy design, I use the same type of capacitor as in the switching network. Other cost differences (i.e., *REACT*'s diodes and voltage instrumentation versus Morphy's secondary microcontroller and real-time clock) are relatively minor as capacitor and switch costs dominate overall cost.

Designers can tune the structure of the capacitor network to vary aspects of *REACT*'s performance. One alternative for the eight-capacitor network described above is to organize *REACT* using two four-capacitor banks; this design does not significantly change unit cost (each four-capacitor bank requires one SPST and three DPDT switches, and the overall system requires half as many diodes) and increases efficiency by extracting more energy from each capacitor bank (§4.3.3). The tradeoff is that using fewer and larger banks reduces the number of possible capacitance configurations and requires increasing the size of the last-level buffer to smooth voltage spikes, limiting reactivity from a cold start. In practice, application demands will determine the ideal *REACT* network and corresponding balance of efficiency and flexibility.

Buffer	770 μF	10 mF	17 mF	Morphy	REACT
RF Cart	6.65	17.73	31.27	5.51	6.65
RF Obs.	14.58	223.07	-	6.50	16
RF Mob.	6.90	148.10	239.88	5.65	6.38
Sol. Camp.	42.11	737.39	741.42	35.59	41.26
Sol. Comm.	119.60	196.30	213.00	108.10	130.6
Mean	37.97	264.92	306.39	32.27	40.18

Table 4.5: System latency (seconds) across traces and energy buffers. - indicates system never begins operation.

4.5.3 *REACT* Minimizes System Latency

Table 4.5 details the time it takes each system to begin operation, across power traces and energy buffers (charge time is software-invariant and constant across benchmarks). Latency is driven by both capacitor size and environment—the $10mF$ buffer is ~$13x$ larger than the $770\mu F$ buffer and takes on average $7x$ longer to activate the system across my traces. High-capacity static buffers incur a larger latency penalty even if mean power input is high if much of that power is contained in a short-term spike later in the trace (e.g., for the Solar Campus trace), but these dynamics are generally impossible to predict at design time. By exclusively charging the last-level buffer while the rest of the system is off, *REACT* matches the latency of the smallest static buffer—an average of $7.7x$ faster than the equivalent-capacity 17 mF buffer, which risks failing to start at all. Morphy further reduces system latency because its smallest configuration is smaller than *REACT*'s last-level buffer (250 μF vs 770 μF), although the limited reduction in average latency compared to the reduction in capacitance (Morphy realizes an average 20% reduction in latency over *REACT* using a 68% smaller capacitance) suggests that further reducing capacitance yields diminishing latency returns in realistic energy environments.

Minimizing latency improves reactivity-bound applications such as the SC and PF bench-

marks; this effect is visible in Table 4.2 as the 770 μF buffer outperforms larger ones in the SC benchmark for relatively low-power traces (RF Mobile/Obstructed). *REACT* inherits the latency advantage due to the small last-level buffer, similarly improving performance on each power trace. Morphy realizes a similar performance improvement over the static systems, but ultimately underperforms *REACT* as a result of inefficient capacitor switching (§ 4.5.6). Small static buffers enable low-latency operation, but at the cost of energy capacity. As power input increases, the latency penalty of large buffers fades and their increased capacity enables them to operate for longer—resulting in higher performance for larger static buffers under high-power traces (RF Cart, Solar Campus). Smaller buffers, in turn, become less efficient as they must burn more incoming energy off as waste heat.

4.5.4 *REACT* Maximizes Energy Capacity

Figure 4.6 illustrates the system-level effects of the capacity-latency tradeoff, and how *REACT* avoids this tradeoff through energy-adaptive buffering. The small 770μF buffer charges rapidly, but reaches capacity and discharges energy when it does not have work to match incoming power (illustrated by clipping at 3.6V on the 770 μF line). The 10 mF buffer sacrifices latency for capacity—starting operation 21x later than the smaller buffer, but avoiding overvoltage. Morphy begins execution early with a small capacitance, but its lossy switching mechanism means it does not overall outperform the 770μF buffer. In contrast, *REACT* achieves low latency, high efficiency, *and* high capacity by efficiently expanding capacitance as necessary after enabling the system.

Tables 4.2 and 4.6 show that high capacity is valuable when average input power exceeds output power (e.g., DE and SC benchmarks executed under the RF Cart trace), or when peak power demand is uncontrollable and uncorrelated with input (e.g., the PF benchmark

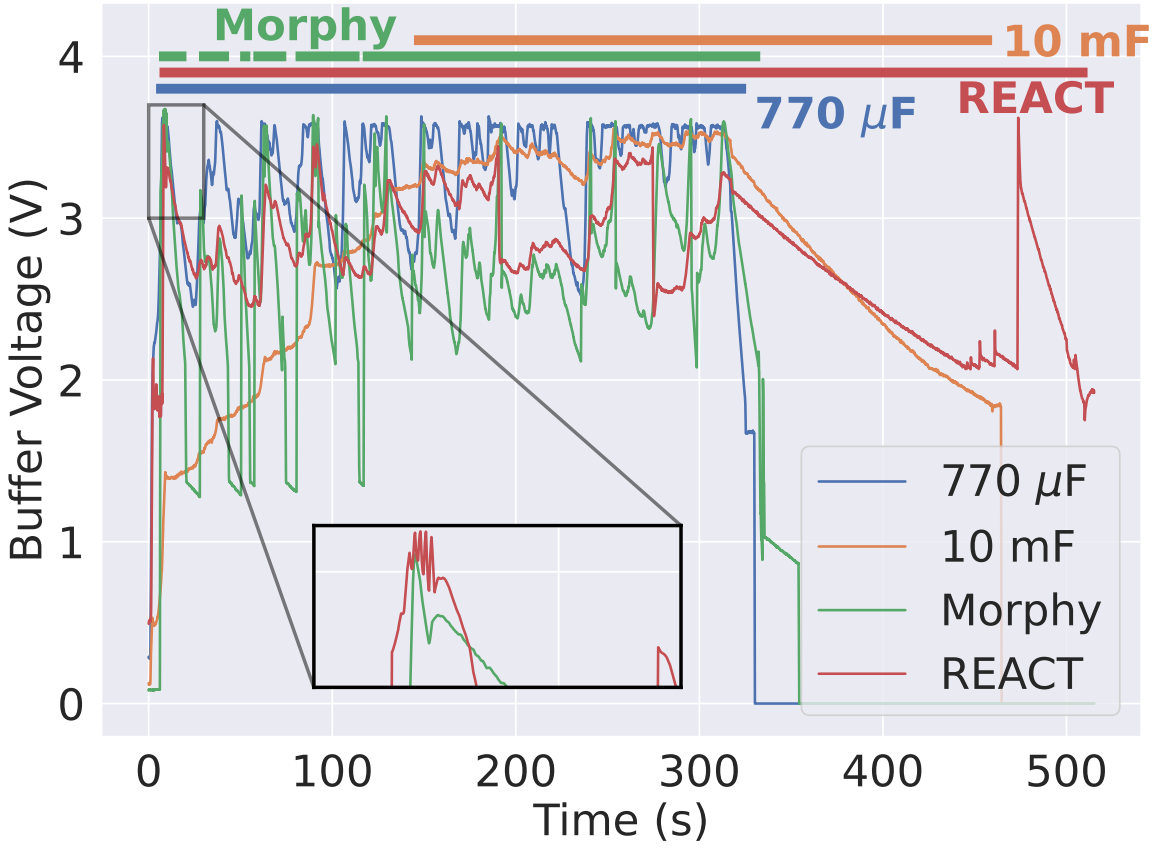


Figure 4.6: Buffer voltage and on-time for the SC benchmark under RF Mobile power. Solid bars indicate when the system is operating.

Buffer	**770 uF**		**10 mF**		**17 mF**		**Morphy**		**REACT**	
Packets	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx
RF Cart	22	10	49	49	48	48	55	22	53	52
RF Obs.	4	4	4	4	0	0	2	0	3	0
RF Mob.	11	4	14	13	9	9	19	0	38	5
Sol. Camp.	163	163	240	240	196	196	206	204	284	277
Sol. Comm.	72	8	35	35	33	33	85	14	84	63
Mean	54	38	68	68	57	57	73	48	92	80

Table 4.6: Packets successfully received and retransmitted during the Packet Forwarding benchmark.

executed on Solar Campus, where both power supply and demand are concentrated in short bursts). In both cases, high-capacity systems store excess energy to continue operation even if future power input falls or demand rises. *REACT* efficiently expands to capture all incoming energy during periods of high net input power, matching or beating the performance of the 10 mF and 17 mF systems when they outperform the small 770 μF buffer.

4.5.5 *REACT* Provides Flexible, Efficient Longevity

I evaluate *REACT*'s software-directed longevity guarantees (§ 4.3.4) on the longevity-bound RT and PF benchmarks. I compare *REACT* to the 770 μF buffer, which cannot sustain a full transmission without additional input power. Figure 4.7 illustrates this limitation, as the 770 μF system charges quickly but wastes power on doomed-to-fail transmissions when incoming power cannot make up for the deficit and the system powers off before completing a transmission. This shortcoming is reflected in Table 4.2, where the 770 μF system ultimately significantly underperforms the other buffers. I augment the RT benchmark code for my *REACT* implementation to include a minimum capacitance level for *REACT*, below which

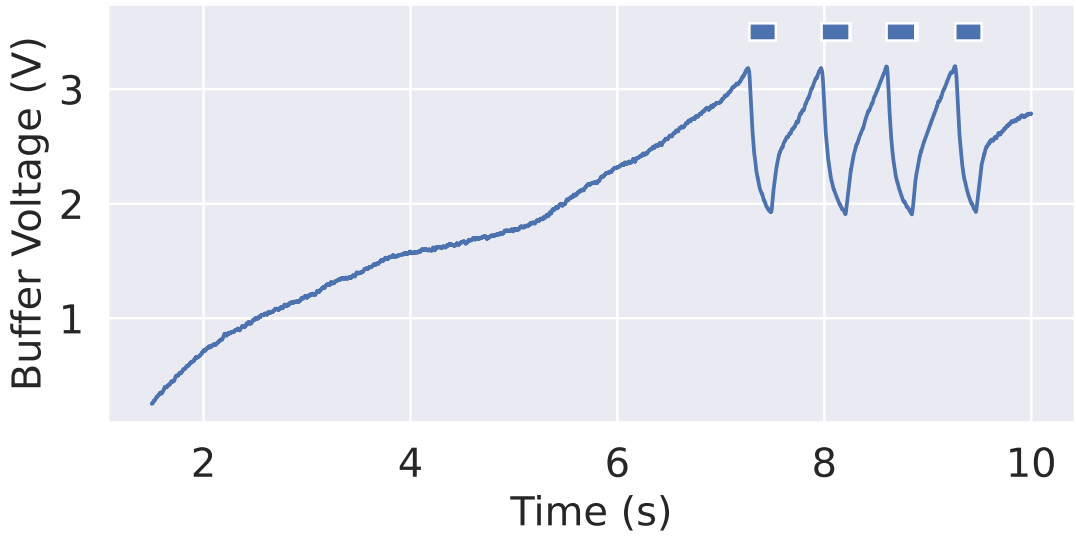


Figure 4.7: Buffer voltage and on-time for the 770 μF system running the RT benchmark under the RF Cart trace. Solid bars indicate when the system is operating.

---

the system waits for more energy in a low-power sleep mode. Leveraging *REACT*'s variable capacitance allows software to buffer energy to guarantee completion, more than doubling the number of successful transmissions and ultimately outperforming even the larger buffers.

I use the same approach to execute the RT benchmark on my Morphy implementation. Similar to *REACT*, Morphy varies capacitance to keep supply voltage within an acceptable level for the application microcontroller while also waiting to gather enough energy to power a full transmission. Morphy's underperformance compared to both *REACT* and the static buffers is a result of Morphy's capacitor network design—as Morphy reconfigures the capacitor array to increase capacitance, stored energy is dissipated as current flows between capacitors in the network. This energy dissipation dramatically reduces Morphy's end-to-end performance, particularly in systems where Morphy *must* switch capacitance to ensure success (i.e., the RT and PF benchmarks). *REACT*'s isolated capacitor banks eliminate this problem by restricting current flow during switching; the energy savings are reflected in the

end-to-end performance, where *REACT* completes on average 38% more transmissions than Morphy.

Fungible Energy Storage

A unified buffer means that energy is fungible, and *REACT* is flexible: software can re-define or ignore previous longevity requirements if conditions or priorities change. The PF benchmark (Table 4.6) shows the value of energy fungibility using two tasks with distinct reactivity and longevity requirements. Receiving an incoming packet requires a moderate level of longevity, but is uncontrollable and has a strict reactivity requirement (the system can only receive a packet exactly when it arrives). Re-transmission requires more energy but has no deadline. Software must effectively split energy between a controllable high-power task and an uncontrollable lower-power task.

I again use the minimum-capacitance approach to set separate longevity levels for each task, using a similar approach for my Morphy implementation. When the system has no packets to transmit, it waits in a deep-sleep until receiving an incoming packet. If *REACT* contains sufficient energy when the packet arrives, it receives and buffers the packet to later send. *REACT* then begins charging for the transmit task, forwarding the buffered packet once enough energy is available. If another packet is received while *REACT* is charging for the transmit task, however, software disregards the transmit-associated longevity requirement to execute the receive task if sufficient energy is available.

Table 4.6 shows that *REACT* outperforms all static buffer designs on the PF benchmark by efficiently addressing the requirements of both tasks, resulting in a mean performance improvement of 54%. *REACT*'s maximal reactivity enables it to turn on earlier and begin receiving and buffering packets to send during later periods of high power, while its high

capacity enables it to make the most of incoming energy during those high periods. Software-level longevity guarantees both ensure the system only begins receive/transmit operations when enough energy is available to complete them, and that software can effectively allocate energy to incoming events as needed. Although Morphy enables the same software-level control of energy allocation, the energy dissipated when switching capacitors in the interconnected array reduce its performance on the PF benchmark to below that of the best performing static buffer.

4.5.6 *REACT* Improves End-to-End System Efficiency

Optimizing buffer behavior maximizes energy available to the end system for useful work. Figure 4.8 illustrates the aggregate performance of *REACT* compared to the baselines across the benchmarks and power traces I evaluate; I find that *REACT* improves performance over the equally-reactive 770 μF buffer by an average of 39.1%, over the equal-capacity 17 mF buffer by 19.3%, and over the next-best-efficient 10 mF buffer by 18.8%. Compared to Morphy, *REACT* improves aggregate performance by 26.2%—demonstrating the necessity of *REACT*'s bank isolation approach and boosting performance where the state of the art underperforms static approaches. In extreme cases where the system is *always* operating in an energy surplus or deficit, *REACT*'s quiescent power consumption causes it to underperform suitable static buffers. In the common case, however, volatile power conditions expose the latency, longevity, and efficiency-related shortcomings of static buffer designs and expose the value of *REACT*'s efficient variable-capacitance approach.

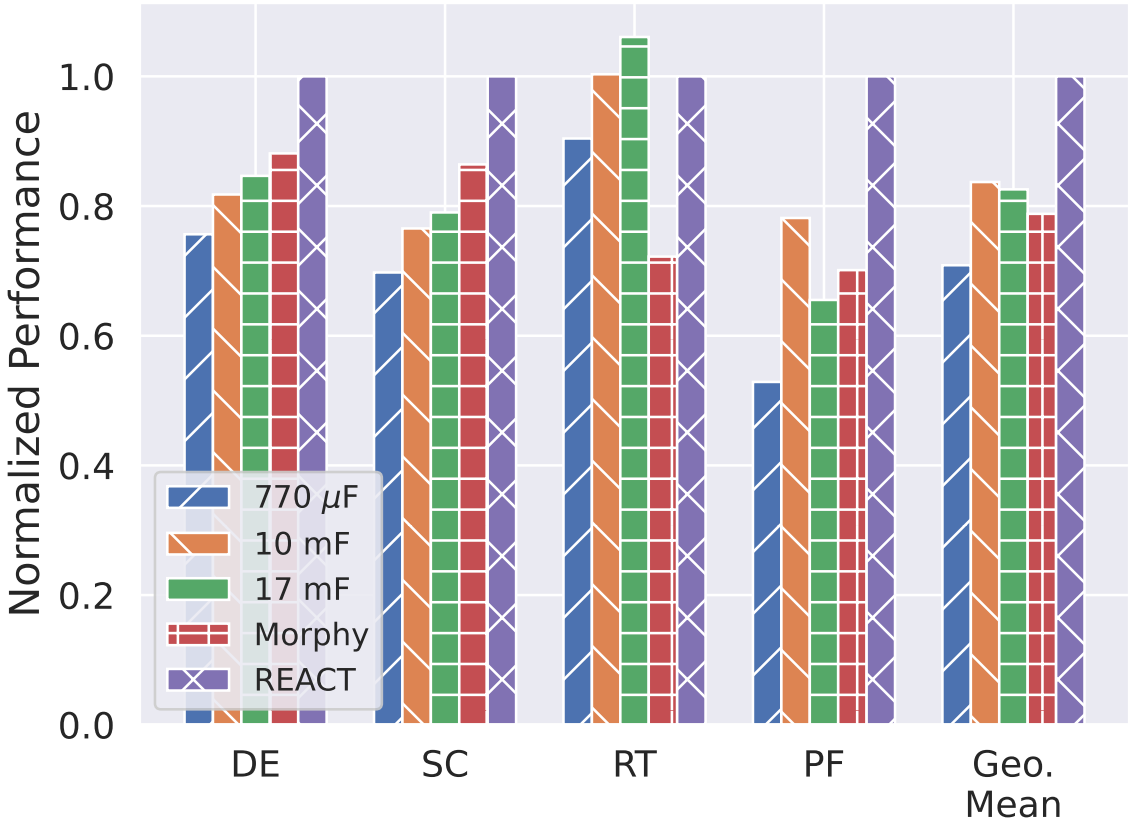


Figure 4.8: Average buffer performance quantified by figures of merit across power traces for each benchmark, normalized to *REACT*.

4.6 Conclusion

Batteryless systems operate on unreliable and volatile power, but use fixed-size buffers which waste energy and functionally limit systems when allocated capacity is a poor fit for short-term power dynamics. *REACT* stores energy in a fabric of reconfigurable capacitor banks, varying equivalent capacitance according to current energy supply and demand—adding capacitance to capture surplus power and reclaiming energy from excess capacitance. *REACT*'s energy-adaptive approach maximizes reactivity and capacity to ensure all incoming energy is captured and efficiently delivered to sensing, computing, and communication devices. My

hardware evaluation on real-world power traces shows that *REACT* reduces system latency by an average of 7.7x compared to an equivalent-sized static buffer and improves throughput by an average of 25.6% over *any* static buffer system, while incorporating software direction allows *REACT* to provide flexible task longevity guarantees. Compared to state-of-the-art switched capacitor systems, *REACT*'s efficient switching architecture improves performance by an average of 26.2%.

REACT's runtime-configurable buffering technique eliminates the tradeoff between system latency and longevity, and affords designers greater control over how batteryless devices respond to incoming power. My results indicate that energy-responsive reconfiguration of hardware is an effective approach to both maximizing energy efficiency and system functionality, opening the door for future work leveraging energy-adaptive hardware and reconfiguration.

Chapter 5

Conclusion and Future Work

Energy harvesting systems have the potential to revolutionize mobile computing by freeing high-performance, ultra-low-power digital systems from the batteries that hold them back. While today's batteryless systems come with fundamental drawbacks that currently preclude widespread deployment, this thesis provides the tools to overcome three major obstacles to practical and performant batteryless systems. *TotalRecall* brings high-performance intermittent operation to tens of thousands of platforms by removing the requirement for systems to integrate high-performance non-volatile memory. *Failure Sentinels* and *REACT* introduce hardware peripherals and platforms designed *from the ground up* for efficient batteryless operation, showing the potential of hardware tailor-made for batteryless operation. These platforms lay the groundwork for and guide future research and development towards truly effective, next-generation batteryless computing systems.

5.1 Future Work

While the research I describe above addresses major obstacles on the way towards a fully batteryless future, it also exposes new challenges for mobile computing systems. Here, I describe several research directions inspired by the work above.

5.1.1 Extending SRAM-based Batteryless Computing

TotalRecall lays the foundation for ubiquitous *NVM-invariant* intermittent computing, but challenges remain. *TotalRecall* inherits the limitations of just-in-time checkpointing systems (§1.1), including the need for hardware support as well as the inability to support atomic operations. In-place checkpoints complicate simple extensions of automatic (i.e., programmer- or compiler-directed) systems, as *any* program execution invalidates the checksum verifying SRAM's integrity—no "known-good" version of program state exists. As even the state of the art just-in-time systems depend on rollback mechanisms developed in automatic checkpointing systems [95], developing these techniques for SRAM-based checkpointing is crucial to make the system truly practical.

To that end, I plan to develop programming models and compiler frameworks that enable *partial state checkpointing* for SRAM-based intermittent computing. As with previous automatic checkpointing techniques, the primary technical challenge will be balancing checkpointing and re-execution overhead—with the added obstacle of minimizing the amount of modified state between checkpoints (to reduce the number of checksums recalculated throughout execution). I will engage with programming language and compiler experts to see how best to overcome these technical hurdles to ubiquitous SRAM-based intermittent computation.

5.1.2 Persistent Peripherals for Intermittent Systems

While a great deal of work has been done to ensure intermittent software execution on batteryless microcontrollers is correct and efficient, the peripheral devices (e.g., sensors, radios, memories) connected to these systems have received comparatively little attention. Sound peripheral operation is essential to batteryless systems, as many projected deployments focus on sensing and data transmission using on-chip peripheral devices.

I envision two major research thrusts targeting intermittent peripheral operation. The first thrust leverages the observation that many low-power peripherals 1) operate at lower minimum supply voltages than the processors controlling them, and 2) are capable of software-free operation by either continuing independently (e.g., a real-time clock) or by buffering samples in anticipation of the processor eventually powering back on (e.g., an analog-to-digital converter). The major research challenge here is detecting when peripheral devices correctly continued operation *without* software knowledge; to this end, I will develop on-chip hardware systems that operate at these low voltage levels to inform software of when peripheral operation continued.

While many peripheral devices can correctly continue operation at low voltages without modification, fully intermittent systems will ultimately require fully intermittent peripheral devices. The complexity of such devices varies significantly, ranging from simple programmable analog/digital converters consisting of only shift registers and analog hardware to complex radio transceivers with integrated microcontrollers. My second research thrust will explore modifying these systems to work on intermittent power by developing new hardware approaches and protocol designs to reliably and correctly interface with other devices both on- and off-chip.

5.1.3 Systems and Architectures for Emerging Memories

Intermittent systems demonstrate both the potential and current limitations of emerging non-volatile memories such as Ferroelectric RAM (FRAM) and Magnetoresistive RAM (MRAM) (collectively referred to as Non-Volatile RAMs, or NVRAMs). Compared to the previous generation of NVMs, these memories enable low-power, high-speed, and high-endurance writes—at the cost of reduced read performance, causing FRAM and MRAM

based systems to incur a significant energy and time penalty during common-case code execution. Re-imagining system design around the unique tradeoffs exhibited by these memories is a promising avenue for increased performance, both for batteryless and traditional embedded systems.

I envision research spanning the system stack to support these memories. Today's NVRAM-based systems include a blend of NVRAM and SRAM, while NVRAM's flexibility allows it to store both code and data—freeing up higher-performance but scarcer SRAM to serve the most-accessed information. I first intend to implement this "intelligent caching" support for hybrid NVRAM/SRAM systems in software to target current platforms, then extend my work into the hardware domain to bring NVRAM platforms up to and beyond the performance of more mature memory technologies.

Bibliography

[1] Saad Ahmed, Qurat ul Ain, Junaid Haroon Siddiqui, Luca Mottola, and Muhammad Hamad Alizai. Intermittent computing with dynamic voltage and frequency scaling. In *Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks*, EWSN '20, page 97–107, USA, 2020. Junction Publishing. ISBN 9780994988645.

[2] Miran Alhaideri, Michael Rushanan, Denis Foo Kune, and Kevin Fu. The moo and cement shoes: Future directions of a practical sense-control-actuate application, September 2013. URL `http://terraswarm.org/pubs/111.html`. Presented at First International Workshop on the Swarm at the Edge of the Cloud (SEC'13 @ ESWeek), Montreal.

[3] M. Alioto and G. Palumbo. Impact of supply voltage variations on full adder delay: Analysis and comparison. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(12):1322–1335, 2006. doi: 10.1109/TVLSI.2006.887809.

[4] Anon. Rf traces, October 2022. `https://anonymous.4open.science/r/rf_traces-4B3E/README.md`.

[5] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The

Rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[6] Abu Bakar, Alexander G. Ross, Kasim Sinan Yildirim, and Josiah Hester. Rehash: A flexible, developer focused, heuristic adaptation platform for intermittently powered computing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(3), sep 2021. doi: 10.1145/3478077. URL `https://doi.org/10.1145/3478077`.

[7] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. 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, March 2016.

[8] Domenico Balsamo, Alex Weddell, Geoff Merrett, Bashir Al-Hashimi, Davide Brunelli, and Luca Benini. Hibernus: Sustaining Computation during Intermittent Supply for Energy-Harvesting Systems. In *IEEE Embedded Systems Letters*, 2014.

[9] N. A. Bhatti and L. Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 209–220, 2017.

[10] M. Bhushan and M. B. Ketchen. Generation, elimination and utilization of harmonics in ring oscillators. In *2010 International Conference on Microelectronic Test Structures (ICMTS)*, pages 108–113, 2010. doi: 10.1109/ICMTS.2010.5466847.

[11] M. Bhushan, A. Gattiker, M. B. Ketchen, and K. K. Das. Ring oscillators for cmos process tuning and variability control. *IEEE Transactions on Semiconductor Manufacturing*, 19(1):10–18, 2006.

[12] James Blackman. What is mmtc in 5g nr, and

how does it impact nb-iot and lte-m, October 2019. https://enterpriseiotinsights.com/20191016/channels/fundamentals/what-is-mmtc-in-5g-nr-and-how-does-it-impact-nb-iot-and-lte-m.

[13] J. Blank and K. Deb. Pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[14] Michael Buettner, Richa Prasad, Alanson Sample, Daniel Yeager, Ben Greenstein, Joshua R. Smith, and David Wetherall. Rfid sensor networks with the intel wisp. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, page 393–394, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595939906. doi: 10.1145/1460412.1460468. URL `https://doi.org/10.1145/1460412.1460468`.

[15] Michael Buettner, Ben Greenstein, and David Wetherall. Dewdrop: An energy-aware runtime for computational rfid. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, page 197–210, USA, 2011. USENIX Association.

[16] Thomas D Burd, Trevor A Pering, Anthony J Stratakos, and Robert W Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of solid-state circuits*, 35(11):1571–1580, 2000.

[17] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu. New paradigm of predictive mosfet and interconnect modeling for early circuit simulation. In *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference (Cat. No.00CH37044)*, pages 201–204, 2000.

[18] T. Chan, P. Gupta, A. B. Kahng, and L. Lai. Ddro: A novel performance monitoring

methodology based on design-dependent ring oscillators. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pages 633–640, 2012.

[19] J. Choi, H. Joe, Y. Kim, and C. Jung. Achieving stagnation-free intermittent computation with boundary-free adaptive execution. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, pages 331–344, April 2019.

[20] Jongouk Choi, Qingrui Liu, and Changhee Jung. Cospec: Compiler directed speculative intermittent computation. In *International Symposium on Microarchitecture*, MICRO, pages 399–412, 2019.

[21] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. Compiler-directed high-performance intermittent computation with power failure immunity. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–54, 2022. doi: 10.1109/RTAS54340.2022.00012.

[22] Jongouk Choi, Jianping Zeng, Dongyoon Lee, Changwoo Min, and Changhee Jung. Write-light cache for energy harvesting systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700958. doi: 10.1145/3579371.3589098. URL `https://doi.org/10.1145/3579371.3589098`.

[23] Alexei Colin and Brandon Lucia. Chain: Tasks and channels for reliable intermittent programs. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA, pages 514–530, October 2016.

[24] Alexei Colin, Emily Ruppel, and Brandon Lucia. 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, page 767–781, New York, NY, USA, 2018. Association for

Computing Machinery. ISBN 9781450349116. doi: 10.1145/3173162.3173210. URL https://doi.org/10.1145/3173162.3173210.

[25] Abracon Corporation. AB08X5 Real-Time Clock Family, October 2014. https://abracon.com/Precisiontiming/AB08X5-RTC.PDF.

[26] M. Danesh, S. T. Chandrasekaran, and A. Sanyal. Ring oscillator based delta-sigma adcs. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 113–116, 2018.

[27] John H. Davies. *MSP430 Microcontroller Basics*. Elsevier Ltd., 1 edition, 2008. ISBN 0750682760.

[28] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. Reliable timekeeping for intermittent computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 53–67, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378464. URL https://doi.org/10.1145/3373376.3378464.

[29] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. Battery-free game boy. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(3), sep 2020. doi: 10.1145/3411839. URL https://doi.org/10.1145/3411839.

[30] Bradley Denby and Brandon Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 939–954, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378473. URL https://doi.org/10.1145/3373376.3378473.

[31] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[32] H. Desai and B. Lucia. A power-aware heterogeneous architecture scaling model for energy-harvesting computers. *IEEE Computer Architecture Letters*, 19(1):68–71, 2020.

[33] Analog Devices. 0.5 Ω CMOS 1.65 V to 3.6 V Dual SPDT/2:1 Mux in Mini LFCSP Package, March 2012. `https://www.analog.com/media/en/technical-documentation/data-sheets/adg824.pdf`.

[34] Analog Devices. Micropower, 3-Axis, ±2g/±4g/±8gDigitalOutput MEMS Accelerometer, 2015. `https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf`.

[35] Analog Devices. Ltspice, September 2020. `https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html`.

[36] Knowles Electronics. SPU0414HR5H-SB, December 2012. `https://www.mouser.com/datasheet/2/218/knowles_01232019_SPU0414HR5H_SB-1891952.pdf`.

[37] K. Ganesan, J. San Miguel, and N. Enright Jerger. The what's next intermittent computing architecture. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA, pages 211–223, Feb 2019.

[38] G. Ge, C. Zhang, G. Hoogzaad, and K. A. A. Makinwa. A single-trim cmos bandgap reference with a 3σ inaccuracy of $\pm 0.15\%$ from -40°c to 125°c. *IEEE Journal of Solid-State Circuits*, 46(11):2693–2701, 2011.

[39] Graham Gobieski, Amolak Nagi, Nathan Serafin, Mehmet Meric Isgenc, Nathan Beckmann, and Brandon Lucia. Manic: A vector-dataflow architecture for ultra-low-

power embedded systems. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 670–684, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369381. doi: 10.1145/3352460.3358277. URL https://doi.org/10.1145/3352460.3358277.

[40] M. Gorlatova, A. Wallwater, and G. Zussman. Networking low-power energy harvesting devices: Measurements and algorithms. In *2011 Proceedings IEEE INFOCOM*, pages 1602–1610, 2011. doi: 10.1109/INFCOM.2011.5934952.

[41] Peter Gutmann. Data remanence in semiconductor devices. In *USENIX Security Symposium*, USENIX Security, August 2001.

[42] V. Gutnik and A. Chandrakasan. An efficient controller for variable supply-voltage low power processing. In *1996 Symposium on VLSI Circuits. Digest of Technical Papers*, pages 158–159, 1996.

[43] M. R. Halesh, K. R. Rasane, and H. Rohini. Design and implementation of voltage control oscillator (vco) using 180nm technology. In Srija Unnikrishnan, Sunil Surve, and Deepak Bhoir, editors, *Advances in Computing, Communication and Control*, pages 472–478, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-18440-6.

[44] Wang Song Hao and Ronald Garcia. Development of a digital and battery-free smart flowmeter. *Energies*, 7(6):3695–3709, 2014. ISSN 1996-1073. doi: 10.3390/en7063695. URL https://www.mdpi.com/1996-1073/7/6/3695.

[45] C. Helfmeier, C. Boit, D. Nedospasov, and J. P. Seifert. Cloning physically unclonable functions. In *International Symposium on Hardware-Oriented Security and Trust*, HOST, pages 1–6, June 2013.

[46] M. Hempstead, N. Tripathi, P. Mauro, Gu-Yeon Wei, and D. Brooks. An ultra low power system architecture for sensor network applications. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 208–219, 2005. doi: 10.1109/ISCA.2005.12.

[47] Mark Hempstead, Gu-Yeon Wei, and David Brooks. Architecture and circuit techniques for low-throughput, energy-constrained systems across technology generations. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, page 368–378, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595935436. doi: 10.1145/1176760.1176805. URL https://doi.org/10.1145/1176760.1176805.

[48] Josiah Hester and Jacob Sorber. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354592. doi: 10.1145/3131672.3131674. URL https://doi.org/10.1145/3131672.3131674.

[49] Josiah Hester, Timothy Scott, and Jacob Sorber. Ekho: Realistic and repeatable experimentation for tiny energy-harvesting sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, page 330–331, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450331432. doi: 10.1145/2668332.2668382. URL https://doi.org/10.1145/2668332.2668382.

[50] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *ACM Conference on Embedded Networked Sensor Systems*, SenSys, pages 5–16, 2015.

[51] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin

Fu, Wayne P. Burleson, and Jacob Sorber. Persistent clocks for batteryless sensing devices. *ACM Transactions on Embedded Computer Systems*, 15(4):77:1–77:28, August 2016.

[52] Matthew Hicks. Clank: Architectural support for intermittent computation. In *International Symposium on Computer Architecture*, ISCA, pages 228–240, 2017.

[53] D. E. Holcomb, W. P. Burleson, and K. Fu. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, September 2009.

[54] Daniel E. Holcomb, Amir Rahmati, Mastooreh Salajegheh, Wayne P. Burleson, and Kevin Fu. Drv-fingerprinting: Using data retention voltage of sram cells for chip identification. In *Proceedings of the 8th International Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec, pages 165–179, 2012.

[55] G. Huang, L. Qian, S. Saibua, D. Zhou, and X. Zeng. An efficient optimization based method to evaluate the drv of sram cells. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(6):1511–1520, June 2013.

[56] IBM. Combating fraud with blockchain and crypto-anchors, 2018. URL `https://www.research.ibm.com/5-in-5/crypto-anchors-and-blockchain/`.

[57] Fraunhofer IIS. RFicient Basic, Ultra-Low-Power WakeUp Receiver, January 2019. `https://www.iis.fraunhofer.de/content/dam/iis/en/doc/il/ics/ic-design/Datenblaetter/Factsheet_WakeUp_v4.pdf`.

[58] Texas Instruments. MSP430 competitive benchmarking, July 2006. `https://people.eecs.berkeley.edu/~boser/courses/40/labs/docs/microcontroller%20benchmarks.pdf`.

[59] Texas Instruments. MSP430L092—MSP430L092, MSP430C09x Mixed-Signal Microcontrollers, September 2010. http://www.ti.com/lit/ds/symlink/msp430l092.pdf.

[60] Texas Instruments. MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller datasheet (Rev. J), 2013. http://www.ti.com/lit/ds/symlink/msp430g2553.pdf.

[61] Texas Instruments. MSP430x2xx Family User's Guide (Rev. J), 2013. http://www.ti.com/lit/ug/slau144j/slau144j.pdf.

[62] Texas Instruments. Msp430g2x52, msp430g2x12 mixed signal microcontroller datasheet (rev. g), May 2013. https://www.ti.com/lit/ds/symlink/msp430g2252.pdf.

[63] Texas Instruments. FRAM FAQs, January 2014. http://www.ti.com/lit/ml/slat151/slat151.pdf.

[64] Texas Instruments. ADS7042 Ultra-LowPower, Ultra-Small Size, 12-Bit, 1-MSPS, SAR ADC, 2015. https://www.ti.com/lit/ds/sbas608c/sbas608c.pdf.

[65] Texas Instruments. MSP430F5438A—MSP430F543xA, MSP430F541xA Mixed-Signal Microcontrollers, September 2018. http://www.ti.com/lit/ds/symlink/msp430f5438a.pdf.

[66] Texas Instruments. MSP430FR5964—MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers, August 2018. http://www.ti.com/lit/ds/symlink/msp430fr5964.pdf.

[67] Texas Instruments. MSP430FR596x, MSP430FR594x Mixed-Signal Microcontrollers, 2018. https://www.ti.com/lit/ds/symlink/msp430fr5969.pdf.

[68] Texas Instruments. MSP430FR698x(1), MSP430FR598x(1) Mixed-Signal Microcontrollers, 2018. http://www.ti.com/lit/ds/symlink/msp430fr6989.pdf.

[69] Texas Instruments. CRC Implementation with MSP430™MCUs, 2018. http://www.ti.com/lit/an/slaa221a/slaa221a.pdf.

[70] Texas Instruments. MSP430 Flash Memory Characteristics (Rev. B), 2018. http://www.ti.com/lit/an/slaa334b/slaa334b.pdf.

[71] Texas Instruments. Lm66100 5.5-v, 1.5-a 79-milliohm, low iq ideal diode with input polarity protection, June 2019. https://www.ti.com/lit/ds/symlink/lm66100.pdf.

[72] Texas Instruments. MSP430 GCC, June 2019. http://www.ti.com/lit/ug/slau646e/slau646e.pdf.

[73] Texas Instruments. Msp432p401r, msp432p401m simplelink mixed-signal microcontrollers, June 2019. https://www.ti.com/lit/ds/symlink/msp432p401r.pdf.

[74] Texas Instruments. MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers, January 2021. https://www.ti.com/lit/ds/symlink/msp430fr5994.pdf.

[75] Texas Instruments. Tlv703x and tlv704x small-size, nanopower, low-voltage comparators, July 2021. https://www.ti.com/lit/ds/symlink/tlv7044.pdf.

[76] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Hypnos: an ultra-low power sleep mode with sram data retention for embedded microcontrollers. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330510. doi: 10.1145/2656075.2656089. URL https://doi.org/10.1145/2656075.2656089.

[77] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers. In *International Conference on VLSI Design and International Conference on Embedded Systems*, 2014.

[78] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Energy-Aware Memory Mapping for Hybrid FRAM-SRAM MCUs in IoT Edge Devices. In *International Conference on VLSI Design and International Conference on Embedded Systems*, VLSID, 2016.

[79] A. Jayaraj, M. Danesh, S. T. Chandrasekaran, and A. Sanyal. Highly digital second-order $\delta\sigma$ vco adc. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66 (7):2415–2425, 2019.

[80] B. E. Jonsson. A survey of A/D-converter performance evolution. In *International Conference on Electronics, Circuits, and Systems*, ICECS, pages 766–769, Dec 2010.

[81] Goran Jovanović, Mile Stojcev, and Zoran Stamenkovic. A cmos voltage controlled ring oscillator with improved frequency stability. *Scientific Publications of the State University of Novi Pazar: Applied Mathematics, Informatics, and Mechanics*, 2:1–9, 06 2010.

[82] Raja Jurdak, Antonio G. Ruzzelli, Gregory M. P. O'hare, and C. V. Lopes. Mote-based underwater sensor networks: Opportunities, challenges, and guidelines, 2008.

[83] Joseph Kahn, Randy Katz, and Kristofer Pister. Next Century Challenges: Mobile Networking for "Smart Dust". In *Conference on Mobile Computing and Networking (MobiCom)*, 1999.

[84] Udo Karthaus and Martin Fischer. Fully Integrated Passive UHF RFID Transponder

IC With 16.7-μ W Minimum RF Input Power. In *IEEE Journal of Solid-State Circuits, Volume 38*, 2003.

[85] Kemet. Supercapacitors fm series, July 2020. `https://www.mouser.com/datasheet/2/212/1/KEM_S6012_FM-1103835.pdf`.

[86] P. Koopman and T. Chakravarty. Cyclic redundancy code (crc) polynomial selection for embedded networks. In *International Conference on Dependable Systems and Networks, 2004*, pages 145–154, June 2004.

[87] Madhusudan Kulkarni and Kalmeshwar Hosur. Design of a linear and wide range current starved voltage controlled oscillator for pll. *International Journal on Cybernetics & Informatics*, 2:23–30, 02 2013. doi: 10.5121/ijci.2013.2104.

[88] R. Kumar and V. Kursun. Modeling of temperature effects on nano-cmos devices with the predictive technologies. In *2007 50th Midwest Symposium on Circuits and Systems*, pages 694–697, 2007.

[89] Silicon Labs. EFM32 Gecko Family EFM32WG Data Sheet, December 2021. `https://www.silabs.com/documents/public/data-sheets/efm32wg-datasheet.pdf`.

[90] Q. Liu, C. Jung, D. Lee, and D. Tiwari. Compiler-Directed Lightweight Checkpointing for Fine-Grained Guaranteed Soft Error Recovery. SC, pages 228–239, November 2016.

[91] Brandon Lucia and Benjamin Ransford. A simpler, safer programming and execution model for intermittent systems. In *Conference on Programming Language Design and Implementation*, PLDI, pages 575–585, 2015.

[92] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan. Architecture exploration for ambient energy harvesting nonvolatile

processors. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA, pages 526–537, Feb 2015.

[93] Kaisheng Ma, Xueqing Li, Karthik Swaminathan, Yang Zheng, Shuangchen Li, Yongpan Liu, Yuan Xie, John Sampson, and Vijaykrishnan Narayanan. Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power. In *IEE Micro Volume 36, Issue 3*, 2016.

[94] Kiwan Maeng and Brandon Lucia. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *USENIX Conference on Operating Systems Design and Implementation*, OSDI, pages 129–144, November 2018.

[95] Kiwan Maeng and Brandon Lucia. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, pages 1101–1116, 2019.

[96] Kiwan Maeng and Brandon Lucia. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 1005–1021, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376136. doi: 10.1145/3385412.3385998. URL `https://doi.org/10.1145/3385412.3385998`.

[97] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: Intermittent execution without checkpoints. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA, pages 96:1–96:30, October 2017.

[98] M. K. Mandal and B. C. Sarkar. Ring oscillators: Characteristics and applications. *Indian Journal of Pure & Applied Physics*, 48(1):136–145, 2010.

[99] Microchip. PIC16(L)F15356/75/76/85/86, 2016. http://ww1.microchip.com/downloads/en/devicedoc/40001866a.pdf.

[100] Microchip. PIC18(L)F65/66K40, November 2017. https://ww1.microchip.com/downloads/en/DeviceDoc/40001842D.pdf.

[101] Microsemi. ZL70251 Ultra-Low-Power Sub-GHz RF Transceiver, March 2018. https://www.microsemi.com/document-portal/doc_view/132900-zl70251-datasheet.

[102] A. Mirhoseini, E. M. Songhori, and F. Koushanfar. Idetic: A high-level synthesis approach for enabling long computations on transiently-powered ASICs. In *International Conference on Pervasive Computing and Communications*, PerCom, pages 216–224, March 2013.

[103] Murata. GRM31CR60J227ME11L Chip Monolithic Ceramic Capacitor for General. https://search.murata.co.jp/Ceramy/image/img/A01X/G101/ENG/GRM31CR60J227ME11-01.pdf.

[104] Murata. Supercapacitors FM Series, July 2020. https://www.mouser.com/datasheet/2/212/1/KEM_S6012_FM-1103835.pdf.

[105] Phillip Nadeua, Dina El-Damaj, Deal Glettig, Yong Lin Kong, Stacy Mo, Cody Cleveland, Lucas Booth, Niclas Roxhed, Robert Langer, Anantha P. Chandrakasan, and Giovanni Traverso. Prolonged energy harvesting for ingestible devices. *Nature Biomedical Engineering*, 1(0022), Feb 2017.

[106] Nichicon. ALUMINUM ELECTROLYTIC CAPACITORS. https://www.nichicon.co.jp/english/products/pdfs/e-kl.pdf.

[107] University of Washington. WISP 5 GitHub, April 2014. http://www.github.com/wisp/wisp5.

[108] Yossef Oren, Ahmad-Reza Sadeghi, and Christian Wachsmann. On the effectiveness of the remanence decay side-channel to clone memory-based PUFs. In *International Conference on Cryptographic Hardware and Embedded Systems*, CHES, pages 107–125, August 2013.

[109] Panasonic. Panasonic coin type lithium batteries, August 2005. `https://datasheet.octopart.com/CR1616-Panasonic-datasheet-9751741.pdf`.

[110] J. Park and J. A. Abraham. A fast, accurate and simple critical path monitor for improving energy-delay product in dvs systems. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 391–396, 2011.

[111] S. Park, C. Min, and S. Cho. A 95nw ring oscillator-based temperature sensor for rfid tags in 0.13um cmos. In *2009 IEEE International Symposium on Circuits and Systems*, pages 1153–1156, 2009. doi: 10.1109/ISCAS.2009.5117965.

[112] Powercast. P2110B 915 MHz RF Powerharvester Receiver, December 2016. `https://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf`.

[113] Powercast. TX91501B – 915 MHz Powercaster Transmitter, October 2019. `https://www.powercastco.com/wp-content/uploads/2019/10/User-Manual-TX-915-01B-Rev-A-1.pdf`.

[114] Powercast. 915 mhz dipole antenna datasheet, November 2020. `https://www.powercastco.com/wp-content/uploads/2020/11/DA-915-01-Antenna-Datasheet_new_web.pdf`.

[115] P. Prabha, S. J. Kim, K. Reddy, S. Rao, N. Griesert, A. Rao, G. Winter, and P. K.

Hanumolu. A highly digital vco-based adc architecture for current sensing applications. *IEEE Journal of Solid-State Circuits*, 50(8):1785–1795, 2015.

[116] Huifang Qin, Yu Cao, Dejan Markovic, Andrei Vladimirescu, and Jan Rabaey. Sram leakage suppression by minimizing standby supply voltage. In *International Symposium on Quality Electronic Design*, ISQED, pages 55–60, 2004.

[117] Qingrui Liu and Changhee Jung. Lightweight Hardware Support for Transparent Consistency-Aware Checkpointing in Intermittent Energy-Harvesting systems. In *Non-Volatile Memory Systems and Applications Symposium*, NVMSA, August 2016.

[118] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne Burleson, and Kevin Fu. TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks. In *USENIX Security Symposium*, 2012.

[119] Y. K. Ramadass and A. P. Chandrakasan. An efficient piezoelectric energy harvesting interface circuit using a bias-flip rectifier and shared inductor. *IEEE Journal of Solid-State Circuits*, 45(1):189–204, 2010.

[120] Benjamin Ransford and Brandon Lucia. Nonvolatile Memory is a Broken Time Machine. In *Workshop on Memory Systems Performance and Correctness*, 2014.

[121] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System Support for Long-Running Computation on RFID-Scale Devices. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[122] Andres Georg Rosch, Andre Gall, Silas Aslan, Matthias Hecht, Leonard Franke, Md. Mofasser Malick, Lara Penth, Daniel Bahro, Daniel Friderich, and Uli Lemmer.

Fully printed origami thermoelectric generators for energy-harvesting. *Flexible Electronics*, 2021.

[123] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith. Design of an rfid-based battery-free programmable sensing platform. *IEEE Transactions on Instrumentation and Measurement*, 57(11):2608–2615, Nov 2008.

[124] A. Sanyal, S. Li, and N. Sun. Low-power scaling-friendly ring oscillator based $\sigma\delta$ adc. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

[125] Fred Schlachter. No moore's law for batteries, April 2013. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3619319/`.

[126] ROHM Semiconductor. Low Voltage Standard CMOS Voltage Detector ICs, October 2021. `https://fscdn.rohm.com/en/products/databook/datasheet/ic/power/voltage_detector/bu48xxg-e.pdf`.

[127] Henry Sodano, Gyuhae Park, and Daniel Inman. Estimation of Electric Charge Output for Piezoelectric Energy Harvesting. In *Strain, Volume 40*, 2004.

[128] ST. Small signal schottky diode, October 2001. `https://www.st.com/content/ccc/resource/technical/document/datasheet/group1/11/76/e4/a3/df/07/49/14/CD00000767/files/CD00000767.pdf/jcr:content/translations/en.CD00000767.pdf`.

[129] Pico Technology. Picoscope 2000 series, 2016. `https://www.picotech.com/download/datasheets/picoscope-2000-series-data-sheet-en.pdf`.

[130] TestEquity. Model 123H Temperature/Humidity Chamber With F4 Controller and

EZ-Zone Limit Operation and Service Manual, 2010. https://www.testequity.com/UserFiles/documents/pdfs/123Hman.pdf.

[131] Priya Thanigai. Frams as alternatives to flash memory in embedded designs, July 2011. https://www.embedded.com/design/mcus-processors-and-socs/4390688/2/FRAMs-as-alternatives-to-flash-memory-in-embedded-designs.

[132] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De. Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In *2009 Symposium on VLSI Circuits*, pages 112–113, 2009.

[133] Voltaic. Voltaic systems p121 r1g, April 2020. https://voltaicsystems.com/content/VoltaicSystemsP121R1G.pdf.

[134] Harrison Williams, Xun Jian, and Matthew Hicks. Forget failure: Exploiting sram data remanence for low-overhead intermittent computation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 69–84, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378478. URL https://doi.org/10.1145/3373376.3378478.

[135] Harrison Williams, Michael Moukarzel, and Matthew Hicks. Failure sentinels: Ubiquitous just-in-time intermittent computation via low-cost hardware support for voltage monitoring. In *International Symposium on Computer Architecture*, ISCA, pages 665–678, 2021.

[136] Lynnette Reese Wolfgang Lutsch. Energy optimization tools for ultra-low-power microcontrollers, 2020. URL https://www.mouser.com/applications/low-power-ewc-design/.

[137] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *USENIX Symposium on Operating Systems Design and Implementation*, OSDI, pages 17–32, November 2016.

[138] X. Wu, I. Lee, Q. Dong, K. Yang, D. Kim, J. Wang, Y. Peng, Y. Zhang, M. Saliganc, M. Yasuda, K. Kumeno, F. Ohno, S. Miyoshi, M. Kawaminami, D. Sylvester, and D. Blaauw. A 0.04mm316nw wireless and batteryless sensor system with integrated cortex-m0+ processor and optical communication for cellular temperature measurement. In *2018 IEEE Symposium on VLSI Circuits*, pages 191–192, 2018.

[139] Yu-Chi Wu, Pei-Fan Chen, Zhi-Huang Hu, Chao-Hsu Chang, Gwo-Chuan Lee, and Wen-Ching Yu. A Mobile Health Monitoring System Using RFID Ring-Type Pulse Sensor. In *Conference on Dependable, Autonomic, and Secure Computing (DASC)*, 2009.

[140] Xilinx. 7 series fpgas data sheet: Overview, 2018. URL `https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`.

[141] Xilinx. Artix-7 fpgas data sheet: Dc and ac switching characteristics, 2018. URL `https://www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf`.

[142] W. Xu, X. Chen, and J. Wu. An overview of theory and techniques for reducing ring oscillator supply voltage sensitivity in mixed-signal soc. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 2039–2042, 2011.

[143] Fan Yang, Ashok Samraj Thangarajan, Wouter Joosen, Christophe Huygens, Danny Hughes, Gowri Sankar Ramachandran, and Bhaskar Krishnamachari. Astar: Sustain-

able battery free energy harvesting for heterogeneous platforms and dynamic environments. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, EWSN '19, page 71–82, USA, 2019. Junction Publishing. ISBN 9780994988638.

[144] Fan Yang, Ashok Samraj Thangarajan, Sam Michiels, Wouter Joosen, and Danny Hughes. Morphy: Software defined charge storage for the iot. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 248–260, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390972. doi: 10.1145/3485730.3485947. URL https://doi.org/10.1145/3485730.3485947.

[145] Masayuki Yano, James Douglass Penn, George Konidaris, and Anthony T Patera. Interpolation, August 2013. https://ocw.mit.edu/courses/mechanical-engineering/2-086-numerical-computation-for-mechanical-engineers-fall-2014/nutshells-guis/MIT2_086F14_Interpolation.pdf.

[146] Jianping Zeng, Jongouk Choi, Xinwei Fu, Ajay Paddayuru Shreepathi, Dongyoon Lee, Changwoo Min, and Changhee Jung. Replaycache: Enabling volatile cachesfor energy harvesting systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 170–182, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385572. doi: 10.1145/3466752.3480102. URL https://doi.org/10.1145/3466752.3480102.

[147] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. Moo: A Batteryless Computational RFID and Sensing Platform. In *Technical Report UMCS-2011-020*, 2011.

[148] Yuchen Zhou, Jianping Zeng, Jungi Jeong, Jongouk Choi, and Changhee Jung. Sweepcache: Intermittence-aware cache on the cheap. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 1059–1074, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703294. doi: 10.1145/3613424.3623781. URL https://doi.org/10.1145/3613424.3623781.

[149] Kenneth M. Zick and John P. Hayes. Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems. *ACM Trans. Reconfigurable Technol. Syst.*, 5(1), March 2012. ISSN 1936-7406. doi: 10.1145/2133352.2133353. URL https://doi.org/10.1145/2133352.2133353.