UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Off and On Again: Modeling and Optimizing Intermittent Computing Systems

by

Sivert T. Sliper

ORCiD: 0000-0002-8991-3783

A thesis for the degree of
Doctor of Philosophy

November 2022

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Doctor of Philosophy

Off and On Again: Modeling and Optimizing Intermittent Computing Systems

by Sivert T. Sliper

Intermittent computing (IC) is a vital technology for realizing IoT at a vast scale. By harvesting energy from the environment, and leveraging non-volatile memory (NVM) to retain computational progress through power cycles, IC enables untethered and battery-free devices that perform incremental computation whenever ambient energy is available. The backbone of IC is NVM, and recent advances in energy-efficient byte-addressable NVM have the potential to expand the application domain of IC substantially. The tight interaction between energy and execution does, however, cause complexities that require specialized modeling, hardware design, and software design; complexities that hamper adoption. This thesis is focused on optimizing IC using software and hardware support leveraging energy-efficient NVM, and also on system-level modeling of IC systems to enable such research.

Supporting IC on commercially available microcontrollers is important to lower the barrier to entry, and for adoption of IC in real-world applications. However, current methods suffer from inefficient state retention. To address this issue, this thesis proposes *ManagedState*, a new page-based memory manager that tracks used and modified regions of memory to reduce the overhead of state retention by 26–87 %. To make IC as efficient, robust and useful as possible, however, software support alone is insufficient. Hardware support for IC is needed; and to research system-level hardware support for IC, electronic system-level modeling is the most reasonable approach. However, modeling IC presents unique challenges that current modeling tools do not support, most prominent of which is the tight interactions between energy and execution. To model IC systems, this thesis proposes a new full-system simulator, named *Fused*, that specifically models the interplay between energy and execution in IC devices. Using *Fused*, this thesis then explores new hardware support for IC that leverages energy-efficient and byte-addressable NVMs. The result is *MEMIC*, a memory architecture with hardware support that improves performance by 13–39 % compared to the state-of-art, and furthermore allows systems to operate under harsher conditions. *MEMIC* strives to combine volatile- and non-volatile memory in such a way that the operations of IC are as efficient as possible, while also maximizing computational performance per joule.

Contents

List of Figures

List of Tables

Definitions and Abbreviations

ADC	analog to digital converter
AES	advanced encryption standard
AMS	analog and mixed signal
API	application programmer's interface
CMOS	complementary metal oxide semiconductor
COTS	commercial off-the-shelf
CPU	central processing unit
DMA	direct memory access
EDA	electronic design automation
EH	energy harvesting
ESL	electronic system level
FASE	failure-atomic section
FIFO	first in first out
FRAM	ferroelectric random access memory
FPGA	field-programmable gate-array
GDB	GNU debugger
GPIO	general purpose input/output
IC	intermittent computing
ILPM	instruction level power modelling
IoT	internet of things
ISA	instruction set architecture

LFU	least frequently used
LRU	least recently used
NNLS	non-negative least squares
NVFF	non-volatile flip flop
NVM	non-volatile memory
NVP	non-volatile processor
MMU	memory management unit
MRAM	magnetoresistive random access memory
MTJ	magnetic tunnel junction
PCB	printed circuit board
OS	operating system
PCM	phase-change memory
PMM	power management module
QoS	quality of service
RAM	random access memory
RTL	register transfer level
SoC	system on chip
SRAM	static random access memory
STT-MRAM	spin transfer torque MRAM
SVS	supply voltage supervisor
TDF	timed data flow
TLM	transaction level modeling
UVM	unified verification methodology
VM	volatile memory

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: listed in § 1.3

Signed:..... Date:.....

Chapter 1

Introduction

The internet of things (IoT) is increasing demand for low-power computing devices to bridge between the cyber- and physical worlds with advanced monitoring capabilities [7]. These devices are beginning to be deployed in a wide variety of domains, such as smart buildings and cities, logistics, healthcare, remote surveillance, industrial automation and wearable computing [2]. However, despite widespread optimism for an IoT comprising tens of billions, or even a trillion, devices, progress has been slower than expected [39]. At such scale, computers need to be small and low-cost, with minimal environmental impact and maximal deployment lifetime. When tens of billions of connected devices are deployed[1], maintenance operations such as battery replacement or replenishment become practically and economically infeasible.

To extend maintenance-free deployment lifetime without requiring increased battery capacity, devices can instead be powered by energy harvesting (EH). EH devices can extract a small amount of electrical power from ambient energy sources such as light, thermal gradients, or mechanical motion/vibration for an indefinite period of time. However, EH is typically an unstable and unpredictable power source.

To cope with the dynamics of EH, energy-neutral devices [72] buffer harvested energy in a rechargeable battery or supercapacitor, and modulate their energy consumption to match harvested energy over the periodicity of the power source (e.g. 24 h for outdoor solar). Thus they can theoretically operate indefinitely. A typical method of adjusting the power consumption is to operate at a variable duty cycle, calculated based on the amount of harvested and stored energy. During periods of abundant energy harvest, an energy-neutral device can operate at a higher performance level, i.e. at higher duty cycle or clock frequency, and it can buffer some of the extra energy to make up for periods of diminished harvest. However, even rechargeable batteries and supercapacitors

[1]The Ericsson mobility report predicts that there will be 26 billion internet-connected devices by 2026, excluding PCs, laptops, tablets and mobile phones. Source: https://www.ericsson.com/en/mobility-report/mobility-visualizer

degrade and eventually expire after hundreds or thousands of charge cycles or years of operation, and, together with their charging circuitry, they often constitute the majority of the total cost of an IoT device.

Instead of relying on energy storage to compensate for EH dynamics, intermittent computing (IC) systems operate in an incremental manner, making progress whenever energy is available [103, 132]. They can be coupled directly to energy harvesters supplying as little as a few micro watts of power, without the use of batteries, because they retain computational progress despite frequently losing power. By enabling tiny long-life devices at low environmental and monetary cost, IC widens the application domain for IoT, thereby opening new opportunities for computation in environments that were previously too harsh and/or constrained. For example, researchers have proposed that IC can be used in micro-satellites [91], which experience potentially hundreds of kelvins of temperature variation, and where every cubic centimeter of volume counts.

Figure 1.1 depicts a typical IC system, comprising a low-power microcontroller, an energy harvester, and some power management circuitry. The microcontroller itself typically embeds volatile memory, which has fast and energy-efficient access but loses data if power is lost, and some non-volatile memory which is usually slower and less energy efficient, but retains data through power loss. The power management circuitry can perform a wide set of functions, such as voltage conversion, voltage detection etc., based on the specifics of the IC method. The energy harvester harvests energy from ambient sources, such as e.g. solar irradiation. Figure 1.2 illustrates intermittent operation by showing a possible supply voltage trace, whereby brief on-periods of execution are interleaved with off-periods where the supply voltage recovers. The key ability of IC is to retain state so that execution can continue from where it left off after a power failure. Various methods of retaining state have been proposed, as detailed in the background chapter of this thesis. In the pioneering work by Ransford et al. [116], computational state is saved periodically in a checkpoint. When the supply recovers after a power loss, the execution then resumes from the latest checkpoint.

A recent example of an application IC system, was demonstrated by Bing et al. [16]. The application is a batteryless and wireless bicycle trip counter comprising a sensor node attached to the wheel of a bicycle. To the fork of the bike, they attached a strong magnet. The sensor node embeds a copper coil that harvests a pulse of energy every time it passes the magnet as the wheel rotates. It then sends information, , wirelessly to a battery-powered bicycle computer. The transmitted information includes analysis computed through several power cycles (revolutions of the wheel). This is a good example of the usage of IC with a system mounted in a harsh environment (on the rim of a bicycle wheel) with stringent size and weight constraints. Furthermore, there is a symbiotic relation between the variable being measured (the rotation of the wheel) and the energy harvesting method: the data is only valuable while the wheel is rotating,

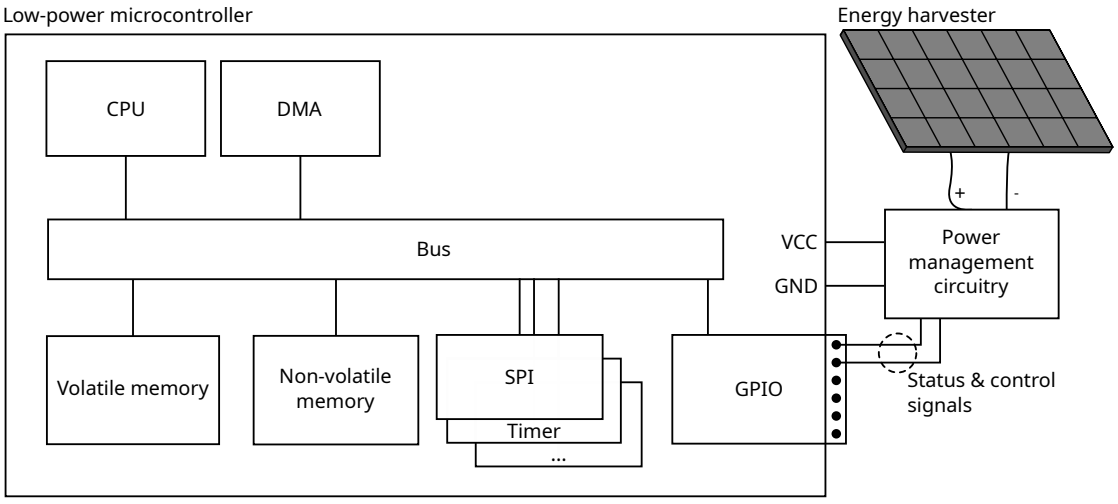


FIGURE 1.1: A typical intermittent computing system comprising a low-power microcontroller, some power management circuitry, and an energy harvester.

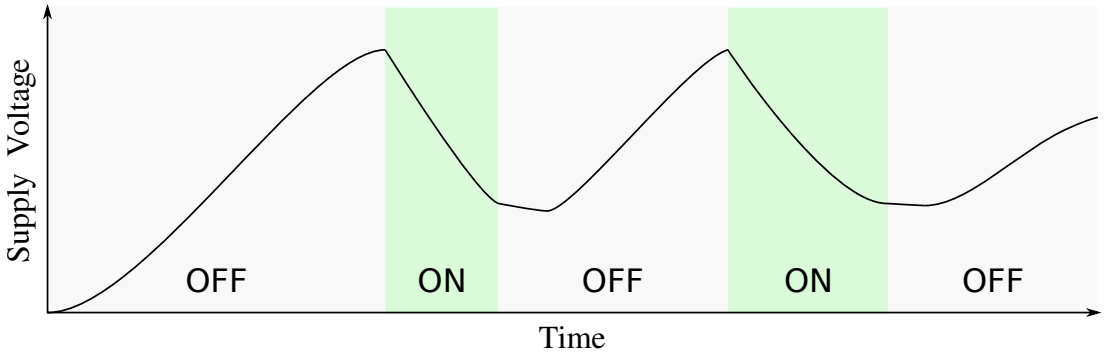


FIGURE 1.2: Illustration of intermittent operation. The device operates during brief on-periods (typically in the order of milliseconds), until its stored energy is depleted. It then remains off until the supply voltage recovers.

and that is also the only time when energy is harvested. Other notable applications of IC includes a batteryless step-counter [118] and a batteryless *Game Boy* clone [155].

1.1 Research justification

Most of the IC literature diverges into either purely software-based approaches that support IC on commercial off-the-shelf (COTS) microcontrollers [9, 68, 92, 116], or custom non-volatile processors (in which volatile logic is backed up in, or replaced by, non-volatile logic)[88, 89, 95, 123, 151].

Software solutions to IC generally aim to make the best use of existing hardware by implementing various ways to retain computational progress through power cycles. Providing support for IC through software is attractive because it can enable IC on existing devices with little development cost. Among methods proposed for IC, those

based on checkpointing have the best performance [96, 97, 98], and the best compatibility with existing software [8, 9, 12, 68, 77, 98]. Early checkpointing methods saved and restored the entirety of volatile state, i.e. CPU registers and the contents of volatile memory, in every power cycle [8, 9]. A straightforward optimization of this was to instead only checkpoint the *allocated* volatile memory, comprising the regions of memory allocated for the various data sections (`.data`, `.bss`, `.stack`) [12, 15]. However, during short on-periods, not all allocated state is used, because only a small portion of the application is executed, so loading the entirety of allocated state during boot is inefficient. Furthermore, when saving a new checkpoint at the end of an on-period, it is wasteful to write unmodified data to non-volatile memory (NVM), especially for NVMs with high write-energy [10, 43, 84, 148]. These factors motivate research into a checkpointing method that loads variables only when they are used, and also writes to NVM only data that have changed.

Software support for IC on COTS microcontrollers can only go so far. To achieve optimal performance, hardware support for core IC operations is necessary. Although hardware changes require substantially more development effort than software changes, they have greater potential for efficiency gains. Furthermore, the envisioned use cases for IC involve very large quantities of devices over which to amortize the initial cost of developing and fabricating new hardware platforms; the marginal cost likely outweighs initial development costs.

The backbone of IC is NVM. While initial IC research used flash memory as their NVM, it was soon demonstrated that the byte-addressable and more energy-efficient ferroelectric random access memory (FRAM) was a better candidate [68, 116]. Using a more suitable NVM meant that IC devices could operate using only the energy buffered in a few micro farads of capacitance already necessary for power conditioning purposes, as opposed to requiring supercapacitors to supply the checkpointing energy [9]. Today, magnetoresistive random access memory (MRAM) and phase-change memory (PCM) are emerging as replacements for flash in advanced process nodes for microcontrollers [24]. MRAM is available from several foundries, and in some commercially available microcontrollers [18, 50]. Like FRAM, MRAM is byte-addressable, but it is also more energy-efficient and scalable to advanced process nodes. It is still, however, not as energy-efficient as static random access memory (SRAM), which has orders of magnitude lower write-energy, and a fraction of the read-energy. As such, it is clear that MRAM-enabled conventional microcontrollers still benefit from using SRAM in addition to MRAM. For microcontrollers designed for IC, combining SRAM and NVM is a complex topic with many trade-offs.

Researchers in the closely related field of non-volatile processors (NVPs) tend to incorporate NVMs using a bottom-up approach. They start with new device-level innovations and use them to build a processor that implicitly retains state through power

loss. They either replace each volatile cell (flip-flop/SRAM) with its non-volatile counterpart, or use NVM-backed cells [89, 152]. The latter technique is designed to minimize the active power of the system by essentially adding a non-volatile storage element, such as a magnetic tunnel junction (MTJ), to every volatile flip-flop and SRAM-cell. Most accesses are then served from the energy-efficient volatile part of the cell, and the non-volatile part is only accessed to checkpoint and restore state. Replacing each flip-flop in the processor core with its non-volatile counterpart may be a good solution for a simple 8-bit processor, but could impose substantial design effort and power/performance/area overhead if a larger 32-bit pipelined processor, such as the *Arm Cortex-M0+*, were used. The idea of replacing every volatile storage element with its non-volatile, or NVM-backed, counterpart is also at odds with modern system-on-chip (SoC) practices, whereby the chip is composed of multiple unmodifiable functional units (intellectual property blocks) acquired from different design companies. Additionally, fabricating irregularly distributed single memory elements may not be practical due to manufacturing process control requirements. An approach that uses specialized hardware to accommodate emerging NVMs, and uses hardware/software co-design to minimize complexity has the potential to result in a versatile IC platform that can be used with legacy software across myriad applications.

To research hardware support for IC, however, an appropriate modeling methodology is needed. The complex interactions between energy availability, energy consumption, energy management and intermittent operation complicate IC research. Industry-standard development tools and workflows typically decouple power consumption and execution. For example, register transfer level (RTL) power estimation flows first simulate execution to produce activity traces, then use the activity traces to estimate how much power was consumed. Hence, this flow assumes that execution is unaffected by power consumption. Although this flow, or slight variations of it, is common practice in IC literature, it yields incorrect results because the execution depends on energy, ultimately leading researchers in the wrong direction. In particular, the decoupled method leads towards optimizing for *execution time* without sufficient concern for *energy consumption*. Furthermore, IC is a field of research where obtaining repeatable and reproducible results from physical experiments is particularly challenging because energy harvesters, which have complex dynamics, are used as power sources. A method of modeling IC is thus an integral addition to experimentation, that will greatly improve researchers' ability to share trustworthy results and methods, ultimately accelerating the progress of IC research.

Other areas of research within the field of IC, include such topics as, for example, wireless communication under intermittent operation, cybersecurity within intermittent computing, formal verification. Because this thesis focusses on the computational aspects of IC, these related topics are only reviewed briefly, and are considered to be outside the scope of this thesis.

1.2 Research questions

Following from the above research justification, this thesis addresses three main research questions:

Q1 How can the efficiency of IC running on COTS microcontrollers be improved?

Several methods and programming models have been proposed to support IC on COTS microcontrollers. Among them, methods based on checkpointing are the most promising, due to their good performance and compatibility with existing embedded software. The current state of the art for checkpointing is to save and restore all allocated volatile memory. However, not all allocated memory is modified, or even used, in every on-period, especially not when they are short. Through answering **Q1**, this thesis addresses such inefficiencies to improve checkpointing on COTS hardware.

Q2 How can microcontrollers targeting IC leverage energy-efficient NVM and hardware support to improve efficiency?

Software to support IC on COTS hardware can only go so far. To run IC optimally, hardware support is necessary. Furthermore, new byte-addressable and energy-efficient NVMs are emerging as low-power process nodes scale below 28 nm. Given that NVM is the backbone of IC, these have the potential to greatly improve the performance of IC devices, ultimately enabling IC to be used for a wider range of applications. However, though more energy efficient than existing technologies like flash and FRAM, emerging memories are still much less energy efficient than SRAM, and furthermore have asymmetric read and write energies. **Q2** encompasses the challenge of how best to leverage energy-efficient NVMs, organize the memory system, and provide hardware support for core IC operations in order to accommodate reliable and energy-efficient IC.

Q3 How can new hardware and software for IC systems be developed efficiently, and their performance evaluated with a high degree of repeatability and reproducibility?

Researching both hardware and software for IC, and particularly assessing performance, is complex due to the interactions between energy availability, energy consumption and intermittent operation. Existing modeling tools are inadequate for modeling IC because they lack flexibility (RTL), accuracy (*gem5*[2]), and closed-loop power-performance simulation (both).

[2] A widely used simulation framework in the mobile, desktop, and server computing space (https://www.gem5.org)

Furthermore, the dynamics of EH make it difficult to obtain repeatable and reproducible results. A simulation method that can model the interactions correctly and ensure repeatable and reproducible results is needed.

To address this gap, i.e. **Q3**, *Fused*, a full-system simulator for energy-driven computers was presented in Chapter 4.

1.3 Research contributions

This section outlines the key novel contributions reported in this thesis, addressing the aforementioned research questions. The contributions are listed in the order they are presented in the thesis, and associated publications are listed under each contribution. In addition to publishing datasets that support each article (available through `eprints.soton.ac.uk`), the source code has also been published (where relevant), as mentioned in each contribution.

1.3.1 Taxonomy of IC methods for COTS microcontrollers

To begin addressing **Q1**, an analysis of the current state of the art for IC support on COTS hardware was needed. To this end, Chapter 2 presents a taxonomy that divides existing methods for IC into three classes, and qualitatively compares them against a set of correctness requirements, performance goals, and scalability goals. An early version of this analysis was included in the first publication listed below, and subsequently extended substantially for the second listed publication.

- **S. T. Sliper**, D. Balsamo, A. S. Weddell, and G. V. Merrett. "Enabling Intermittent Computing on High-Performance Out-of-Order Processors". In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ENSsys '18. New York, NY, USA: ACM, 2018, pp. 19–25. ISBN: 978-1-4503-6047-0. DOI: 10.1145/3279755.3279759
- **S. T. Sliper**, O. Cetinkaya, A. S. Weddell, B. Al-Hashimi, and G. V. Merrett. "Energy-Driven Computing". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2164 (Feb. 7, 2020), p. 20190158. DOI: 10.1098/rsta.2019.0158

1.3.2 Paged memory management for efficient state retention on COTS microcontrollers

To complete addressing **Q1**, Chapter 3 proposes a software memory management layer to support IC on COTS microcontrollers. It substantially reduces time and energy spent on saving and restoring state by carefully controlling which data are loaded when restoring from a checkpoint, and written to NVM when saving checkpoints. This memory management method functionally resembles hierarchical memory, and the evaluation results indicate that hierarchical memory is warranted for IC, even though it is rarely seen in today's low-power microcontrollers (**Q2**). This work was published in the publication listed below, and the full source code of the method was published as an open-source project at `https://github.com/UoS-EEC/ICLib`.

- **S. T. Sliper**, D. Balsamo, N. Nikoleris, W. Wang, A. S. Weddell, and G. V. Merrett. "Efficient State Retention Through Paged Memory Management for Reactive Transient Computing". In: *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA). DAC '19. New York, NY, USA: ACM, 2019, 26:1–26:6. ISBN: 978-1-4503-6725-7. DOI: 10.1145/3316781.3317812

1.3.3 Full-system simulator for IC research

Addressing **Q3**, and facilitating research of **Q2**, a new full-system energy-driven computing simulator, named *Fused*, is proposed in Chapter 4. Unlike existing simulators, *Fused* simulates energy and execution in a closed feedback loop, and is therefore able to model the complex interactions between execution and energy (storage, management, consumption, harvesting). *Fused* enables investigation into **Q2** with the flexibility needed to explore emerging NVMs and novel hardware support for IC.

A paper describing and evaluating *Fused* has been published in the first reference listed below. *Fused* has been published to the community as an open-source project at `https://github.com/UoS-EEC/Fused`, and several improvements have been made since the original paper. In the second publication listed below, I assisted the first author by implementing a board-level abstraction in an effort towards simulation of a full wireless sensor node.

- **S. T. Sliper**, W. Wang, N. Nikoleris, A. S. Weddell, and G. V. Merrett. "Fused: Closed-Loop Performance and Energy Simulation of Embedded Systems". In: *Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'20). Boston, MA, USA: IEEE, Apr. 2020

- S. C. Wong, **S. T. Sliper**, W. Wang, A. S. Weddell, S. Gauthier, and G. V. Merrett. "Energy-Aware HW/SW Co-Modeling of Batteryless Wireless Sensor Nodes". In: *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. ENSsys '20. New York, NY, USA: Association for Computing Machinery, Nov. 16, 2020, pp. 57–63. ISBN: 978-1-4503-8129-1. DOI: 10.1145/3417308.3430272

1.3.4 Memory system support for efficient IC

Memory system support for IC, which provides added mechanisms and functionality to simplify or accelerate the core operations of IC, can be used to significantly improve performance. Chapter 5 presents memory system support for IC, called *MEMIC*, to address **Q2**. Motivated by results from Chapter 3, Chapter 5 explores hardware-managed instruction and data caching, to optimize the core operations of IC. Additionally, it proposes a method for limiting the volatile state, which reduces the necessary amount of energy buffering, simplifies application development and increases robustness. To support atomicity, *MEMIC* includes a hardware undo-logger that enables the capability to roll back NVM state when re-executing sections of code. *MEMIC* was modeled and evaluated using hardware-software co-design in *Fused*.

This contribution has been submitted for publication and is currently under review. The reference is listed below. To make *MEMIC* readily available to anyone, all *Fused* models, software support code, and simulation scripts associated with *MEMIC* have been published at https://github.com/UoS-EEC/MEMIC and large parts of the software developed for *MEMIC* have been up-streamed to the aforementioned *ICLib* and *Fused* repositories.

- **S. T. Sliper**, W. Wang, N. Nikoleris, A. S. Weddell, A. Savanth, P. Prabhat, and G. V. Merrett. "Pragmatic Memory-System Support for Intermittent Computing using Emerging Non-Volatile Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022). DOI: 10.1109/TCAD.2022.3168263

1.4 Organization

This thesis begins by providing background, review and a taxonomy of contemporary works in IC in Chapter 2. Then, to address **Q1**, Chapter 3 proposes a novel page-based memory manager that improves the efficiency of checkpointing. Before addressing **Q2**, an appropriate modeling method is needed, so **Q3** must be addressed before **Q2**. Chapter 4 addresses **Q3** by proposing *Fused*, a full-system simulator for IC. Then, in

Chapter 5, memory organization and specialization for IC is presented to address **Q3**. Finally Chapter 6 concludes the thesis and suggests topics for future work.

Chapter 2

Intermittent computing

This chapter provides background and a review of current literature relating to IC. It starts by introducing the main challenges pertaining to IC by categorizing published literature (§ 2.1), then outlines application constraints for intermittent operation (§ 2.2); in essence answering the question *"Which applications are amenable to intermittent operation?"*. Next, challenges that are particularly pertinent to this thesis are reviewed in more detail. First, bugs that may occur when re-executing code are then described in § 2.3, to motivate a set of correctness requirements that define the minimum viable IC method in the following section, § 2.4. Performance and scalability goals are then introduced; these are referenced throughout, in order to systematically compare published works. Evaluating related works against the set of requirements and goals set in § 2.4, a taxonomy of software-based IC methods is given in § 2.5. Next, § 2.6 reviews support for atomic sections in detail. Then, background on prototyping techniques for IC systems is presented in § 2.7. Finally, emerging NVM, and how it can be utilized to provide efficient IC, as well as a review of NVPs are discussed in § 2.8.

The majority of this chapter has been published as part of [130, 131, 132].

2.1 Challenges of IC

IC brings unique challenges that slow down adoption in the real world. This section provides an overview of published literature in field of IC, categorized into the specific challenges each work addresses, as summarized in Table 2.1. Challenges addressed by this thesis are highlighted in bold font in the table. The remainder of this section provides a brief description of each main category of challenges.

TABLE 2.1: Categorized literature on intermittent computing. Challenges addressed in this thesis are highlighted in bold font.

Category	Literature
Software	**Reactive IC** [8, 9, 12, 15, 68, 69, 117, 128, 129, 130, 148] Task-based IC [21, 25, 28, 30, 38, 46, 53]... [62, 92, 96, 122, 161, 164] Static IC [4, 14, 26, 93, 97, 116, 146, 153] OS-support [12, 77] Task scheduling [73, 99]
Processor hardware	NVP [76, 80, 88, 89, 90, 95, 123, 151, 152]... [42, 74, 87, 104, 107, 135, 165, 166] **HW support** [54, 65, 85, 108, 159, 160]
Energy mgmt. & optimization	[30, 40, 52, 53, 57, 60, 70, 75, 93, 110, 136, 154, 161]
Timers	[62, 64]
Peripheral support	Software [12, 19, 98, 119, 137] Hardware [85]
FASE support	[12, 98]
Sisyphean tasks	[29, 98]
Security	[49, 78, 124, 140, 145]
Reliability	[26]
Formal verification	[11, 138, 139]
Approximate computing	[46, 55, 94, 105]
Development tools	**Simulation** [45, 163] Emulation [47, 58, 121] Debugging [27] Static analysis [30, 137] Hardware prototyping [61]
Networking	[1, 3, 22, 31, 34, 37, 48, 79, 101, 141, 144, 162]
Applications	[16, 33, 35, 102, 106, 118, 127]
Machine learning	[51, 67, 81, 82, 106, 158]

Software support. Works relating to software to support IC on COTS microcontrollers are divided into three categories, *reactive IC*, *task-based IC* and *static IC*. These IC methodologies are detailed in § 2.5. Researchers have also worked on the challenges of operating system (OS) support and task scheduling under intermittent operation.

Processor hardware. In the hardware domain, the literature is categorized into NVP and hardware support for IC; where NVP involves replacing volatile logic with non-volatile alternatives, and hardware support for IC involves adding logic that improves checkpointing, peripheral support etc. By adding hardware support to improve IC to an otherwise conventional ultra low power microcontroller, instead of replacing or

augmenting all volatile logic with their non-volatile counterparts, hardware support for IC can remain compatible with conventional system on chip (SoC) design practices like combining black-box functional units ("hard IP") acquired from different IP vendors.

Energy management (mgmt.) & optimization. This category broadly includes works that have focused on techniques for reducing or managing energy consumption through various techniques, for example by proposing PCB-level circuits that manage energy storage in several capacitors of varying capacity.

Timers. Perhaps the most obvious challenge with IC is to maintain a sense of time despite power outages, a topic addressed by the works listed under *Timers*. Without power, clocks cannot run, so an IC device can usually only keep track of time within the current on-period. Recent works have proposed methods that can maintain an approximate sense of time through short off-periods by measuring the decay of a charged capacitor, or the gradual information loss in volatile memory [64], but they are far too inaccurate for e.g. synchronizing wireless communication (3.7–8.7 % error over a period of 35 s). These methods may suffice in specific applications, but an IC device is generally unaware of absolute time.

Peripheral support. Methods for recovering peripheral state after power-loss; this is generally more complex than recovering memory and processor state, as some peripherals require an ordered and timed sequence of commands to enter certain states.

FASE support. A failure-atomic section (FASE) is a section of code that needs to be restarted from the beginning if it is interrupted by a power failure. *FASE support* is needed to express indivisible operations, such as radio transmissions, time-series sampling, and sampling of temporally correlated sensors. Later in this chapter, § 2.3 describes bugs that may occur when re-executing arbitrary sections of code, and § 2.6 describes techniques to avoid them.

Sisyphean tasks. In reference to the Greek mythological king who was condemned "to push a immense boulder up a hill only for it to roll down every time it neared the top, repeating this action for eternity" [wikipedia], a *Sisyphean task* is a FASE that requires more energy than the device can muster, hence it will be restarted indefinitely, rendering the device non-functional [115]. Using a task-based IC method involves a manual and error prone process of dividing computation into appropriately-sized tasks; this process often leads to sisyphean task. Under certain constraints, it is, however, possible to do this process automatically and with protection against sisyphean tasks [29]. To reduce the effort in sizing application-level FASEs without introducing a sisyphean task, Maeng and Lucia [98] proposed a framework to divide certain peripheral operations, such as DMA data transfer or performing computations in an accelerator, into several FASEs.

Security. Literature on security for IC mainly addresses encryption of checkpoints to protect secrets from an attacker that reads the contents of NVM. Reading the contents of locked on-chip NVM, however, requires physical access to the device under attack, a large time-investment, and sophisticated equipment [124]; unless checkpoints are stored off-chip, it is therefore, in the author's opinion, unlikely that encrypted checkpoints are warranted in most realistic scenarios. A more judicious approach would, in most cases, be to address access protection issues with on-chip NVM.

Reliability. To the author's knowledge, there is only a single published work that specifically addresses reliability. Choi et al. [26] consider degradation of ceramic capacitors due to e.g. temperature variation or cracking [142]. The characteristics of the energy buffering capacitor are critical for IC methods that depend on stored energy to save checkpoints (reactive IC), or complete tasks (task-based IC). Adapting to changes in capacitor characteristics during deployment may therefore be necessary to ensure decades of operation.

Formal verification. Researchers have built up a foundation for formally modeling and verifying IC, by relating intermittent and continuous execution models and forming proofs of equivalence between them (at specific points of execution) [11, 137]. These have since been used for automatic instrumentation of checkpoints and FASEs [138].

Approximate computing. The energy-driven nature of IC makes it a good complement for approximate computing, wherein the core idea is to produce produce high-quality results (highly accurate sensing and computation) when energy is plentiful, and lower-quality results when energy is scarce. Thus a device can utilize the energy optimally to produce the most accurate results according to the operating conditions.

Development tools. Because of the tight interaction between energy availability and operation, traditional development tools are insufficient for IC. Prior works have highlighted several unique challenges that emerge when researching and developing energy-driven, and particularly intermittent, computing devices [27, 59, 100]. The challenges mostly stem from the interactions between energy and execution, and from the fact that IC devices frequently reboot, thereby complicating debugging. Another issue is with repeatability of EH sources, which have complex power output dynamics that depend on their environment as well as their load. Development tools for IC are discussed further in § 2.7.1.

Networking. Because IC devices lack an accurate sense of time, reliably synchronizing communication between intermittent nodes is particularly challenging. Proposed solutions include exploiting external sources of synchronization, such as power grid AC signals recovered from artificial light sources [48], or using back-scatter communication [101] combined with radio-frequency energy harvesting (RF-EH).

Applications. Despite a decade of research into IC, only a handful of applications have been demonstrated in the literature. This may be caused by a lack of suitable development tools, and the unsolved challenges related to wireless communication under intermittent operation.

Machine learning. Machine learning, most often deployed in high-power data centers, may seem like an odd companion for IC, but when deployed to reduce the need for communication, it can reduce overall energy consumption. For example, *Camaroptera* [106], uses machine learning to decide which photos should be transmitted from a wildlife tracking IC device.

Among the challenges listed in this section, a few can be seen as fundamental challenges that can be mitigated, but not overcome. To answer the question *"Which applications are amenable to intermittent operation?"*, the next section presents a set of conditions that aim to clarify which types of applications are most suitable for intermittent operation.

2.2 Application constraints for intermittent operation

From the aforementioned research challenges of IC, a set of application constraints for intermittent operation can be derived to indicate which types of applications are amenable to intermittent operation. Because intermittent operation does, by definition, include periods where the device is completely turned off, an application that is to be executed intermittently must:

A1 not depend on absolute time, unless it can be gathered from an external source (e.g. via wireless communication);

A2 be guaranteed sufficient power during events of interest, or allow some events to be missed due to a lack of energy;

Note that all conditions typically can result in unreliable and/or reduced quality of service (QoS) when compared to a battery-powered or energy-neutral system.

A typical application for IC is to sense the environment, detect events, and report them to a central server [29, 98]. Without a consistent source of power, an IC device cannot maintain an absolute sense of time; it can only keep track of time within an on-period. Wireless communication between IC devices is therefore a major challenge, and, despite considerable research effort [1, 20, 31, 34, 37, 101, 141], still an unsolved problem, so today's IC devices generally communicate with an always-on server. Thus, the central server only needs to be notified when an event happens, and can itself annotate absolute timing.

Similarly, under frequent and random power outages, events of interest can be missed as a result of the device being off when the event occurred. This issue can sometimes be alleviated by using an energy harvester that can guarantee power during the event of interest, as exemplified in the following text.

A good example of an IC device is the wireless bicycle trip counter by Bing et al. [16], which counts and reports the revolutions of a bicycle wheel. The device itself is powered by the current induced when a magnet attached to the bicycle wheel passes a coil mounted on the bicycle's fork, hence the event to be detected directly coincides with the power source. By use of intermittent operation, their sensor node is made with a minimum of components, leading to small size and cost without introducing maintenance requirements or sacrificing lifetime. Another potentially suitable application for IC is predictive maintenance on industrial equipment via vibrational or acoustical monitoring [150]; energy for the system could be supplied by a piezoelectric vibration harvester.

2.3 Re-execution bugs

To motivate the set of criteria, performance goals and scalability goals in the following section, § 2.4, this section describes bugs that can occur when re-executing arbitrary sections of code. Re-execution can occur as a consequence of the IC method used, but is also necessary if an operation must restart from the beginning after a power outage (see § 2.6).

Figure 2.1 illustrates three possible faults when re-executing arbitrary sections of code. The function `sampleAndLog` reads a sensor value and appends it to an array of samples, `log`. If a value greater than five is sampled, the `alarm` should be set (assume that alarm is a global flag that gets checked and reset by the caller). A power failure occurs at the end of `sampleAndLog`, as denoted by ↯. The points A, B and C mark potentially problematic checkpoint/task-boundary locations. Consider a naive approach that allocates all variables to NVM, does not checkpoint peripheral state, and allows re-execution (because of statically placed checkpoints, or failed reactive checkpoints). Assume that a checkpoint has been saved immediately before calling `sampleAndLog`. The following three scenarios describe three bugs that can occur when calling `sampleAndLog`, with a subsequent power failure at ↯.

Scenario A illustrates a repeated-IO bug [137]. In the first on-period, a checkpoint was saved at A. If we assume that the sensor value was read to be > 5, `alarm` gets set to `true`. After the power failure, execution resumes from A, but assume that the sensor now reads ≤ 5; the alarm should not be set. The resulting state is corrupt, because the latest logged sensor value is < 5 and the alarm has been set; this is a state that could not have occurred were it not for the power failure.

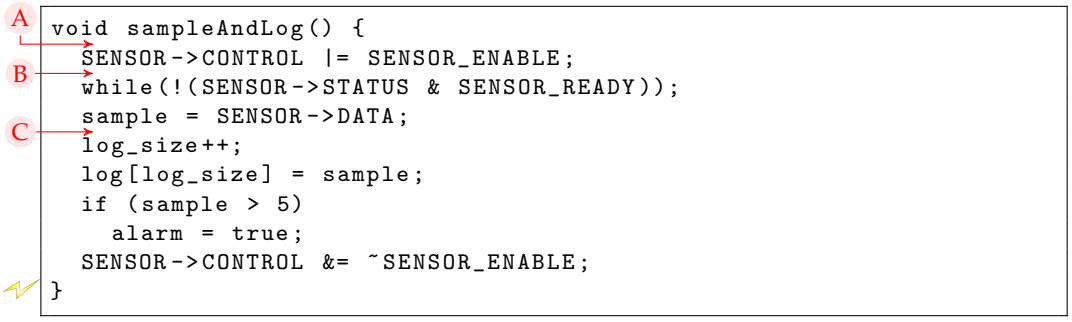


FIGURE 2.1: Pseudo code illustrating three possible faults when re-executing arbitrary sections of code.

In scenario B, a checkpoint was saved at B, and execution continued until the power failure at ⚡. After resuming from B, execution stalls indefinitely, because the sensor was never initialized and so the ready flag never gets set.

Finally, in scenario C, the checkpoint was saved just before incrementing `log_size`. Since this system allocates all variables to NVM, the state of `log_size` and `log` persists through the power outage. When power returns, `log_size` gets incremented yet again, and another log entry is saved. This bug lead to two log entries being saved on one call to `sampleAndLog`; in fact, the log would continue growing if power failed repeatedly at ⚡. Scenario C is an example of a write-after-read idempotency violation[1] [115]. In fact, this same bug also occurs in scenario A.

Scenarios A and C can be protected against by allocating all variables to volatile memory and never re-executing code, by double buffering checkpoints, or by logging and rolling back changed state after a checkpoint/task-boundary [96]. The simplest way to avoid B is to avoid such checkpoint placements entirely, for example by executing (parts of) `sampleAndLog` as a FASE. Other works have focused specifically on the topic of saving and restoring peripheral state [12, 120].

[1] In this context, an *idempotency violation* refers to re-execution of a section of code that is not *idempotent*. This, in turn, means that the same section of code produces different results when it is re-executed. Conversely, an idempotent section of code could theoretically be re-executed infinitely many times, and produce the same effect (i.e. the same memory state, execution state etc.) every time.

2.4 Correctness requirements, performance goals and scalability goals

For systematic evaluation of IC methods, this thesis will use the following correctness requirements(**C1**-**C4**), performance goals (**P1**-**P3**), and scalability goals (**S1**-**S3**). Correctness requirements must be met for IC to function correctly. Performance and scalability goals are ideals that should be sought after, but are not critical — and will inevitably lead to compromises. The requirements and goals are inspired by the ones introduced by Maeng et al. [96], but have been generalized and expanded in order to include recent progress and insights in the field [11, 29, 30, 60, 61, 137, 138, 139, 164].

For correctness, an IC device *must*:

C1 retain computational progress through power outages;

C2 ensure consistency between volatile and non-volatile memory despite power outages;

C3 allow blocks of code to be executed atomically within a single on-period;

C4 avoid sisyphean tasks.

For performance, an IC device *should*:

P1 maximize forward progress through applications;

P2 be reactive to external events;

P3 minimize extra memory footprint imposed by IC framework.

For scalability, and to accommodate widespread adoption, an IC device *should*:

S1 require minimal IC-specific expertise from the programmer;

S2 be compatible with existing software;

S3 be portable across hardware platforms.

C1 is the base premise of intermittent operation: *some* computational progress must be preserved through power cycles.

In certain situations, arbitrary sections of code can be re-executed because the system has to restore to an earlier state after a power failure, for example when re-executing a FASE. This can lead to inconsistency between volatile memory, which loses its state

when power fails, and NVM, which does not, as previously detailed § 2.3. **C3** guards against such inconsistencies.

Support for user (or machine) annotated FASEs is necessary to express application-level atomicity constraints, **C3** encompasses this constraint.

The requirement to avoid sisyphean tasks (**C4**) follows from **C3**: if certain tasks are to be executed atomically, it is imperative that they can be completed within a power cycle.

P1 implies that IC methods should minimize the time and energy spent on state retention, boot, and rollback. This can be measured, for example, as the time and energy spent on the aforementioned overheads compared to the time spent towards completing the workload. To remain reactive to external events (**P2**), IC methods should support asynchronous interrupts, and should generally wake up frequently so as to not miss an event while recharging for the next on-period. Note that **P1** and **P2** often conflict with each other. To optimize for **P1** means that reboots are rare, so that boot and rollback overheads are amortized over a long period of application-execution. On the contrary, optimizing for **P2** means frequent reboots to ensure that no events are missed. In addition to time and energy overheads, IC methods generally impose extra memory footprint to, for example, checkpoint system state, or to instrument application code with checkpoints; this extra footprint should be minimized, because the IC devices typically have strict memory constraints (**P3**). The memory extra footprint an IC system imposes, can be measured as the fraction of memory and storage spent on the IC method as compared to the memory footprint of the application without IC was implemented.

The scalability goals express basic characteristics to enable widespread use of IC, with a large number of devices deployed. As stated in **S1**, the IC method should strive to require little IC-specific expertise from the application programmer, and should therefore aim to encapsulate complexities such as voltage thresholds and FASE support in a lower software layer (or in hardware). This lowers the barrier to entry, and enables more numerous application programmers to employ IC systems. To further lower the barrier to entry and overall development effort, IC systems should strive to be compatible with existing software (**S2**) so that they can leverage prior efforts and mature software libraries. Hardware compatibility (**S3**) is also important – an IC method should strive to be easily portable across different hardware platforms (i.e. printed circuit boards) without large changes.

2.5 Taxonomy

In order to differentiate and evaluate existing works, this section presents a taxonomy based on the following three classes of IC methods.

Static: where checkpoints are inserted at predetermined (design-time or compile-time) points in the program.

Task-based: where the application is divided into small tasks that are executed atomically by a runtime.

Reactive: where checkpoints are triggered as a reaction to the environment.

The fundamental divergence is between static and reactive IC; task-based IC can be regarded as a development of static IC, where checkpoints are replaced by task boundaries. Static and reactive IC typically retain progress through power cycles by periodically (static) or reactively (reactive) checkpointing the volatile state in NVM. Task-based IC needs, in principle, only to save the result from a task's execution and an indicator to the next task to be executed.

State diagrams for the three classes are shown in Figure 2.2. These diagrams show the fundamental states, although most methods diverge slightly —typically by having additional states and edges to improve performance. A key insight is that both static and task-based methods have power-off edges (red) from all states, meaning that the system could potentially instantaneously shut down at any point in the program/runtime. Reactive IC (Figure 2.2c), on the other hand, ensures that the system only powers off after reaching a safe sleep state.

Figure 2.3 illustrates the behavior of each class in relation to the power supply voltage. Static IC restores (R) as soon as the supply voltage, v_{cc}, exceeds the on-threshold of the processor, V_{on}, and starts useful computation (C) while checkpointing (CP) state according to heuristics, for example every few milliseconds. Task-based IC also restores (boot runtime and restart the latest task) as soon as $v_{cc} > V_{on}$. The *restore* operation can be quick because only the runtime and a single task need to be booted, not the entire program. When complete, the results of the current task are saved, and the next task is booted and started; this is termed a task-transition (TT). Reactive IC sleeps until an interrupt triggers *restore*, when v_{cc} exceeds the restore threshold V_R. Reactive IC does not take checkpoints along the way, but rather continues useful computation until v_{cc} drops below V_S, the suspend threshold, which triggers *suspend* (S), checkpointing just before power is lost. In the literature, this is also referred to as Just-In-Time checkpointing [98]. The circles show the latest checkpoint before power is lost; any computation after the latest checkpoint is a waste of resources. Note that some task-based or static approaches may mitigate wasted computation after a checkpoint by measuring stored

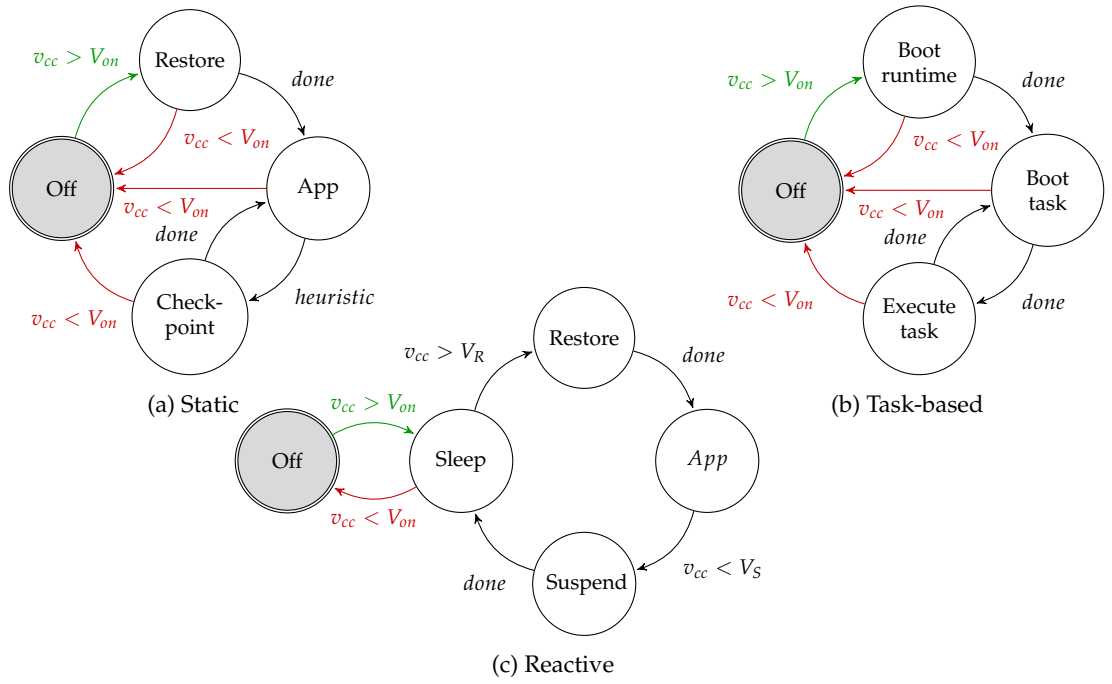


FIGURE 2.2: State machine representation of (a) static, (b) task-based and (c) reactive IC. Reactive IC has simpler state transitions, most importantly avoiding power-off edges (red) from any states where idempotency violations can occur.

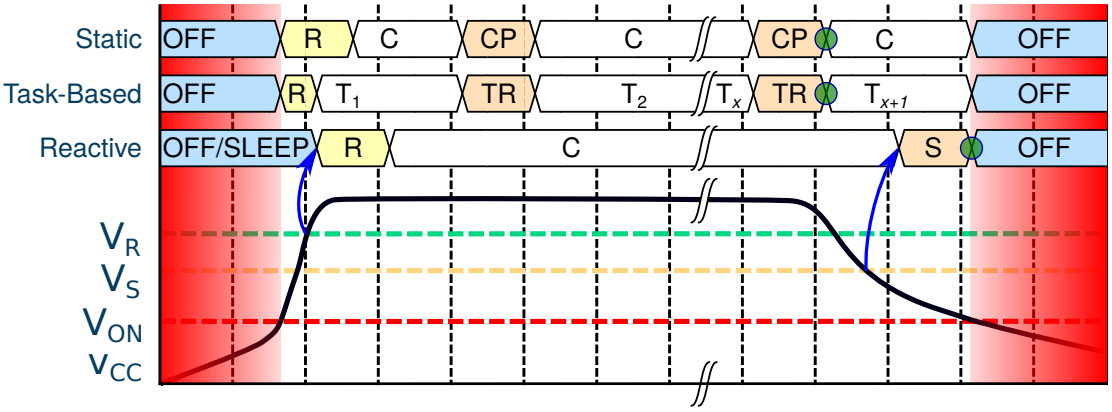


FIGURE 2.3: Behavior of the three classes of IC strategies with regards to v_{cc}.

energy and comparing it to predictions [14] or observations [69] of the energy required to reach the next task-boundary or checkpoint.

The following subsections analyze the three classes of IC in the context of the aforementioned correctness requirements, performance objectives and scalability objectives. Table 2.2 summarizes the findings.

2.5.1 Static IC

Static IC approaches are based on instrumenting application code with checkpoints by the user during application development, or automatically at compile time [14, 97, 116,

TABLE 2.2: Summarized analysis of the three classes of IC.

Objective/Requirement	Static IC	Task-based IC	Reactive IC
C1: Comp. progress	✓	✓	✓
C2: Consistent memory	Challenging	Challenging	✓
C3: FASE support	✓	✓	Challenging
C4: Avoid sisyph. tasks	✗	✗	✓
P1: Forward progress	Slower[1]	Slower[1]	Faster[1]
P2: Reactive to events	Challenging	Challenging	✓
P3: Memory overhead	Instr.[2], checkpoint	Instr.[2], runtime, logs, tiny checkpoint	Checkpoint
S1: Required IC expertise	Moderate	Major	Minor
S2: Software compatibility	✓[3]	✗	✓
S3: Hardware portable	✗	✗	✓

[1]Depends on specific implementation. [2]Code instrumentation. [3]With full recompilation.

146]. Because the checkpoints are inserted a priori, off-line analysis of program execution and control flow graphs can be applied to guarantee memory consistency (**C2**) and improve **P1**-**P3**. The main two advantages of static IC are: (1) information about program execution flow can be exploited to improve restore and suspend performance (**P1**) [14, 83, 97, 165], and (2) checkpoints are triggered regardless of the environment. The latter eliminates the need for energy buffering when powered by a pulsed supply, although some amount of buffering may be necessary for performance reasons.

Key metrics for static IC systems is the accuracy of checkpoints (how many checkpoints were saved compared to how many checkpoints were actually needed), the energy-efficiency of checkpoints (how much energy is spent on saving each checkpoint), the re-execution overheads, and often the overhead of tracking various state for checkpointing and roll-back.

The main challenges to static IC are in the placement of checkpoints: placing them too far apart minimizes superfluous checkpoints, but makes it unlikely or even impossible to reach the next checkpoint; placing them too densely wastes time and energy on superfluous checkpoints. The optimal checkpoint placement depends both on hardware and on harvesting conditions, making solutions unlikely to be portable (**S3**).

Static IC also suffers from frequent code re-execution, because the state rolls back to the latest checkpoint when power returns after a power failure (the computation after the circle in Figure 2.3 is re-executed in the next power cycle). Code re-execution is a waste of energy (**P1**), and can corrupt memory if the re-executed section of code is not idempotent (**C2**) [115].

2.5.2 Task-based IC

Task-based IC is a recent development of static IC where the application is divided into a set of small tasks that are executed atomically by a runtime [28, 38, 69, 96]. This

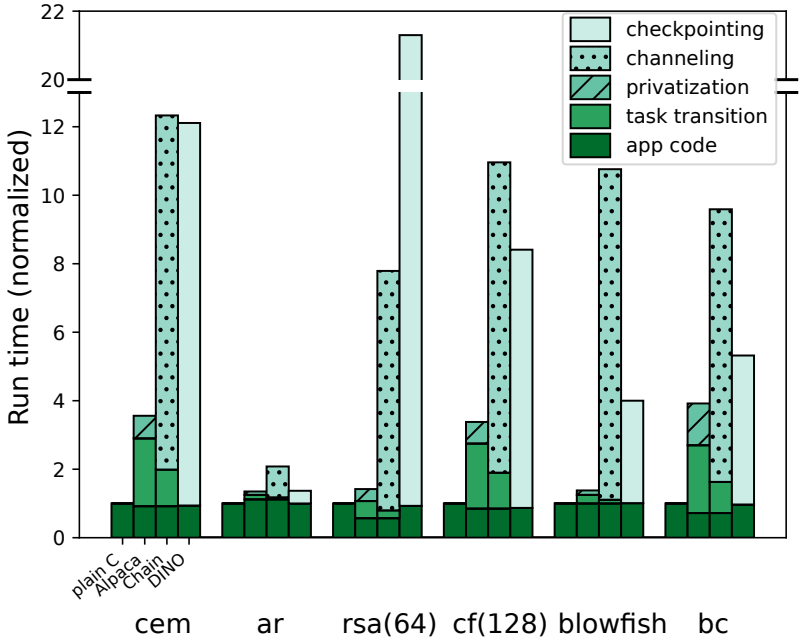


FIGURE 2.4: Run-time overhead of task-based methods while continuously powered (no reboots). Reprinted from [96], 10a.

development is motivated by protecting static IC against idempotency violations, by carefully controlling or recording each task's accesses to NVM, thus making code re-execution safe (**C2**) [28]. Additionally, systems like *Alpaca* [96] offer fast reboot and state-saving because only the runtime and the current task is booted, and only persistent data from the task needs to be saved (**P1**). However, in addition to re-execution overheads, handling idempotency is expensive. Figure 2.4, shows the run-time overhead of contemporary task-based methods when continuously powered (no reboots).

Key metrics for a task-based IC system include the overhead of the task-based runtime (and sometimes scheduler), and the overhead spent on various memory access tracking. Figure 2.4 shows these static overheads (no reboot) clearly, as compared to the plain C implementation where IC functionality was not implemented. Additionally, the time and energy spent on rolling back and re-executing code after power failures plays a significant role in the overheads of task-based IC.

In addition to technical challenges, such as

- expensive and complicated roll-back schemes necessary for protection against idempotency violations (**C2**),
- large memory footprint (**P3**),
- expensive task-transitions (**P1**),
- and handling interrupts to react to asynchronous events (**P2**),

task-based systems struggle to meet any of the three scalability objectives. The imposed programming model is not compatible with existing software (**S2**), as it requires redesigning applications into tasks. Finding optimal task boundaries (analogous to checkpoint placement in static IC) is difficult, and depends on both the specific underlying hardware and the energy harvesting conditions; great care must be taken to avoid sisyphean tasks (**C4**). Thus designing a task-based application generally requires a high degree of IC-specific expertise (**S1**) [30]. Furthermore, since optimal task-boundaries are hardware-specific, applications are not portable across hardware platforms (**S3**).

CleanCut [30] is a recent method that checks for sisyphean tasks (termed "non-terminating path bugs" in the paper) and performs automatic task decomposition. The method is a significant step towards making task-based methods scalable (**S1**-**S3**). However, limitations of the method, such as the requirement that the programmer must specify an iteration bound for each unbounded loop in the program, restrict its utility for applications that depend on existing libraries (**S2**).

Classical task-based approaches such as *Alpaca* [96] are incompatible with asynchronous interrupts, because they would break the premise of an application divided into atomically executed tasks. *Ink* [164] aims to resolve this by use of task-threads and scheduling. However, tasks in *Ink* cannot be pre-empted, making reaction time slow and unpredictable. A more recent alternative is *Coati*, which "employs a split-phase model that handles time-critical I/O immediately in a brief interrupt handler, but defers processing the interrupt's results until after the interrupted task or transaction completes" [122] thus accomplishes real-time response to interrupts.

2.5.3 Reactive IC

Contrary to static and task-based approaches, reactive IC is generally implemented in an application agnostic manner [8, 9, 68], greatly improving **S1** and **S2** when compared to typical task-based or static methods. Reactive IC consists of the following two operations:

> **Suspend:** Checkpoint volatile state to NVM, and enter low power mode or shut down.
>
> **Restore:** Restore volatile state from a checkpoint.

The volatile state is any state that is lost in the event of a power outage, and that is needed for computation to proceed correctly when power returns. The set of volatile state is architecture and platform specific. However, reactive IC remains portable across hardware (**S3**), because only the *suspend* and *restore* functions need be redesigned for each platform, not the entire application.

Power supply monitoring is set up to generate interrupts that trigger *suspend* and *restore* operations when the supply voltage, v_{cc}, crosses a threshold. When the restore threshold is exceeded, an interrupt triggers *restore*, which restores state from a previous checkpoint. Similarly, when v_{cc} drops below the suspend threshold, V_S, a checkpoint is triggered and the system enters low-power mode (where volatile state is retained) or shuts down. If power returns while the system is still in low-power mode, *restore* is unnecessary because the volatile state was retained, so the application can continue execution directly, thus improving **P1** [8]. When powered by a low-current source, *suspend* can often be avoided altogether by pre-emptively entering sleep mode before v_{cc} drops below V_S [32, 93]; this can, in some cases, improve **P1**.

The suspend threshold ensures that checkpoints are only triggered when power failure is imminent, thus nearly eliminating superfluous save operations (**P1**) [9, 98]. The restore threshold is used to guarantee the success of the next checkpoint; it eliminates code re-execution, and thus also idempotency violations (**C2**). Determining such a V_R is intractable for static and task-based IC because all possible execution paths must be exhaustively analyzed. In reactive IC, V_R can be determined by online measurement, or calculated based on the amount of state to *suspend* and *restore*[8]. Note that to guarantee the success of the next checkpoint, a minimum amount of energy buffering is required. The energy buffer needs only suffice to power *suspend*, but additional capacity may improve performance.

With proper, pre-emptive, interrupt prioritization, reactive IC is natively reactive to asynchronous external events (**P2**), and furthermore can guarantee forward progress in application code (assuming that any FASEs used are sufficiently small).

The main drawback of reactive IC is that saving and restoring the entire state is expensive compared to task-based systems where only one task is saved/restored at a time. This is especially wasteful if the power cycle is short, because only a small part of the program is expected to execute. This problem can be partially alleviated by techniques such as comparing the current state to the latest checkpoint, and only saving the difference [15, 43, 148]. Another drawback is that reactive IC does not generally handle FASEs (**C3**) as gracefully as task-based methods. A solution to this is proposed in [12], where a formal division between application code (which accesses only application memory) and driver code (which accesses peripherals and NVM) assists a kernel to ensure atomic execution of driver-functions without introducing idempotency violations (**C2**).

The key metrics for a reactive IC system is the time and energy spent on restoring state after a power failure and on suspending state when a power failure is detected. This is often referred to as the restore time/energy and the suspend time/energy.

2.6 Supporting failure-atomic section (FASE) execution

Due to the potential for re-execution bugs, as outlined in § 2.3, FASE support demands special attention. As previously mentioned, FASEs are used to express indivisible operations, such as radio transmissions, time-series sampling, and sampling of temporally correlated sensors. For example, if the transmission of a radio packet gets interrupted partway by a power outage, it needs to be restarted from the beginning. Another example is recording a contiguous window of time-series data from a sensor such as a microphone or accelerometer: the resulting data is only sensible if it was recorded without power interruptions.

For such application-specific atomicity constraints, the FASE is annotated by the programmer, for example by using a wrapper function [12]. The programmer is responsible for ensuring that FASEs are small enough to be executed in a single power cycle; specifying a FASE that requires more energy than the device can muster leads to (a sisyphean task, as explained shortly), because the FASE, by definition, cannot be subdivided. In particular cases, FASEs can be annotated automatically to prevent re-execution bugs (see § 2.3) [137, 138, 139].

Reactive and static IC methods have often omitted FASE support [8, 9, 116]. Task-based IC implements FASE support by default, since task-based IC is, in fact, based on dividing programs into a set of FASEs. These task-based methods use various techniques to ensure protection against re-execution bugs:

- data versioning, where relevant variables residing in NVM are loaded to the stack during a task execution, then saved back to NVM during the next task-transition [92];
- accessing NVM through abstract channels in a fashion similar to transactional memory [28, 56];
- buffering (certain) writes to NVM, and only committing the changes during checkpointing [96].

These techniques could also be used to provide FASE support for static and reactive IC. However, they all impose substantial runtime overhead, increase memory footprint because of privatization or double-buffering, and require programmers to declare "protected" variables and use explicit application programmer's interfaces (APIs) to access protected variables, and often rely on custom static analysis tooling.

A simpler method is possible for static and reactive IC, under the reasonable constraint that all variables be allocated to volatile memory [12]. Static and reactive IC methods can then support FASEs by:

1. saving a checkpoint immediately before starting the FASE,
2. disabling checkpointing during the FASE,
3. and finally enabling checkpointing again once the FASE has completed.

This method may be preferable because it reuses the existing checkpointing mechanisms, and maintains the software-compatibility of reactive IC, i.e. does not require static analysis tools or manual annotation of special variables.

Using FASEs can, however, introduce sisyphean tasks (**C4**). *Samoyed* [98] handles sisyphean tasks by profiling all FASEs in hardware, and, for suitable peripherals/operations, dynamically breaks down the FASE into parts that are small enough to execute in a single on-period. Not all peripheral operations can be divided into smaller parts, but as shown in their paper, this method is very effective for certain memory mapped accelerators. In the general case, though, FASEs cannot be divided, and so it is up to the programmer to avoid sisyphean tasks, possibly with assistance from static analysis tools [29].

2.7 Prototyping and evaluating IC devices

This section describes different approaches to evaluating new methods and ideas within IC. First, hardware prototyping is discussed, where COTS hardware components, usually combined with software support is used to implement and evaluate a new method. Second, high-level analytical models are discussed. These have the advantage of enabling rapid evaluation across a wide range of parameters, but could suffer from inaccuracies. Last, simulation, a middle-of-the-road option between analytical modelling and hardware prototyping, is discussed. All three approaches are useful and necessary tools within their applicable domain, but none can be used for all experiments.

2.7.1 Hardware prototyping

Hardware prototyping, in the form of making a functional IC device out of COTS components mounted on a circuit board, is commonly used, and is a good evaluation strategy for software-based IC methods and new peripheral hardware such as energy management circuits. For many experiments and IC methods, hardware prototyping can be the most straightforward approach, as COTS hardware platforms that can support IC are readily available. For instance, the *MSP430FR*-series of microcontrollers from *Texas Instruments* is a popular choice in IC literature [4, 8, 9, 10, 12, 13, 14, 16, 27, 28, 29, 30, 32, 77, 122, 129, 164] due to its energy-efficient and byte-addressable FRAM NVM.

Furthermore, researchers have designed printed circuit board (PCB) level prototyping platforms to make IC more accessible [61].

Given appropriate measurement equipment and design of experiment, hardware prototyping yields accurate results. However, in addition to common complications in the field of computing, such as selection of appropriate benchmarks etc., evaluation of IC systems is greatly complicated by its inherent tight interactions between energy availability and operation [27, 58, 59, 100]. A related issue is with the repeatability of EH sources, which have complex power output dynamics that depend on their environment as well as their load [126]. The combination of a system with complex interactions between energy and operation, and an energy source that is difficult to control in terms of both repeatability and reproducibility, mean that realistic experiments with IC systems can become unreliable.

To help overcome challenges related to reproducibility and repeatability, as well as simplifying hardware and software development for IC devices built with COTS components, the following works have proposed new development tools. *Ekho* [58, 59] addresses the repeatability challenge by recording IV-surfaces (current-voltage curves over time) of energy harvesters, so that they can be replayed in the lab for realistic and repeatable evaluation of energy-driven systems. This approach was later refined and extended to support a experiments on networks of several intermittent nodes, all powered by replayed IV-surfaces [47]. To simplify development of IC systems, Colin and Lucia [29] proposed *EDB*, the energy-interference-free debugger, which is a debugger that aims to enable debugging of energy-driven systems without interfering with their energy state. *EDB* proposes new debugging primitives, such as energy-breakpoints that halt execution when the system's energy storage depletes or charges to a specific level, to facilitate efficient debugging of intermittently-powered devices.

However, although researchers have made significant improvements in hardware prototyping of IC systems, hardware prototyping will always be limited by available hardware. To research new hardware approaches at the level of integrated circuits, modeling is the only feasible option, due to the prohibitive cost of designing, verifying, and manufacturing integrated circuits. The next two subsections discuss two different approaches to modeling IC systems.

2.7.2 Analytical modelling

When studying a system through a large parameter space, analytical modelling can be a powerful tool. It is very well suited to answer questions like "What's the trend of application completion time as the size of the energy buffer increases", or "How much relative performance would be gained by doubling the size of the energy harvester?".

As it is based on abstracting mechanics from a real system into solvable equations, it does not model every detail, but can provide fast answers and show trends. Thus it can be used as a first step before hardware prototyping or simulation, to determine where in the design space time-consuming experiments and simulations should focus. It can also be used during development to reveal areas of improvement that are likely to yield a high return.

Several analytical models have been proposed to evaluate IC methods [9, 119, 125] and energy-neutral systems [72]. The most comprehensive in the space of IC, *EH-model* [125], can provide early estimations of completion time for several IC methods using different state retention mechanisms, but does, however, depend on detailed input data obtained through offline profiling.

2.7.3 Simulation of IC hardware and software

In a sense, simulation, or more specifically virtual prototyping, can be a middle way between hardware prototyping and analytical modelling, offering more flexible design space exploration than the first, and higher accuracy than the latter.

The typical workflow for early performance and power modeling of digital circuits consists of RTL design, synthesis and simulation to obtain accurate power and timing information. This workflow works well for continuously-powered circuits, and can yield very accurate results for a specific process node. It is, however, not suitable for IC, because *it does not take into account the effect of energy availability and consumption on execution and vice versa*. For example, under intermittent operation, a device commonly reboots as a consequence of severely constrained power supply, thus its execution flow cannot be determined without modeling energy availability. RTL design, synthesis and simulation is is also inflexible, very time consuming, and computationally expensive, hindering efficient design space exploration.

A popular option in high-performance computing (mobile, desktop and server) is to use *gem5* [17], where processing systems are modeled at a much higher level of abstraction. Here, software is written to model hardware behavior, mainly by abstracting signals and operations to function calls. It is not synthesizable, hence not a replacement for RTL, but it provides for fast simulation and prototyping, allowing reasonably accurate power and performance modeling of complex systems running complex software. Power consumption can be estimated in *gem5*, e.g. by recording high-level architectural events and feeding them to a power modeling tool [149]. However, being optimized for simulation speed and for modeling high-performance systems, *gem5* lacks cycle-accurate capability, and is far too complex to efficiently and flexibly model low-power microcontrollers.

A key factor for IC is that their behavior is governed by the availability of energy. Their power consumption can even affect the amount of power being harvested, because of the non-linear IV-curve of many energy harvesters [63]. This is not the case for traditional "always on" computers, hence *gem5* and RTL workflows can decouple power and performance modeling.

I postulate that, for energy-driven computers, power and performance must be modeled simultaneously, in a closed feedback loop. Closed-loop modelling of power and performance means that the power model feeds its results to the performance/functional model, which in turn feeds back its results to the power model. In simulation, this feedback might occur once per simulation time-step, such that the power model makes its estimations based on the power results from previous time step and immediately feed them to the performance model; thus the otherwise infinite loop is broken. Furthermore, timing accuracy requirements may be stricter for energy-driven IC devices, because they might only be active for a few thousand clock cycles at a time (as is shown later, in Chapter 3).

NVPSim, a *gem5*-based model of the *THU1010N* [151] NVP was proposed in Yizi Gu et al. [163], and extended in Wu et al. [157] to include limited support for modeling peripherals. To model intermittent execution, *gem5* was modified to include a simple capacitor model, and the replaying of a power trace. *NVPSim* supports only a fully custom in-house NVP based on an architecture rarely used in modern embedded systems. Its extension for modeling peripherals also requires specialized driver code. The simulator therefore does not run the exact same binary as the real device, and may operate differently, potentially hiding hardware and/or software bugs.

Ma et al. [95] explored the microarchitecture of NVPs with RTL simulation, using *NVSim* [36] as the NVM model. The paper focused on comparing execution pipelines (non-pipelined, in-order, out-of-order), as well as exploring which parts of the microarchitectural state to save on power failure. Simulating the *THU1010N*, their RTL based simulations achieved intermittent execution of several benchmarks with reported performance errors of less than 5 %; however the simulation method is only briefly described. Notably, with a fully non-volatile processor, power and performance can more easily be calculated from an execution trace after simulation (decoupled power and performance), because the executed program can be agnostic to reboots.

Ruffini et al. [121] recently proposed *NORM*, an field-programmable gate-array (FPGA)-based framework for prototyping of IC systems. It includes a framework for emulating non-volatile logic and memories (and their longer access times), a method for replaying voltage traces, and a simple counter-based "energy approximator". This framework emulates prospective IC designs at the RTL abstraction level, meaning that it is mostly suitable for validation/verification, rather than design space exploration. *NORM* uses static voltage traces, so the effects of power consumption on harvested energy are not

modelled. In a subsequent paper, Philip et al. [112] used *NORM* to simulate a RISC-V core running intermittently, and to evaluate different software-implemented checkpointing strategies (different hardware architectures were not explored).

Siren [45] extended the *MSP430* simulator *MSPSim* [41] to include NVM, and basic energy simulation capabilities. *Siren*'s analog/energy domain consists of a capacitor model that also controls whether or not execution is active, and an *Ekho* emulator, that replays *Ekho* IV surfaces in simulation. *Siren* does not, however, realistically simulate energy consumption, because it assumes a single static energy consumption per instruction.

Power estimation is an integral part of any simulator that targets IC. A common approach to model the power consumption of a central processing unit (CPU) is to use instruction level power modelling (ILPM) [23], where the aim is to first ascertain how much energy each instruction (or instruction type) consumes, and then estimate total energy consumption based on the tally of how many times each instruction was executed. This methods is very practical, as it offers a simple way to add power estimation to existing CPU emulators. However, it can only accurately model the power consumption of the CPU, as it does not account for other parts of the SoC. For example, microcontrollers comprise several peripheral units and heterogeneous memories in addition to the CPU; all of which can contribute significantly to total power consumption. Full-system simulators, like *gem5*, therefore use a much wider set of events (e.g. reads and writes to each specific memory and so on), and also the state of individual components to model the full-system power consumption [149]. The events are used to model dynamic power consumption, i.e. the component of power consumption that varies with activity, and the states are used to model static power consumption (chiefly leakage power). All the aforementioned simulation methods used in IC either rely on RTL power estimation (which is decoupled from execution), assume constant power consumption, or assume constant energy per instruction.

Although there have been several prior works that target simulation of energy-driven/intermittent systems, they lack support for modeling external circuitry beyond a simple storage capacitor, and none include a methodology for capturing power modeling parameters. Most also lack the modularity and flexibility necessary to allow efficient hardware-software co-design.

2.8 Leveraging emerging NVM for IC

Nearly all COTS microcontrollers today use flash as their embedded NVM. Flash memories can be made to have high density and low read access energy. However, writing to flash is expensive in terms of energy and time. A page of memory (hundreds to thousands of bytes) must first be erased in a process which uses a high-voltage pulse to

set all the bits in the page to a common value (typically 1). This erase process also limits the write-endurance of flash memories to be on the order of 10^5 cycles. Therefore flash memory cannot be used as regular memory, but is instead used as a programmable read-only memory for storing instructions, constants and initial values, but not variables.

The other type of embedded NVM that is commercially available for ultra low power microcontrollers is FRAM, which has orders of magnitude lower write energy than flash, and virtually unlimited write-endurance [143]. FRAM can thus be used as a regular memory (like SRAM), albeit with higher access energy and latency than SRAM. A decade after the first COTS microcontroller using FRAM was demonstrated [167], however, FRAM is still only available from a single manufacturer in niche product line.

Recent developments in NVM technologies, such as various forms of MRAM (MRAM, STT-RAM, SOT-RAM, and more), PCM, and resistive RAM (RRAM/ReRAM/memristor) are opening new opportunities for IC. Emerging NVMs have order-of-magnitude improvements in write-energy, write-time and endurance when compared to the traditional embedded flash memory in today's microcontrollers [24]. Additionally, new NVMs such as spin transfer torque MRAM (STT-MRAM) and PCM are compatible with existing complementary metal oxide semiconductor (CMOS) processes and scales to smaller process nodes than what is economically feasible with embedded flash or FRAM ($\lesssim 28\,\text{nm}$) [6, 18]. However, whereas embedded PCM is available only from a single manufacturer (*ST Microelectronics*) targeting a high-performance microcontroller, MRAM is available from two foundries (*GlobalFoundries* and *Samsung*) [18, 50] and microcontrollers optimized for low power consumption, featuring embedded MRAM are also beginning to emerge [5]. MRAM is therefore the primary candidate for ultra low power microcontrollers, with PCM being a likely contender.

Because IC requires extensive use of NVM to retain computational progress through power cycles, more energy-efficient NVM technologies are likely to enable substantial improvements if employed appropriately. Reductions in NVM access energy will substantially decrease the active power consumption of IC devices, as well as the more critical *suspend* and *restore* energies, leading to relaxed requirements for both energy supply and storage. Similarly, small access granularity allows for further energy-optimization by selectively saving specific data instead of large blocks.

These emerging NVM technologies have been used at varying levels of integration. The following subsections first describe the related field of NVP, where non-volatility is tightly integrated at the level of bit cells and flip-flops, then reviews hardware-supported IC, which instead use functional units that help saving and restoring state in an otherwise conventional SoC. The latter approach is enticing because it can leverage efficient memory compilers and enables SoCs composed of functional units from different vendors without the need to re-design them for IC.

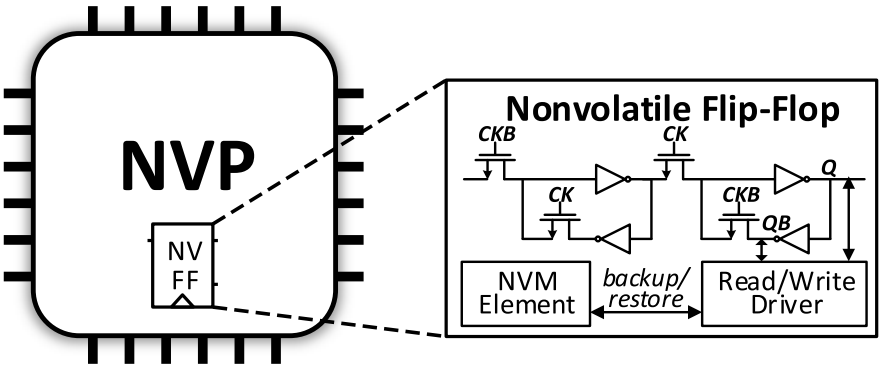


FIGURE 2.5: Example NVM-backed flipflop used as registers in [87]. The figure is rearranged and reprinted from [87].

2.8.1 Non-volatile processors

Instead of backing up and restoring volatile state in NVM, NVPs[42] eliminate the volatile state by using only NVM as memory, and integrating a non-volatile register file in the processor core [76, 80, 83, 86, 87, 88, 89, 123, 135, 151, 152, 160]. Because emerging NVMs still suffer from low endurance and relatively high write energy compared to SRAM, most published NVM designs opt for NVM-backed SRAM memory cells and registers in place of using non-volatile registers directly. They then operate as reactive IC, backing up register contents in parallel when a power failure is detected [80, 95]. Figure 2.5, reprinted from [87] shows the principle, and a circuit diagram of their non-volatile flip flop (NVFF) used in their recent NVP. Using their previously developed 8051 instruction set architecture (ISA) NVP [151], together with a novel integrated supply voltage monitoring circuit, 32 kB FRAM instruction memory, 32 kB FRAM data memory and five voltage domains, their non-volatile SoC achieved 9 µs *suspend* time and 3 µs *restore* time with an active power of 371 µW/MHz at an operating frequency of up to 16 MHz. Their non-volatile SoC was fabricated using a 130 nm CMOS process, amounting to a 22.09 mm^2 die area. To contextualize this, a commercially available *MSP430FR-series* microcontroller with integrated FRAM spends a minimum of 12.3 µs on suspend and restore when using FRAM as unified memory, running at it's maximum clock frequency of 8 MHz at an active power of ≈375 µW/MHz ($V_{CC} = 3.0$ V)[2] [68].

Although NVP can retain state very efficiently (**P1**), FASE support becomes complicated (**C3**). To handle FASE execution, NVPs must either incorporate energy management circuits that halt execution until sufficient energy to complete the next FASE is guaranteed [71], or carefully roll back state after a failed attempt. The former necessitates precise knowledge about how much time and energy the next FASErequires. The latter involves carefully controlling NVM writes and applying state logging and

[2]MSP430FR5739 datasheet from Texas Instruments. Power consumption read from $I_{AM,FRAM\_UNI}$ @8 MHz and multiplied by a supply voltage of 3.0 V.

roll-back in order to return to a previous execution state without corrupting memory (**C2**).

The tight integration of non-volatile elements in NVP also poses fabrication challenges. Replacing each flip-flop in the processor core with its non-volatile counterpart may be a good solution for a simple 8-bit processor, but may impose substantial design effort and power/performance/area overhead if a larger 32-bit pipelined processor, such as the *Arm Cortex-M0+*, were used. The idea of replacing every volatile storage element with its non-volatile (-backed) counterpart is also at odds with modern SoC practices, where the chip consists of multiple unmodifiable functional units (IP blocks) acquired from different companies. Additionally, fabricating irregularly distributed single memory elements may not be practical due to manufacturing process control requirements.

2.8.2 Hardware support

As an alternative to making a processor fully non-volatile, one can instead create hardware support units that provide fast and efficient state retention through efficiently moving data between volatile memory and NVM.

Clank proposes additional circuits in the memory subsystem that track write-after-read dependencies (which can cause idempotency violations), and buffers them in a volatile buffer. When the buffer is full, a checkpoint is triggered, saving state to avoid re-execution. In the event of a power failure, the offending memory accesses are implicitly discarded because the buffers are volatile; execution can easily and safely resume from a previous checkpoint. *Clank* satisfies **C1** and **C2** without software modifications, but does not have explicit support for FASEs; one potential problem occurs if the buffer consistently overflows during an FASE, leading to the equivalent of a sisyphean task. Another important aspect that *Clank* does not explore, is the difference in energy-consumption between volatile memory and NVM. *Clank* also does not protect against repeated IO bugs, which were first described after *Clank* was published.

Freezer proposes a simple hardware peripheral that controls accesses to SRAM and NVM, and tracks which blocks of SRAM have been written to in the current on-period, so as to avoid copying unmodified blocks [108]. As shown in our evaluation (§ 5.3.6), *Freezer* can be an effective way to reduce checkpointing energy. However, the worst-case checkpointing energy remains unchanged, so *Freezer* does not reduce the necessary amount of energy buffering. Additionally, *Freezer* considers data memory, but has no mention of instruction memory, which often is a larger factor for total energy consumption.

A hybrid MRAM-SRAM cache was proposed by Xie et al. [159] on a 480 MHz NVP simulated in *gem5*. It comprises a mix of SRAM blocks, which offer fast and energy-efficient access, and MRAM blocks, which offer non-volatility and higher density, and

an access pattern predictor that intelligently allocates cache lines to either block type. This architecture may be efficient for a high-speed device with non-uniform memory access latency and energy (i.e. expensive access to main NVM). However, it is likely not applicable for ultra-low power IC devices that run at a few megahertz and thus would have similarly fast and energy efficient access to non-volatile cache blocks as to NVM. The proposed hybrid cache also does not include FASE support.

2.9 Summary and discussion

This chapter established correctness criteria, performance goals, and scalability goals for IC, and used them to provide a systematic review and taxonomy of contemporary methods for IC on COTS hardware. Among the three classes of IC, reactive IC is the most promising because of its compatibility with existing software, as well as its performance. However, contemporary reactive IC methods suffer from inefficient state retention, as they checkpoint and restore all data in every power cycle, regardless of whether the data are modified, or even referenced during each on-period. This motivates **Q1**, which this thesis addresses in Chapter 3.

While support for IC on COTS hardware is important to lower the barrier to entry, especially while IC is an emerging technology, having *some* hardware support can potentially provide much improved performance. Furthermore, recent years have seen large developments in byte-addressable and more energy-efficient NVM technologies. It is clear that leveraging these can improve the performance of IC substantially. However, it is not clear how best to take advantage of these NVMs to optimize the core operations of IC. While being far more energy efficient than flash, they are still not as efficient as volatile SRAM. Furthermore, they exhibit asymmetric read and write energies. These factors may warrant techniques like instruction and data caching; techniques which are common in performance-optimized computing devices, but are not commonly used in energy-optimized ones. These issues are encompassed by **Q2**, but in order to perform such research, an appropriate modeling method is required (**Q3**). This thesis therefore addresses **Q3** before **Q2**.

To research hardware support for IC, an appropriate modeling tool is required. Whereas energy and execution can be modeled separately for most embedded systems, IC relies on real-time interactions between energy-availability, energy-consumption, and execution; this thesis therefore postulates that their energy-environment and execution must be modeled in a closed feedback loop. *Fused*, presented in Chapter 4 addresses this challenge, i.e. **Q3**. In the subsequent chapter, Chapter 5, *Fused* is then employed to address **Q2**.

Chapter 3

Efficient intermittent computing on commercial off-the-shelf (COTS) microcontrollers

3.1 Introduction

Support for IC on COTS microcontrollers can be important for adoption, because it allows early testing, prototyping, and possibly products to start using IC with existing hardware. This chapter improves on the state-of-the art of checkpointing, to address **Q1**.

The previous chapter, Chapter 2, found that, among the three classes of IC, reactive IC is the most promising, due to its software compatibility, low execution overheads, and more (see § 2.5). However, existing reactive IC methods [8, 9] suffer from inefficient state retention because the entirety of volatile memory (VM) is saved and restored in every power cycle. Recent works proposed tracking dynamic memory to avoid backups of unallocated state [15, 148], henceforth termed *AllocatedState*. However, not all allocated memory is modified or even used during power cycles with short on-periods. Figure 3.1 shows the result of an experiment measuring the percentage of modified

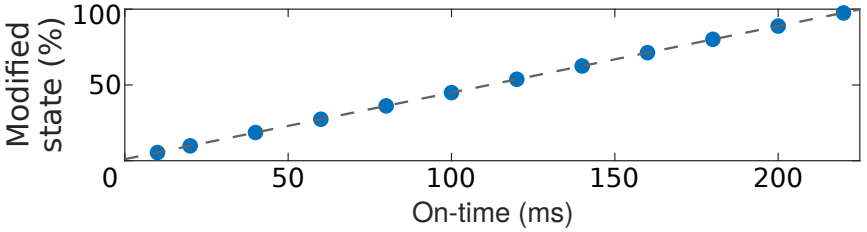


FIGURE 3.1: Percentage of allocated memory that is modified at the end of an on-period in relation to its duration (on-time).

allocated memory in relation to the on-time of a power cycle. The experiment was performed by executing 128 b advanced encryption standard (AES) encryption on a 2 kB string on an *MSP430FR5994* microcontroller running at 8 MHz. Suspend and restore operations were triggered by a function generator connected to general purpose input/output (GPIO) pins. As Figure 3.1 shows, the percentage of state that is modified at the end of an on-period can be very low, especially if the on-period is short. This is true for all applications, although the slope and shape of the trend would differ; some applications may modify a large amount of data during specific phases, whereas others, like the demonstrated, have a more linear pattern.

Bhatti and Mottola [15] proposed comparing the current state with a previous checkpoint to identify unmodified data, but this method is only applicable for asymmetric memories such as flash, where reads are much cheaper than writes [148]. For FRAM, where reads and writes have the same energy cost, the method is counterproductive. An alternative to explicit state retention is to use NVM as main memory [68], but this degrades overall performance due to increased access energy [8, 66].

In this chapter, *ManagedState*, a lightweight page[1]-based memory manager that tracks active and modified regions of memory is proposed. A mathematical model that leverages the memory manager to increase available execution time by calculating suspend and restore voltage thresholds at runtime is then developed. Based on energy consumption, the model also yields a constraint on the number of active and modified pages. Because violating this constraint may lead to a corrupt checkpoint, the memory manager maintains a limit on the number of modified pages by writing back inactive modified pages when necessary.

A conceptual comparison between *ManagedState* and *AllocatedState* is shown in Figure 3.2. Knowing precisely the active regions of memory speeds up *restore* (R), while knowing which regions are modified speeds up *suspend* (S). Improved *suspend* and *restore* performance also enables runtime threshold adjustments to defer *suspend* to the last possible moment and to wake up from sleep (Z) and *restore* at the first possible opportunity; thus maximizing the time spent on useful computation (C). To enable this, *ManagedState* loads (LD) pages as needed, and saves (SV) pages to NVM at runtime when necessary.

The key contributions of this chapter are:

- *ManagedState*, a page-based memory manager for tracking active and modified regions of memory, resulting in 26.8–86.9 % reduction in accumulated *suspend* and *restore* time, with minimal memory footprint (84–1102 B) and minimal to moderate runtime overhead (1.05–1.48 ×) (§ 3.2).

[1] Herein, the term "page" refers to a fixed-size contiguous region of any memory. The size is determined by the user at compile time, and does not change during runtime.

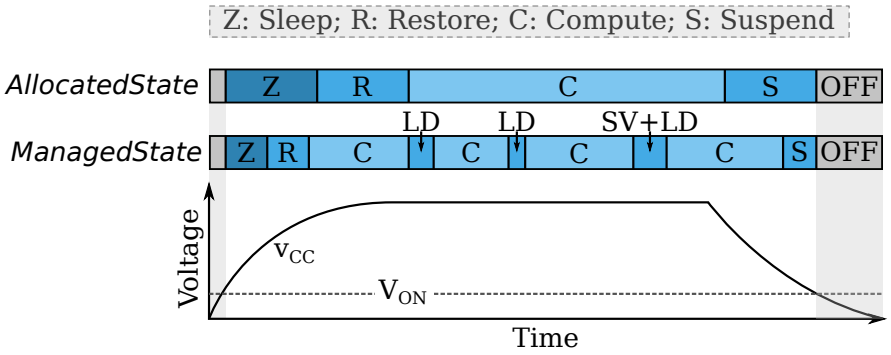


FIGURE 3.2: Conceptual comparison between *AllocatedState*, and *ManagedState*, in response to a power supply trace.

- Runtime calculation of safe suspend/restore thresholds, based on the amount of active and modified memory, that eliminate code re-execution and corrupt checkpoints, as well as increase available execution time. The combination of both contributions result in up to 5.3 × faster application execution (§ 3.4).

This chapter begins by presenting *ManagedState*, then presents a method for determining safe and efficient suspend and restore thresholds, followed by experimental validation. It concludes with a discussion on key findings and how they relate to the research questions of this thesis.

The majority of this chapter has been published in Sliper et al. [129].

3.2 *ManagedState*: tracking and limiting volatile state

In order to load regions of data memory only when it is needed, and to avoid writing unmodified data when suspending, tracking of active and modified memory is necessary. While high-performance processors use hardware memory management units (MMUs) to translate and control access to memory [111], low-power microcontrollers suited for EH-powered applications do not have such features. Additionally, MMUs are designed for applications with megabytes to gigabytes of memory footprint as opposed to the kilobytes of a typical embedded application.

ManagedState is a light-weight memory manager that applies the well-known concept of paging to track accesses to volatile memory. In this work, it is implemented and evaluated with bare-metal applications, although it could also be used in conjunction with a typical embedded systems operating system. Like traditional paged memory, *ManagedState* loads pages only when they are referenced, and writes them back to main memory only if they are modified. However, *ManagedState* imposes little overhead because it is much simpler than traditional paged memory systems: it does not perform

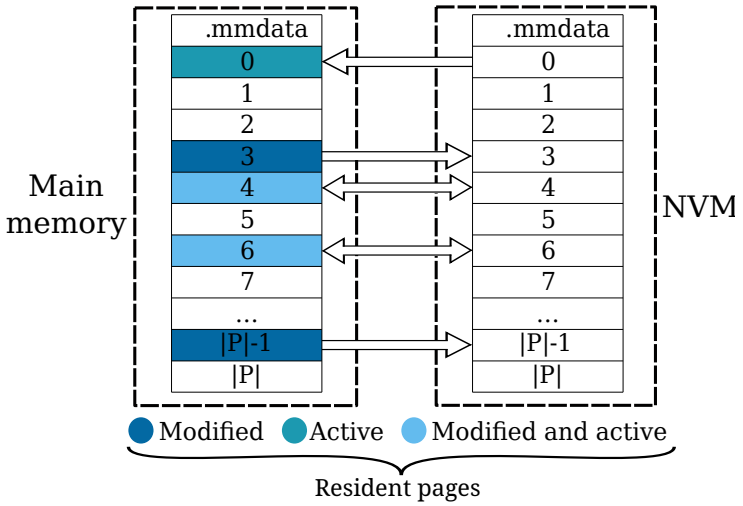


FIGURE 3.3: Illustration of memory operations of *ManagedState* where only active pages are loaded during *restore* (←) and only modified pages are saved during *suspend* (→); pages which are both modified and active must be loaded during *restore* and saved during *suspend* (↔). Pages that are colored in main memory are considered *resident*, i.e. they are loaded.

address translations or relocations, as the entire application is assumed to fit in main memory.

To use *ManagedState*, the application calls *Acquire* before using a block of data with a pointer to the start of the data in main memory, the number of bytes to be acquired and the reference mode (*RO*/*RW*, explained later in this section). *ManagedState* is then responsible for loading the relevant pages from NVM into main memory and for maintaining their persistence through power cycles. When the application no longer needs the block of data, it calls *Release* with a pointer to the data, and the number of bytes.

When possible, it is most efficient to process blocks of data one page at a time; thus keeping only a single page active (or two, if an element crosses a page boundary) without excessive *Acquire*/*Release* calls. For non-linear *RW* access, where data residing in several pages are accessed sporadically, the application must either acquire the entire block (minimal overhead, but keeps the entire block active) or acquire a few elements at a time (minimal active pages, but large overhead). A third option to alleviate the overhead of non-linear *RW* memory accesses is to restructure the application algorithm to improve locality, as demonstrated in § 3.5.2.

ManagedState divides a memory section, `.mmdata` (memory managed data), into a set P of pages p, where each page is of size $|p|$. Accesses to variables located in P are tracked to determine the following three sets at runtime.

- R: the **r**esident set of pages held in main memory
- M: the set of **m**odified pages

- A: the set of **a**ctive, i.e. currently referenced, pages

A reference consists of the operations *Acquire* and *Release*, shown in Algorithms 1 and 2; these two operations maintain R, M and A. There are two types of references:

- *RO* - **R**ead **O**nly reference
- *RW* - **R**ead **W**rite reference

An *RO* reference to a variables causes the corresponding page to be marked as active, i.e. added to A. Similarly, an *RW* reference to a variable causes the corresponding page to be marked as active and modified, i.e. added to A and to M. If the page is not already resident, i.e. not in R, it is loaded into main memory and added to R. The number of references to each page is tracked. When the reference count to a page p is 0, i.e. the page is no longer referenced, it is marked as non-active (removed from A).

When suspending, all modified pages, i.e. pages that exist in M, are saved to NVM. Pages which are in A, but not in M are *not saved*; their values are already guaranteed to be persistent. Pages that are in M, but not in A may be modified, but are not currently referenced; hence they are safely removed from M when they are saved to NVM.

During startup, the volatile state is restored. With *ManagedState* method, only active pages are loaded; that is, only pages in A. All other pages will be loaded only if and when they are referenced.

The pages in M comprise the volatile state of `.mmdata`. *ManagedState* maintains an upper limit $\hat{M}$ (determined in §3.4) on the number of modified pages $|M|$. When an *RW* access that would cause $\hat{M}$ to be exceeded occurs, a page needs to be removed from M. *ManagedState* finds a page in M that is not in A (a page that may have been modified, but is not currently active), saves it to NVM and removes it from M. If no such page can be found, $\hat{M}$ must be increased or the application's usage of *Acquire* and *Release* modified to reduce the number of concurrently active pages. Note that the page is still resident after being saved to NVM, so it does not have to be loaded again the next time it is used within the current on-period. A least-recently-used (LRU) table is maintained to select which pages get removed from M to avoid excessive page thrashing (repeated saving of the same page). A page is never loaded more than once during a power cycle.

3.3 Usage

This section gives a brief introduction to the practical usage of *ManagedState* as a software library. Figure 3.4 shows a code excerpt from an AES workload, and an example of how it can be modified to take advantage of *ManagedState*. Figure 3.4a shows

```
1  #define AES_BLOCK_SIZE 16u; // 128-bit AES
2  static unsigned char buffer [2048];
3  static unsigned char key [16];
4
5  int main () {
6    // ...
7    // Prepare the buffer with sample data etc...
8    // Prepare the key
9    // ...
10
11   // Encrypt results
12   unsigned char *ptr = buffer;
13   while (ptr < buffer + sizeof(buffer)) {
14     aes_encrypt(ptr, key);
15     ptr += AES_BLOCK_SIZE;
16   }
17   // ...
18 }
```

(a) Unmodified code.

```
1  #define AES_BLOCK_SIZE 16u; // 128-bit AES
2  static unsigned char buffer [2048] MMDATA ;
3  static unsigned char key [16];
4
5  int main () {
6    // ...
7    // Prepare the buffer with sample data etc...
8    // Prepare the key
9    // ...
10
11   // Encrypt results
12   unsigned char *ptr = buffer;
13   while (ptr < buffer + sizeof(buffer)) {
14     mm_acquire(/*addr=*/ptr, /*size=*/AES_BLOCK_SIZE, /*mode=*/MM_READ_WRITE);
15     aes_encrypt(ptr, key);
16     mm_release(/*addr=*/ptr, /*size=*/AES_BLOCK_SIZE);
17     ptr += AES_BLOCK_SIZE;
18   }
19   // ...
20 }
```

(b) Basic usage of *ManagedState*. Modifications and additions to the original code are highlighted.

FIGURE 3.4: A code snippet that encrypts a 2 kB string using AES. For clarity, this example is somewhat simplified compared to the AES workload used in the evaluation section (cipher block chaining is ommited).

Algorithm 1 Acquire a variable residing in `.mmdata`.

```
1:  function ACQUIRE(pointer, ReferenceMode)
2:      p ← getPageNumber(pointer)
3:      if ReferenceMode = RW then
4:          while |M| ≥ M̂ do
5:              w ← nextPage ∈ LRU
6:              if w ∉ A then
7:                  saveNVM(w)
8:                  M ← M − {w}
9:          M ← p ∪ M
10:     if p ∉ R then
11:         load(p)
12:         R ← p ∪ R
13:     refCount[p] = refCount[p] + 1
14:     A ← p ∪ A
    return
```

Algorithm 2 Release a variable residing in `.mmdata`.

```
1:  function RELEASE(pointer)
2:      p ← getPageNumber(pointer)
3:      refCount[p] = refCount[p] − 1
4:      if refCount[p] = 0 then
5:          A ← A − {p}
    return
```

the original code, where a data buffer is encrypted block-by-block using 128-bit AES. Figure 3.4b shows an example of modifications and additions to leverage *ManagedState*. First, in line 2, `buffer` is allocated to the `.mmdata` section by use of the `MMMDATA` macro. Note that only the (large) data buffer was allocated to **MMDATA**, all other local and global variables *can* remain as they were. Generally, only variables that are large enough to have a significant negative impact on *restore* and *suspend* operations benefit from being managed through *ManagedState*'s API. In line 14, one 16 B block of data is acquired vi *ManagedState*'s API before being processed in the next line. Finally, in line 16, the block is released. By using the *ManagedState* API in this fashion, only 16 B of the 2048 B `buffer` variable is active at a time. Whereas Figure 3.4a loads the entirety of `buffer` during boot, and save the entirety of `buffer` when checkpointing, Figure 3.4b only loads parts of `buffer` immediately before they are used, and only saves the parts of `buffer` which have been processed when saving a checkpoint.

Note that the *ManagedState* library uses semantics similar to those used for heap allocation in the C language. As such, there is no protection against use-after-free and similar programming bugs. The library also does not implement security features.

3.4 Dynamic suspend and restore thresholds

When the supply voltage drops below the suspend threshold, *suspend* is triggered and execution halted. Later, when supply recovers beyond the restore threshold, *restore* is triggered to resume execution. To maximize the time spent on useful computation, and to minimize superfluous state saving, *suspend* should be postponed as much as possible, as illustrated in Figure 3.2. Hence the suspend threshold V_S should be minimized, while still ensuring sufficient energy for successful state saving. The restore threshold V_R protects against re-execution, so must guarantee that sufficient energy for suspend remains after restoring. For safety, V_R must be calculated under the assumption that no energy is supplied after restore begins. This minimum restore threshold is termed $V_{R,min}$. Practical systems should increase V_R beyond $V_{R,min}$ by a voltage Δv_C that ensures a minimum of computational progress in every power cycle. The optimal Δv_C depends on power supply characteristics and application constraints. For a low-current source, maximizing Δv_C effectively amortizes the cost of suspend and restore. In contrast, minimizing Δv_C makes the device more responsive to events as it wakes up more frequently, but also less energy efficient because less computation is performed per power cycle. For a sparse supply, where energy arrives in short pulses, Δv_C should be small to ensure that every pulse is utilized instead of being wasted through leakage and sleep currents.

Figure 3.5 shows measured voltage traces and operation on a worst-case power pulse, which charges v_{cc} to V_R, then immediately drops to zero. As is typical for microcontrollers, the one used in this work draws nearly constant current (8% variation) over its operating voltage range because of its on-chip linear voltage regulator. Current draw does, however, depend on peripherals, so the application's most power-hungry combination of simultaneously active peripherals should be assumed when calculating suspend/restore thresholds. Constant current draw makes the suspend, restore and compute voltage drops Δv_S, Δv_R and Δv_C, respectively, proportional to their respective execution times t_S, t_R and t_C. Thus, a linear model is suited to calculate the threshold voltages

$$V_S = V_{on} + \Delta v_S = V_{on} + \frac{\Delta v_{CC}}{\Delta t} t_S, \tag{3.1}$$

$$V_R = V_S + \Delta v_C + \Delta v_R = V_S + \frac{\Delta v_{CC}}{\Delta t}(t_C + t_R). \tag{3.2}$$

The voltage drop $\frac{\Delta v_{CC}}{\Delta t}$ can be calculated from the platform's current draw I and capacitance C as

$$\frac{\Delta v_{CC}}{\Delta t} = \frac{I}{C}, \tag{3.3}$$

although measuring $\frac{\Delta v_{CC}}{\Delta t}$ directly is often simpler.

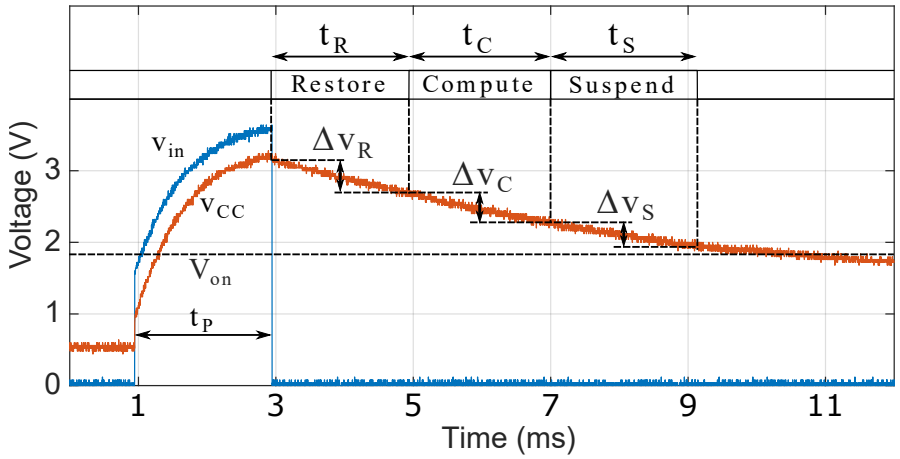


FIGURE 3.5: Safe restore and suspend thresholds, ensuring sufficient energy to guarantee a minimum of useful computation (t_C) and a successful checkpoint.

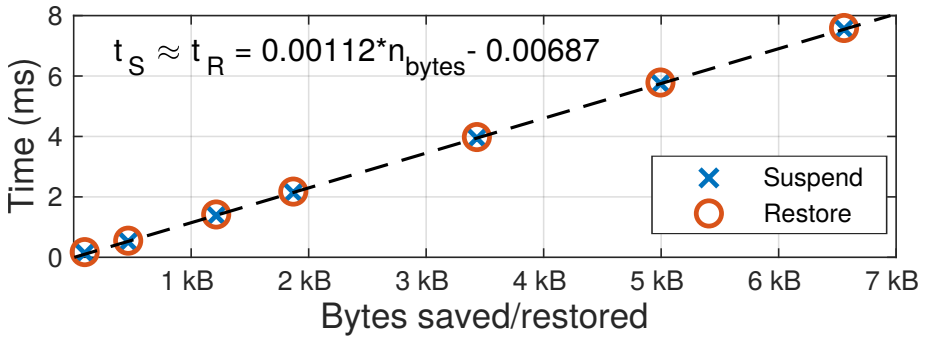


FIGURE 3.6: Execution time of suspend and restore operations.

Furthermore, t_S and t_R are proportional to the amount of data to be saved/restored, as shown in Figure 3.6, where the average execution time of suspend and restore were measured in relation to the amount of memory saved/restored. The dashed line shows the linear fit, confirming that

$$t_S(n_{bytes}) \approx t_R(n_{bytes}) = \alpha n_{bytes}, \tag{3.4}$$

where α is the time it takes to read/save one byte, and n_{bytes} is the number of bytes saved/restored, yields accurate estimation of t_S and t_R. The constant term shown in the figure is negligible for reasonable checkpoint sizes (10 B). Combining (3.1), (3.2) and (3.4) yields

$$V_S = V_{on} + \frac{\Delta v_{CC}}{\Delta t}(\alpha n_{ut} + \alpha|p||M|), \tag{3.5}$$

$$V_R = V_S + \frac{\Delta v_{CC}}{\Delta t}(t_C + \alpha n_{ut} + \alpha|p||A|), \tag{3.6}$$

where n_{ut} is the number of untracked bytes, which must be saved and restored in every power cycle. Hence $n_{ut} + |p||M|$ is the number of bytes written during suspend, and $n_{ut} + |p||A|$ is the number of bytes read during restore. The CPU registers, allocated stack, tables of *ManagedState*, and untracked application variables comprise the untracked memory. The restore threshold has an upper bound V_{max}, given by the maximum output voltage of the supply or the maximum operating voltage, constraining V_R to

$$V_R \leq V_{max}. \tag{3.7}$$

Substituting (3.5) and (3.7) into (3.6) and rearranging yields the constraint

$$|A| + |M| \leq \frac{V_{max} - V_{on}}{\frac{\Delta v_{CC}}{\Delta t}\alpha|p|} - \frac{t_C}{\alpha|p|} - \frac{2n_{ut}}{|p|}, \tag{3.8}$$

determining the maximum number of modified pages at runtime as

$$\hat{M} = floor\Big(\frac{V_{max} - V_{on}}{\frac{\Delta v_{CC}}{\Delta t}\alpha|p|} - \frac{t_C}{\alpha|p|} - \frac{2n_{ut}}{|p|} - |A|\Big). \tag{3.9}$$

During runtime, V_R and $\hat{M}$ are updated when a page is acquired or released, while V_S is updated when the application issues an *RW* reference and when saving modified pages during suspend. Calculating $\hat{M}$ at runtime maximizes the capacity of M, minimizing the number of pages saved during execution, while still eliminating corrupt checkpoints and re-execution by guaranteeing the success of future suspends and restores.

3.5 Evaluation

This section evaluates the performance of *ManagedState* by running a set of experiments to assess memory tracking accuracy and overheads, suspend and restore time, suspend threshold calculations, and finally application performance. But first, the experimental setup, as well as the benchmarks used, are described.

3.5.1 Experimental setup

This work was experimentally evaluated on the *MSP430FR5994 LaunchPad Development Kit*, a development board for the *MSP430FR5994* ultra-low-power microcontroller. This

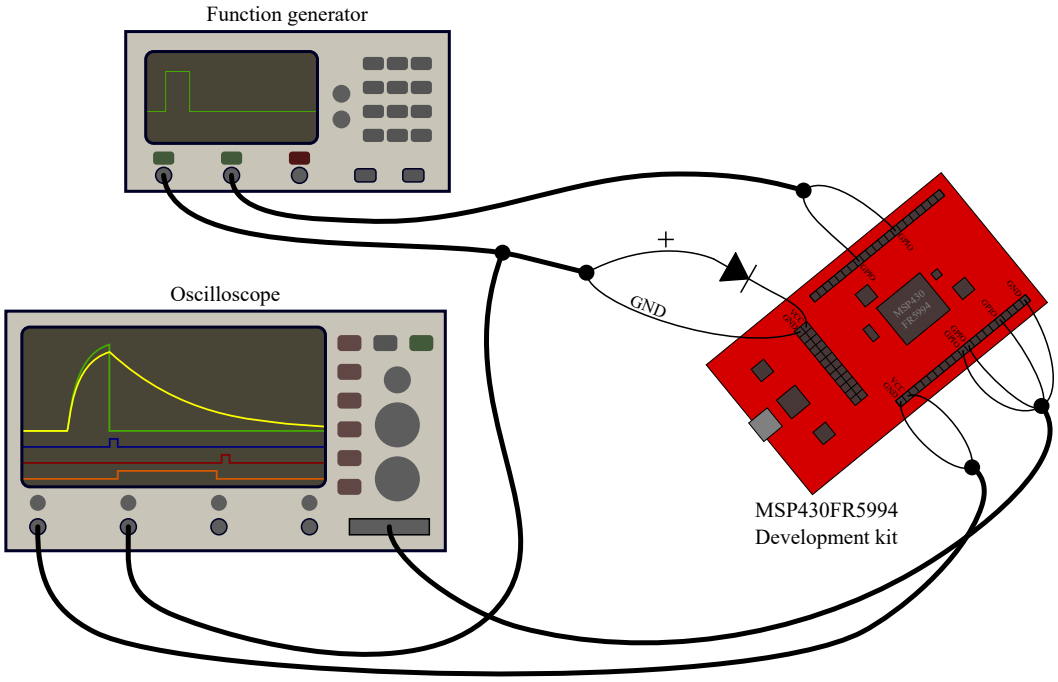
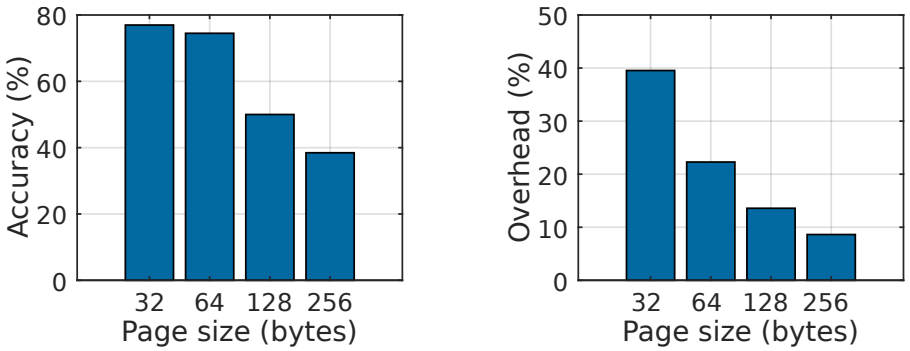


FIGURE 3.7: Experimental setup. The first channel from the function generator is used as a configurable power supply, and the second channel is used (in some experiments) to generate GPIO interrupts. An oscilloscope is used to measure the input voltage, supply voltage, and GPIO pins that indicate activity. The rectifying diode prevents current backflow from the development board to the function generator.

platform was chosen due to its on-chip energy-efficient and byte-addressable FRAM memory; which is more suitable for IC than the traditional flash memory used in most microcontrollers. No energy storage beyond the platform's 10 µF decoupling capacitance was added. This small capacitance limits the amount of state that can be safely restored and suspended in an on-period. For large applications, this necessitates techniques such as this work to limit said state during runtime, as demonstrated in § 3.5.3.

The software used in this evaluation was compiled using the MSP-430 version of the GCC compiler (*MSP430-GCC-OPENSOURCE*), and used the `-Os` optimization target to optimize the machine code for size and performance.

As shown in Figure 3.7, the development board under test was connected to a function generator, which acted as a configurable power supply and a way to generate GPIOs interrupts, and an oscilloscope which was used to measure the supply voltage, the input voltage from the function generator, and GPIOs indicating activity (such as e.g. the start and completion of the suspend operation). A rectifying diode was used on the power input to prevent current flow in the reverse direction (from the development board to the function generator). For some experiments, the power input channel of



(a) (b)

FIGURE 3.8: (a) Tracking accuracy and (b) suspend time overhead, in relation to page size $|p|$.

the function generator was simply used as a constant voltage source, whereas in others it was used to generate voltage pulses, as described for each experiment in their respective subsections.

- **AES**: 128-bit AES encryption of a 2 kB string. Typically used to secure communication.
- **CRC32**: 32-bit checksum generation. Typically used for error checking of communication payloads.
- **MATMUL**: Multiplication of two 25×25 matrices of 16-bit values. Representative workload for signal processing and classification tasks.

AES and CRC32 access memory linearly, hence *ManagedState* tracks their memory accesses efficiently. In contrast, MATMUL, accesses memory in a sparse and repetitive manner, leading to excessive *Acquire* and *Release* calls, hence large overhead. AES modifies the whole data buffer, while MATMUL only modifies the output buffer and CRC32 only modifies a single output variable.

3.5.2 Memory tracking

This subsection validates correct state retention and evaluates the accuracy and overheads of the memory tracking implemented in *ManagedState*. The experiments described in this subsection used the first channel of the function generator as a constant voltage source, and the second channel to trigger suspend operations (via GPIO interrupts).

Efficient and safe state retention through power cycles is the main aim of this work. Memory consistency was verified by taking a core dump (capturing memory and processor state using a debugger) immediately before suspending state, then power cycling the platform, then another core dump immediately after restoring the state. The

TABLE 3.1: Overhead of *ManagedState* on a continuous supply when compared to *AllocatedState*.

Benchmark	Runtime overhead	Memory overhead
AES	1.05×	86 *B* (3.5%)
CRC32	1.14×	86 *B* (3.5%)
MATMUL	6.81×	86 *B* (5.0%)
MATMUL_TILED	1.48×	102 *B* (2.5%)

allocated portion of the two core dumps were confirmed as identical, thus state is consistent through power cycles.

The page size $|p|$ affects tracking accuracy, suspend performance and memory overhead. To evaluate accuracy, the number of bytes saved during suspend by *ManagedState* (i.e. $n_{ut} + |p||M|$) while executing the AES benchmark were compared with the actual number of modified bytes since the previous checkpoint. Results are shown in Figure 3.8. Tracking accuracy decreases with increased page size (Figure 3.8a). Execution time overhead of suspend was measured and compared to the time taken to save the same amount of state without paging. Decreased page size leads to increased suspend overhead (Figure 3.8b) because of the larger number of page attributes to maintain.

The static memory overhead of *ManagedState* consists of the least recently used (LRU) and attribute tables. The LRU table consists of $\hat{M}$ 1-byte page-number entries. The attribute table consists of $|P|$ 1-byte entries; each entry's MSB indicate whether the page is loaded (in R), the next bit indicates whether the page is modified (in M), and the remaining 6 bits constitute the reference counter ($A = \{p \in P : ReferenceCounter[p] > 0\}$). For the remainder of the experiments, $|p| = 128$ B, balancing suspend overhead, memory overhead and tracking accuracy.

ManagedState's memory overhead and application execution time was compared to that of *AllocatedState*, results are shown in Table 3.1. The runtime overheads of AES and CRC32 are both small. MATMUL, however, has very large overhead due to poor spatial and temporal locality. There are two ways to alleviate the overhead of MATMUL. The first is to acquire all three matrices, perform the computation, then release; this alleviates the tracking overhead at the expense of restore and suspend performance. This solution yields nearly identical results to using *AllocatedState*, which also loads/saves the entire matrices at every restore/suspend. The second alternative is to improve locality by e.g. using a tiled implementation — a technique often used for performance improvement in systems with hierarchical memory. Computing MATMUL using 5×5 tiles, reduced the overhead to 1.48 × (MATMUL_TILED in Table 3.1). The matrices were now acquired 3 tiles at a time (2 inputs, 1 output), maintaining superior suspend and restore performance when compared to *AllocatedState*, as evaluated in § 3.5.3. For the remainder of the chapter, evaluation results will be given for MATMUL_TILED, as the raw MATMUL implementation has very large overhead.

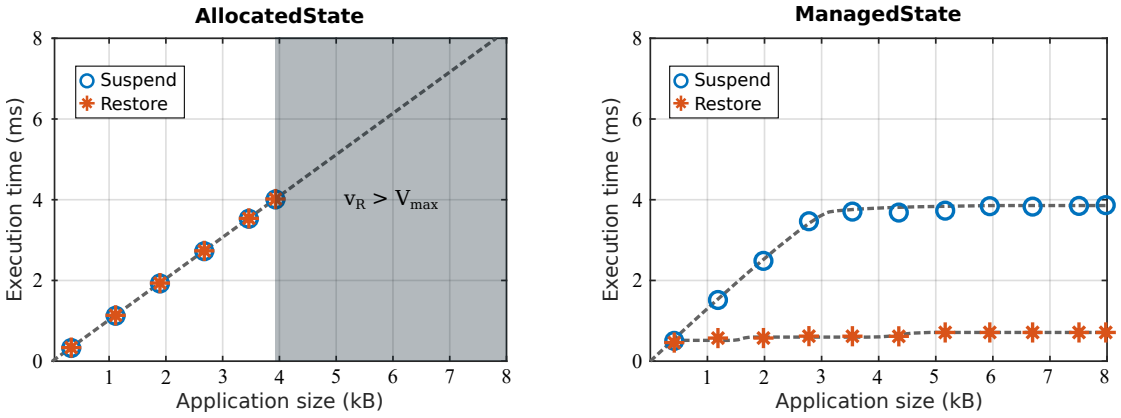


FIGURE 3.9: Suspend and restore time of *ManagedState* and *AllocatedState* in relation to application size.

3.5.3 Suspend and restore time

ManagedState allows large applications to run intermittently. Figure 3.9 shows the result of a comparison between the suspend and restore times (proportional to energy) of *AllocatedState* and *ManagedState* in relation to application size while running the AES benchmark. The size of the string was modified to vary the application size. The experiments described in this subsection used the first channel of the function generator as pulsed power source to emulate intermittent operation while controlling the duration of on and off periods. The second channel was unused. For fair comparison, on-time was set higher than the application's run time. Both methods' suspend times grow linearly until ≈3 kB. *AllocatedState* can no longer find a safe restore threshold after 3.93 kB (shaded area on the figure), because the platform's capacitance cannot store sufficient energy for a safe restore-compute-suspend cycle. The dashed line shows projected suspend/restore time for *AllocatedState*, growing linearly with application size. *ManagedState* limits volatile state according to $\hat{M}$, thus keeping suspend time at a safe level regardless of application size.

ManagedState provides substantial improvement in suspend and restore performance. Figure 3.10 shows measured accumulated suspend and restore time, $t_{S+R} = t_S + t_R$, in relation to on-time. The reduction in t_{S+R} when compared to *AllocatedState* (dashed line) is 26.8–86.9 %, 86.7 % and 49.9–65.3 % for AES, CRC32 and MATMUL_TILED, respectively. For AES, t_{S+R} grows linearly until on-time approaches the run time of the application. For CRC32, t_{S+R} remains constant, because only a single page is active and only a single page is modified, as CRC32 accesses data linearly and only modifies a single output variable. MATMUL_TILED quickly reaches $\hat{M}$, after which *ManagedState* starts saving modified pages to NVM.

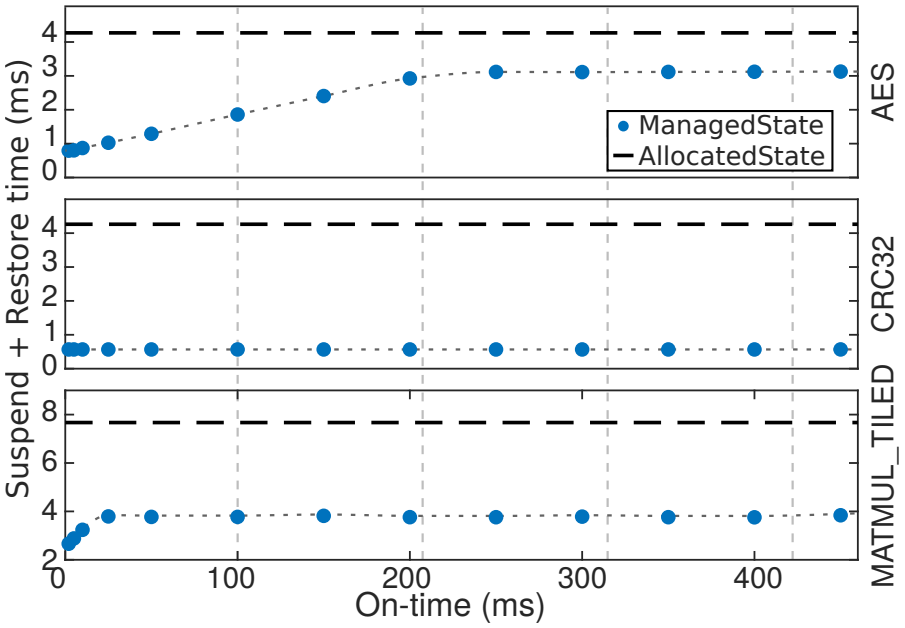


FIGURE 3.10: Accumulated suspend and restore time in relation to on-time.

3.5.4 Suspend threshold

Adjusting the suspend threshold at runtime assures that checkpoints are saved successfully while maximizing energy spent on application execution.

The safety and accuracy of the suspend threshold was evaluated by using an oscilloscope to measure v_{CC} at the start and completion of the suspend operation, v_{start} and $v_{complete}$, respectively. The function generator provided 3.6 V square-wave pulses on its first channel to power the platform, which executed AES. The second channel of the funciton generator was unused in this experiment. The range of on-time spans from a short 10 ms pulse, modifying only a few pages, to a long 350 ms pulse, sufficient to finish the entire application. The results, shown in Figure 3.11, show that the method adjusts the suspend threshold such that corrupt checkpoints are avoided, i.e. $v_{complete} > V_{ON}$, while wasting minimal energy by keeping $v_{complete}$ close to V_{ON}. The variance of $v_{complete}$ increases with the amount of state saved (proportional to on-time), indicating imperfections in the constant-current model of § 3.4. However, to minimize overhead, the computational complexity must be low.

3.5.5 Application performance

Benchmark run times were compared against *AllocatedState* while powered by an intermittent supply, as shown in Figure 3.12. For clear and repeatable results on an intermittent supply, the platform was powered by 3.6 V variable-width square-wave pulses

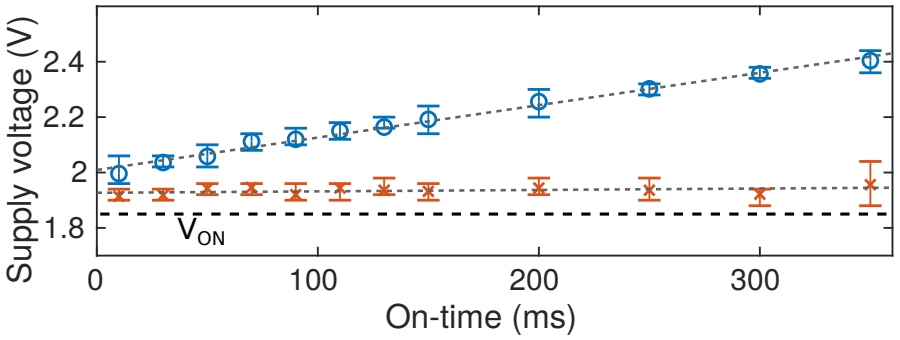


FIGURE 3.11: Supply voltage at the start and completion of suspend. Points show the average of 10 measurements, the bars show minimum and maximum measurements.

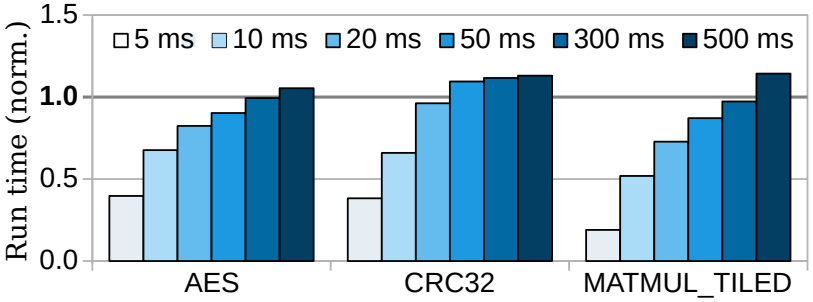


FIGURE 3.12: Benchmark run time relative to *AllocatedState* when powered by square-wave pulses.

from the function generator. Because a rectifying diode was used to prevent the function generator from discharging the platform during off-periods, v_{CC} discharges gradually through the platform's current draw. A voltage trace of this operation mode was shown earlier, in Figure 3.5, where t_P denotes pulse width (note that the figure shows a minimal pulse that with an on-time, t_P, of 2 ms, barely sufficient to charge v_{CC} to v_R before turning off again). The duty cycle of the square-wave was adjusted such that the platform's capacitance was completely discharged to 0 V (through leakage and sleep current) between the pulses. Using a real energy harvester for evaluation would make the results difficult to reason about, and hardly repeatable, because of the complex dynamics of specific harvesters in relation to their environment. Because *ManagedState*'s application performance depends largely on on-time, the square-wave results are readily transferable when considering specific energy harvesters.

As expected from Figure 3.10, *ManagedState* is most effective when on-periods are short, completing the application up to 5.3 × faster than *AllocatedState*. The extra execution time gained by restoring early and deferring suspend, as well as the improved suspend and restore performance, becomes less significant when on-time increases. Hence the run time of *ManagedState* approaches the overheads from Table 3.1 when on-time approaches an application's completion time.

3.6 Discussion

Tracking memory references can alleviate the inefficiencies of existing state retention methods, reducing accumulated suspend and restore time by 26.8–86.9 %, at a small cost in runtime overhead (1.05–1.14 ×) for applications with good locality. For applications with poor locality, tracking becomes expensive (6.81 × overhead), but this may be alleviated by improving locality (1.48 × overhead). After a setup phase, where the user characterizes the voltage drop of the device while saving a checkpoint and set up parameters such as the page size, *ManagedState* provides functionality to limit the amount of state to be saved when power fails, and to adapt suspend and restore voltage thresholds. Limiting the amount of state to be saved when power fails, allows larger application size (`.data` >4 kB) without corrupt checkpoints. Runtime calculation of suspend and restore thresholds capitalize on efficient state retention to improve energy efficiency, while still protecting against corrupt checkpoints. Combining memory tracking using *ManagedState*, and runtime suspend and restore threshold calculations resulted in up to 5.3 × faster workload execution time when on-time was short.

The software developed for this work is published as a library with a build system, examples and usage instructions at `https://git.soton.ac.uk/energy-driven/iclib`.

Pertinent to **Q2**, these results indicate that although hierarchical memory is not typically used in microcontrollers, it may be warranted for IC. A large portion of the overhead incurred by tracking and managing memory could be alleviated by implementing the functionality in hardware, for example with a cache. Hardware support and memory architecture to support IC is are topics researched in Chapter 5. But first, to enable research of hardware support for IC, an appropriate modeling method is required. The next chapter addresses the challenge of modeling IC systems, i.e. **Q3**.

Chapter 4

Modeling intermittent computing systems

4.1 Introduction

In the previous chapter, it was found that tracking active and modified regions of memory could significantly improve the performance of IC systems. However, doing so in software is inefficient, and cumbersome as the programmer is tasked with annotating data usage. A better way would be to use hardware support to perform the tracking and management of data. To research hardware support for IC, though, an appropriate modeling method is required. This chapter addresses that challenge, i.e. **Q3**.

As mentioned in § 2.7.1, the two main approaches to researching new hardware is analytical modelling and simulation. At the level of detail necessary to research **Q2**, simulation is the appropriate approach, as it can enable:

- detailed exploration of new hardware logic; and
- hardware-software co-design, where hardware and software is partitioned, developed, tested and evaluated in parallel.

Simulation is, however, a broad term and can be applied at widely different levels of abstraction. The digital part of integrated circuits, are, for instance, usually designed and simulated using RTL abstraction, then synthesized to lower abstractions (netlists) and simulated further (before eventually being taped out for manufacture). RTL is, however, often too detailed for efficient and flexible design space exploration, thus electronic system level (ESL) simulation tools were created. Among non-proprietary tools, *SystemC*, supported by the open IEEE standard 1666-2011, stands out as the most prominent. It allows modelling at various levels of abstraction spanning from a very

high level using transaction level modeling (TLM) (OSCI TLM-2.0), down to approximately the level of RTL. It also has supporting libraries for surrounding functions such as verification with unified verification methodology (UVM) and modeling of analog and mixed signal (AMS) components. As such, it is a good basis for creating a simulator for IC that addresses **Q3** and can be used to address **Q2** and beyond.

Being governed by energy availability, the operation of IC systems is difficult to reason about, validate, and debug [27, 63, 100]. Traditional development, validation and debugging tools typically assume that energy is always available, rendering them impractical when targeting intermittent operation.

Existing simulation tools used or proposed in prior IC research all lack in one or several important areas, as described previously in § 2.7.3. Table 4.1 and the following paragraph summarizes the previous discussion.

TABLE 4.1: Qualitative comparison of simulation tools pertinent to IC.

Simulator	Energy consumption	Energy input	Closed loop	General	HWSW congruent	AMS
gem5 [17]	✓	✓	✗	✓	✓	✗
NVPSim [157, 163],	✓	voltage trace	✗	✗	✗	✗
Ma et al. [95]	✓	voltage trace	✗	✗	✓	✗
Siren [45]	limited	✓	✓	✗	✓	✗
NORM [121]	✓	voltage trace	✗	✓	✓	✗
Fused (this chapter)	✓	✓	✓	✓	✓	✓

First, the simulator must model energy consumption e.g. energy harvesting. All prior methods have some form of energy consumption modeling, although some are limited. *Siren*, for example, uses an overly simplistic model which simply tallies the count of executed instructions and multiplies it by a constant instruction energy. In reality, the energy per instruction depends on several other factors, such as memory accesses, internal states etc. Second, energy input must be modeled, e.g. in the form of a model of an energy harvester. Again, all methods do model energy input, although some use static voltage traces to do so; thus they are unable to model the effect of the load current on the input energy. The fourth column specifies whether energy and execution is modeled in a closed loop. This is important to model energy consumption and input correctly when execution depends on energy conditions, as is the case in IC. To emphasize this point: consider a device that is about to run out of stored energy, as indicated by the supply voltage dropping below a certain threshold. In reality, that device would likely start performing numerous writes to NVM, which would deplete the stored energy quickly, meaning that the supply voltage drops: the device under test is mainly constrained by energy, not by time. The voltage-trace based systems do not model this – the voltage trace is not affected by the device's power consumption – so the device will seemingly have a time-constraint on checkpointing, rather than an energy-constraint. *Siren* is the only current method that checks this box, the others model execution first,

and estimate power afterwards based on an activity trace. The fifth column specifies whether the simulator is general purpose, in the sense that it can be used to model different hardware architectures, i.e. different CPUs etc. All existing methods except *gem5* and potentially *NORM* are purpose-built for a specific microcontroller, and are thus not general in this sense. The sixth column specified whether each simulator is hardware-software (HWSW) congruent, meaning that the same software binary runs on both the targeted hardware and in simulation. This can be an important aspect for evaluation and hardware-software co-design. *NVPSim* uses mocked peripherals that require specialized driver code, and hence the simulator is unable to execute the same binary as the targeted hardware. Finally, AMS simulation, i.e. the ability to model complex analog and mixed-signal components, is important in IC, as power components (voltage regulators, voltage detectors, capacitors etc.) are analog components which should be modeled as such. This is an area where none of the existing methods tick the box, as none of them are based on a simulation framework that supports AMS.

To address the gap in simulation tools, and as an enabler and accelerator for the research and development of intermittent computing systems, this chapter proposes *Fused*[1], a mixed-signal *SystemC*-based simulator that:

- simulates execution, power consumption, and power supply in a closed loop, thus enabling efficient hardware-software co-design and design space exploration in energy-driven computing;
- hosts a GNU debugger (GDB) server to interface with most software development environments;
- enables debugging functionality across power cycles, and the ability to freeze and step through the dynamic energy state in lockstep with execution;
- executes unmodified binaries to be deployed on real hardware, and can integrate existing CPU emulators with relatively little development effort;
- enables modeling of external circuitry, such as energy harvesters and power management circuits, improving repeatability in evaluation.

Hardware-software co-design, whereby hardware and software are developed simultaneously and continuously tested with each other, will become increasingly important for IC in coming years, as a plethora of new NVMs are emerging [24]. By designing hardware and software simultaneously, the implementation of desired functionality can be partitioned between software and hardware to optimally balance software complexity, hardware PPA (power, area and performance), and flexibility.

[1] **Fu**ll-system **S**imulation of **E**nergy-**D**riven Computers

Byte addressable NVM with low access energy is a key enabler for intermittent operation, where computational progress must be retained through frequent reboots, but a simulation tool such as *Fused* is necessary to determine which of these NVMs to choose, and how best to utilize them.

The key contributions presented in this chapter are as follows:

- *Fused*, a simulation framework tailored towards modeling energy-driven computers, utilizing powerful *SystemC* and *SystemC-AMS* models of computation to succinctly, flexibly and accurately model the interplay between program behavior, digital hardware, and analogue power management circuits.
- An event-based power modeling methodology, leveraging timing-accurate simulation to correlate a small set of high-level events with the power consumption of a COTS microcontroller.
- A case-study that demonstrates hardware-validated simulation of a state-of-the-art intermittent computing system using *Fused*.

This chapter begins by introducing the experimental platform used for evaluation (§ 4.2), then gives an overview of *Fused*'s model architecture (§ 4.3), before describing its high-level event-based power modeling methodology (§ 4.4). *Fused* is then evaluated through experiments (§ 4.5), and finally a case-study shows how it is able to accurately model a state-of-the-art intermittent system (§ 4.6). This chapter is based on [131].

4.2 Experimental Setup

The experimental platform used for the validation of *Fused* is a customized version of the *MSP430FR5994 Launchpad Development Kit*, which was used in Chapter 3. A simplified schematic is shown in Figure 4.1. The development board was modified in order to more efficiently support intermittent operation, and to provide high-bandwidth measurements of the microcontroller's current consumption. During the experiments performed for this work, the clock frequency of the microcontroller was set to 8 MHz. The external circuitry (i.e. external to the microcontroller) consists of a 4.7 µF ceramic capacitor used for energy storage, a load switch that can disconnect the microcontroller from its supply voltage, and a low-power comparator with built-in voltage reference and hysteresis that monitors the supply voltage.

When charging the system from a cold start ($v_{cap} = 0\,\text{V}$), the load switch remains open until the comparator closes it, i.e. once $v_{cap} > 3.5\,\text{V}$. The microcontroller must then activate a pull up on the positive input of the comparator to keep the switch closed. If the GPIO pin is left in a high impedance state (High-Z), the comparator will open the

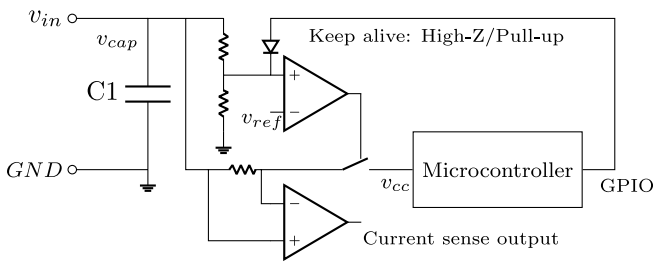


FIGURE 4.1: Hardware test platform.

switch at $v_{cap} = 3.4\,\text{V}$, due to its 0.1 V built-in hysteresis. By enabling pull-up on the GPIO pin, the microcontroller can keep the switch closed, and thus remain operational until it decides to open the switch to recharge C1, or until v_{cap} drops below the minimum operating voltage and the microcontroller browns out. The microcontroller's internal analog to digital converter (ADC) is used to detect power failures, so that it can save execution context to support IC; the internal comparator in the microcontroller could also be used for this purpose. While the microcontroller is powered off, its GPIO pins are left in an undefined state; the diode in Figure 4.1 prevents current from flowing into the GPIO pin in this situation.

The current sense amplifier converts milliamperes of current draw to volts (1 V/mA), and is used for gathering high-bandwidth (100 kHz) current traces of the microcontroller's current consumption. This capability is used to profile the microcontroller's current consumption, to run regression, and for evaluation. The current sense amplifier is powered by a separate supply, so that its power consumption is excluded from measurements. An oscilloscope is used to measure the current sense output, and a logic analyzer for monitoring microcontroller pins; both measurements were triggered by a common GPIO pin.

4.3 Model overview

This section describes the overall architecture of Fused, and the main components used to model the simulation target.

4.3.1 Implementation

To accurately model the complex dependencies between energy and execution inherent in IC systems, Fused simulates energy and execution in a closed loop, as illustrated in Figure 4.2. Note that the figure shows a simplified minimal view, where power management circuits etc. are omitted. An execution model executes the target software binary and outputs a list of simulation events and module states. It also reads input signals from the analog domain, such as the current supply voltage. In the simplest

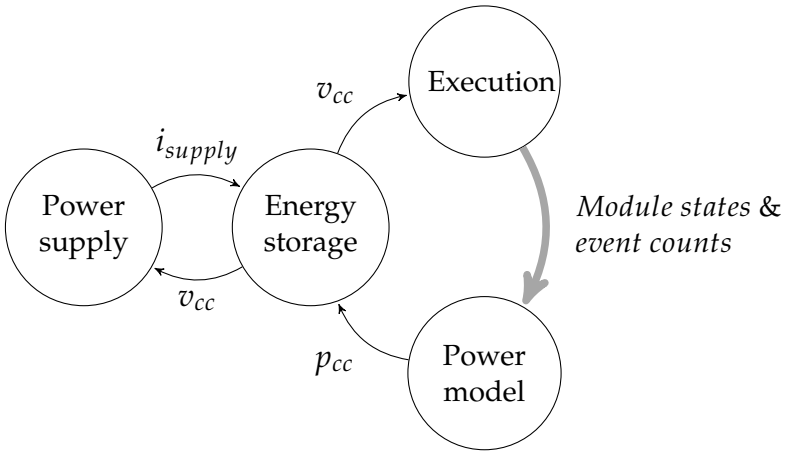


FIGURE 4.2: Abstract view of Fused's closed-loop energy and execution simulation. Thin black lines show analog signals, and the thick gray line represents module states (CPU, memory and peripheral states) and event counts (memory accesses etc.) used by the power model to predict the microcontroller power consumption. All blocks are evaluated in every time step.

case, execution halts and volatile state gets reset when the supply voltage drops below a certain threshold. In more advanced cases, software can, for example, read the value of the supply voltage via an ADC peripheral, and use the result to determine when to checkpoint or restore.

The power model receives the simulation events and module states from the execution module, and uses them to calculate instantaneous power consumption, which is then fed to an energy storage model. On the other side of the energy storage model, a power supply model provides input current (dependent on the supply voltage, i.e. the load voltage as seen from the perspective of the power supply). The energy storage model then calculates the instantaneous supply voltage based on outgoing power, incoming current, and its own capacitance.

In this way, Fused is able to model how execution affects power consumption, how power consumption affects the supply voltage, how the supply voltage affects execution, how power consumption affects input power, and so on, in a closed feedback loop.

Fused is implemented using *SystemC*, an open C++ based IEEE-standardized language for designing and modeling digital electronic devices. *SystemC* brings the advantage of combining several different models of computation to allow a high degree of flexibility for fast design space exploration, while allowing more detailed modeling where necessary.

To achieve high simulation speed and flexible target architecture, most of the modules within *Fused* are implemented using TLM, a high-level fast and flexible modeling methodology which, in simple terms, models hardware as function calls rather than signal toggling. Where more detailed simulation is required, i.e. within peripherals,

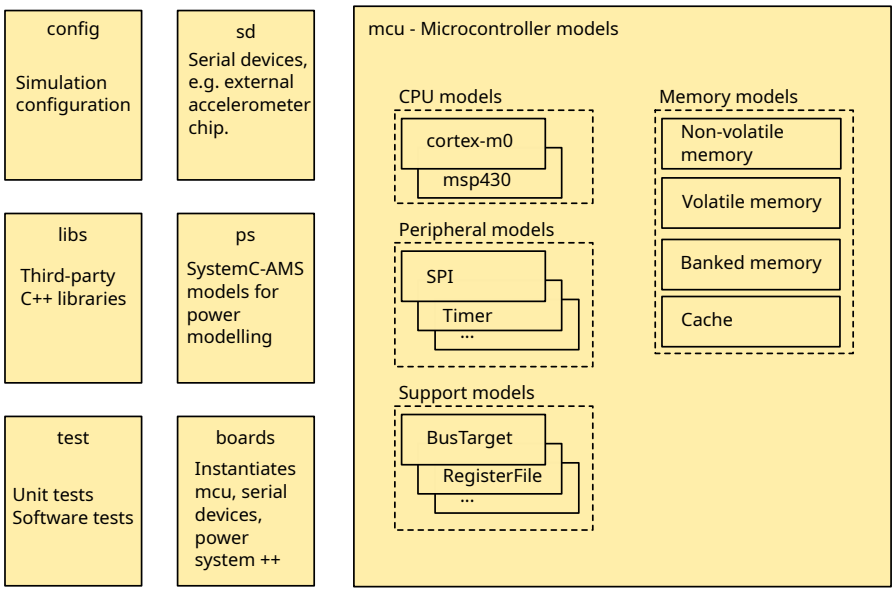


FIGURE 4.3: Illustration outlining how the software implementing *Fused* is organized.

and for interrupts, RTL-like modeling is used. Furthermore, by use of the *SystemC Analogue and Mixed-Signal extensions (SC-AMS)*, complex power supplies and power management circuits are be modeled accurately, flexibly, and in a closed loop with execution.

The software implementing *Fused* is organized as illustrated in Figure 4.3. Apart from support functionality (`config`, `libs`, `test`), the implementation can be roughly divided into modelling the internals of the microcontroller, and the outside world. The microcontroller (`mcu`) model embodies models of CPU(s), peripherals, memories etc., and also contains some support models used to standardize how models interact with each other. The models within the microcontroller generally communicate through a TLM bus, plus some additional interfaces where required. The outside world comprises serial devices, such as an SPI-accelerometer, and power-related circuitry (`ps`). `ps` includes basic electronic components such as capacitors, load switches, voltage detectors and voltage regulators, but also energy sources – i.e. energy harvesters and various power supply models. Finally, the `board` model is used to instantiate all components and connect them together, forming a model of a full PCB. The following subsection describes how *Fused* is used to implement a specific simulation model.

4.3.2 Simulation target

The initial version of Fused targets a typical embedded system, consisting of a microcontroller, power management circuitry, and a power supply, as described in § 4.2. Specifically, the *MSP430FR5994* microcontroller is used as the target for the evaluation in this chapter. This microcontroller has been widely used in the energy-driven computing community because its low-power, byte-addressable and non-volatile FRAM

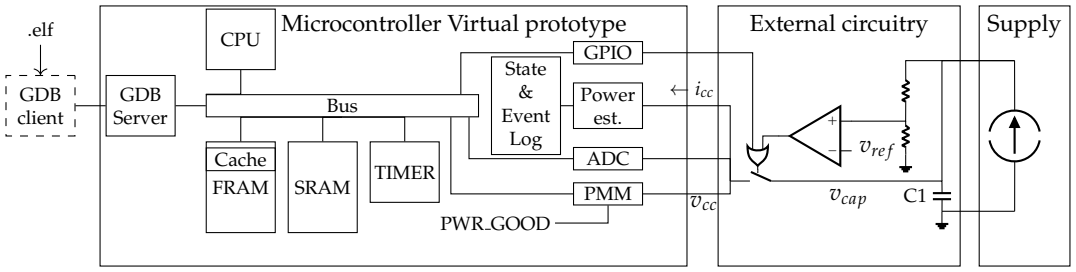


FIGURE 4.4: Model architecture of Fused. This proof-of-concept implementation targets the system shown in Figure 4.1, but all modules within the virtual prototype, external circuitry and supply can readily be modified or replaced.

enables efficient state retention through power cycles. *Fused*'s microcontroller model implements a subset of the modules within an *MSP430FR5994*, including a CPU core, a bus, memories, some internal peripherals, and some externally interfacing peripherals. An overview of *Fused* is shown in Figure 4.4.

The following subsections describe each component of the model.

4.3.3 Software interface

To enable efficient hardware-software co-design, *Fused* hosts a GDB server that acts as the target software interface to the model. This GDB server was adapted from *Embecosm*'s reference design[2]. Through the GDB interface, code can be debugged in the same way as on a hardware platform, i.e. with breakpoints, single-stepping, memory examination etc.

Because *Fused* models power and execution in a closed loop, the analog circuitry also pauses when execution is halted by the debugger. Thus *Fused adds the capability of stepping through code in lockstep with the dynamic power supply and energy storage*. This can be an important feature, e.g. for IC devices which race to finish saving state before the stored energy runs out. Using monitor outputs, such as an execution trace linked to the supply voltage, the programmer can refine the application by iterative testing using *Fused*. When satisfactory results are achieved, the same application binary can be deployed onto the real device.

4.3.4 CPU

Fused includes CPU models for the *MSP430* and the *Armv6-M* instruction set architectures, which execute unmodified binaries to achieve hardware-software congruity. The *Armv6-M* is adapted from the open source *Thumbulator*[3], whereas the *MSP430* model

[2]Documented at `https://www.embecosm.com/appnotes/ean4/embecosm-howto-rsp-server-ean4-issue-2.pdf`. The version adapted for usage in *Fused* is available at `https://github.com/uos-eec/gdb-server`

[3]Available at `https://github.com/dwelch67/thumbulator`.

was implemented by the author. All memory accesses from the CPU models go through the *SystemC* bus, and they use *Fused*'s API to consume simulation time. *Fused* is designed to be flexible, so that other microcontrollers and instruction set architectures can be modeled with minimal effort. To enable comparison with real hardware, the evaluation on *Fused* is focused on an *MSP430*-based microcontroller, which is also currently the most widely used platform in published works on IC due to its energy-efficient on-chip FRAM NVM.

4.3.5 Bus

To allow flexible modeling of different hardware architectures, comprising several heterogeneous peripherals and memories, the bus model is implemented using a TLM interface. Thus bus targets (peripherals etc.) can be flexibly instantiated, attached and address mapped when composing the model. For evaluation, the data and bus width are set to 16 b for the *MSP430FR5994* model. Later, in Chapter 5, the bus is instantiated with 32 b address and data.

4.3.6 Peripherals

Further aiding in flexibility, peripherals implement a common blocking TLM interface. This means that adding new peripherals requires minimal effort as they implement a common interface. Additionally, most peripherals have one or more interrupt request outputs, which are routed to the CPU via an interrupt arbiter. Externally facing peripherals have their ports routed to the top level microcontroller interface, so that they can interact with external digital and analog signals.

4.3.7 Power management module

The power management module (PMM) monitors the supply voltage, and controls the core voltage within the microcontroller, turning it on when the supply voltage reaches approximately 1.88 V, and turning it off when it drops below approximately 1.80 V. *Fused* models this behavior with a signal *PWR_GOOD*. While *PWR_GOOD* is asserted, execution continues, otherwise it is halted. A reset to default values for all internal registers and volatile memory is performed on the positive edge of *PWR_GOOD*.

4.3.8 Memory

The bus communicates with memory through a blocking TLM read/write interface, where the memory module annotates the transaction with access delay. The volatile

byte-addressable SRAM memory has a simple single-cycle access delay. The non-volatile byte-addressable FRAM memory caches reads to reduce power consumption and average access latency. Its access time is nominally one clock cycle, but additional wait states on miss can be added by use of a control register; this is to avoid access time violations when running the CPU at clock speeds above 8 MHz. The cache consists of four 64 B lines arranged in two sets. From experiments with the *MSP430FR5994* (see § 4.5), I found that writes go directly to FRAM, invalidating hits in the cache, and that the replacement policy is likely to be LRU (least recently used). Because the memory model in *Fused* is implemented in TLM, changing the bus parameters, memory sizes and delays, etc. requires minimal effort.

4.3.9 Event & state logging

Fused implements a global logger that records states and event rates at runtime. Each module, i.e. a memory, registers its events during elaboration and reports the event on each occurrence. Similarly, they report their state at the beginning of simulation, and whenever their state changes. Event counts within a configurable time step (e.g. 100 µs) are then aggregated, logged, and reported to the power estimator.

4.3.10 Power estimator

The power estimator computes the current consumption of the microcontroller based on the state and event counts within the current time step. It estimates current consumption rather than power, because of the targeted platform's on-chip linear voltage regulator, which draws (relatively) constant current regardless of supply voltage. This current is then fed to the external circuitry.

4.3.11 External circuitry and power supply

Circuitry external to the microcontroller is modeled in the timed data flow (TDF) model of computation, i.e. essentially a set of equations that are evaluated according to a static schedule. For evaluation, the model includes a constant-current power supply model, also modeled in TDF, but the power model can readily be exchanged for a different supply that e.g. models an energy harvester, or replays EH traces.

4.4 Power modeling methodology

This section describes the proposed methodology for profiling and modeling the energy consumption of a real hardware platform. This is useful both for modeling real

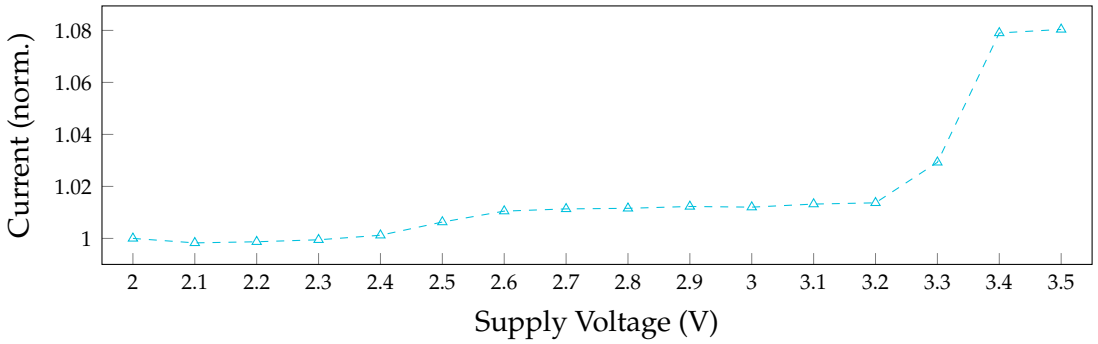


FIGURE 4.5: Average current consumption across all kernels as a function of supply voltage, normalized to the current draw at $2V$.

systems, and for estimating the effect of changes to the hardware. The power model is modular, so if hardware is unavailable, the parameters can also be found from e.g. an RTL power estimation flow. The proposed methodology is similar to ILPM [23], commonly used for high-level power modeling of microcontrollers. However the power model used herein is focused on memory access energy, in place of per-instruction energy, because the target platform's energy consumption depends more strongly on memory accesses than on instructions [66]. The energy consumption of memories is an important factor when developing microcontrollers that utilize emerging NVMs. Another key difference is that the model developed herein relies on far fewer parameters, and thus substantially decreases the burden on writing test programs.

The primary objective of the power model used for this demonstration of *Fused*, is to obtain an explainable model with a high degree of generality, and thereby avoiding over-fitting (rather than demonstrating the highest possible accuracy). To this end, multiple linear regression with a minimal set of explanatory variables[4] was used. The explanatory variables were mostly based on memory accesses. Other users of *Fused* may choose to include more numerous and detailed explanatory variables to gain accuracy, especially if tailoring the model towards specific hardware or investigating the power consumption of specific instructions.

The power estimator of Figure 4.4 estimates power consumption in the current time step based on module states and event-counts. In this context, a module is any part of the model which has a state that may consume current, e.g. a peripheral. To do that, it multiplies each event count with its energy consumption, and similarly sums the current consumption of each module state. The current consumption, i_{cc}, per time step can be expressed as

$$i_{cc} = \left(\frac{\Sigma E_k c_k}{v_{core}\Delta t} + \Sigma I_{m,s}\right) \cdot c_{reg}(v_{cc}), \tag{4.1}$$

[4]Explanatory variables are the inputs to the regression model, used to predict a scalar response. A common alternative term for "explanatory variable" is "independent variable". Similarly "dependent variable" is often used instead of "result scalar".

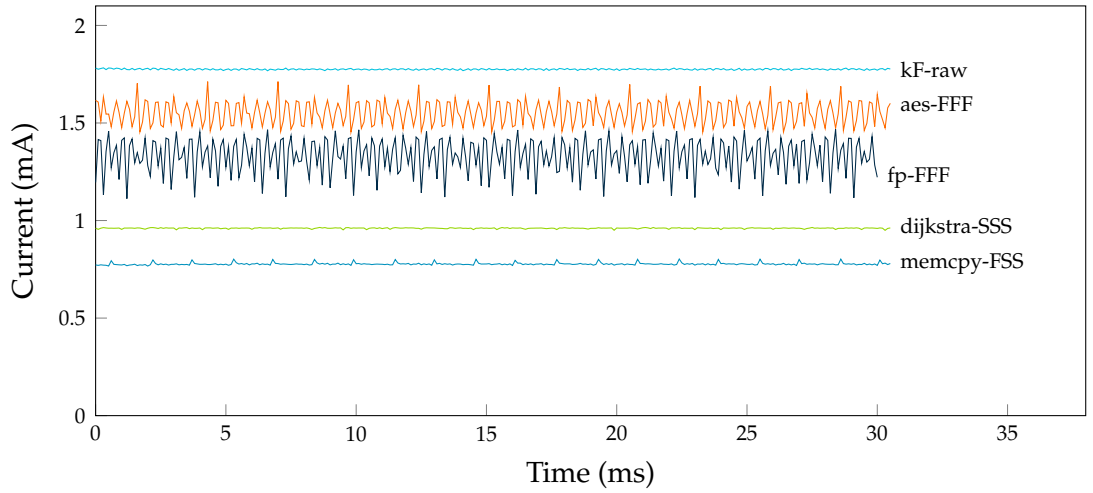


FIGURE 4.6: A selection of measured current traces, showing that current consumption is highly application-dependent, and can exhibit relatively large variations over time.

where E_k is the energy consumption per occurrence of event k, c_k is the number of occurrences of event k within the current time step, v_{core} is the core supply voltage, Δt is the duration of the time step, $I_{m,s}$ is the current consumption of module m in state s, and $c_{reg}(v_{cc})$ is a factor to compensate for current variation as a function of supply voltage. The first term within the parentheses in (4.1) converts dynamic energy (events) into current consumed at the core voltage, the second sums the current consumption of states. Note that the target platform demonstrated herein uses an internal linear regulator to convert the external supply voltage, v_{cc} to the internal supply voltage, v_{core}, and thus should ideally draw constant current regardless of supply voltage (within operating bounds). However, the current consumption increases when v_{cc} approaches the maximum operating voltage of 3.6 V, as shown in Figure 4.5. To compensate for this, the current consumption is multiplied by $c_{reg}(v_{cc})$, implemented as a lookup table of the sample points from Figure 4.5.

4.4.1 Power profiling

To find the energy and current consumption attributable to each event and state, respectively, a power profiling flow based on correlating current measurements from hardware with event and state logs from simulation is proposed in this section. This method leverages timing accurate simulation to pinpoint how much power consumption to attribute to each event.

First, traces of the current consumption of all test programs are measured on real hardware. In this step, it can be beneficial to use a high sampling rate, effectively to gain a large number of samples per test program, so that the model can be robust despite fewer test programs. Figure 4.6 shows three examples of current traces from three test

programs chosen to demonstrate that the average current consumption varies substantially between applications, and, in some applications, also varies rapidly over time. Thus a model that assumes constant power consumption over time and across workloads, is bound to have large errors.

Then, the workloads are simulated to collect event and state logs. A GPIO pin is used to indicate the start of each iteration of the workloads, so that measured current traces and simulated logs can be synchronized temporally. These synchronized event logs and current traces are then passed on to a regression step, which estimates the energy consumption attributable to each state and event.

4.4.2 Selecting explanatory variables for linear regression

Fused can record an arbitrary amount of events during simulation, but not all of these are useful for power modeling. To obtain an explainable and stable model with a high degree of generality, a minimal set of generic explanatory variables was chosen:

- FRAM-RHIT: Read hits in the FRAM cache. These reads are served from the cache, without incurring FRAM accesses.
- FRAM-RMISS: Read misses in the FRAM cache. These cause a read from FRAM, loading a cache line, before serving the data.
- FRAM-W: Writes to FRAM. The written data goes directly to FRAM. If the data is present in the cache, the relevant cache line gets discarded.
- SRAM-RW: Total number of accesses to SRAM.
- EX: CPU execution cycles, here defined as cycles where the CPU does not fetch or store any data.

Fused also records the rate of occurrence of each instruction, addressing mode and more. Including these variables may improve the accuracy of the power model, but would require a vast set of test programs to ensure that every instruction is exercised sufficiently.

For the final regression step, the non-negative least squares (NNLS) method is used to ensure non-negative values for all events and states, corresponding to physical intuition (events cannot consume negative energy). The correlation coefficient of each event k, $coeff_k$, is multiplied by v_{core} and the time step between samples, Δt, to obtain E_k, the energy-consumption per occurrence of the event:

$$E_k = \Delta t \cdot v_{core} \cdot coeff_k. \tag{4.2}$$

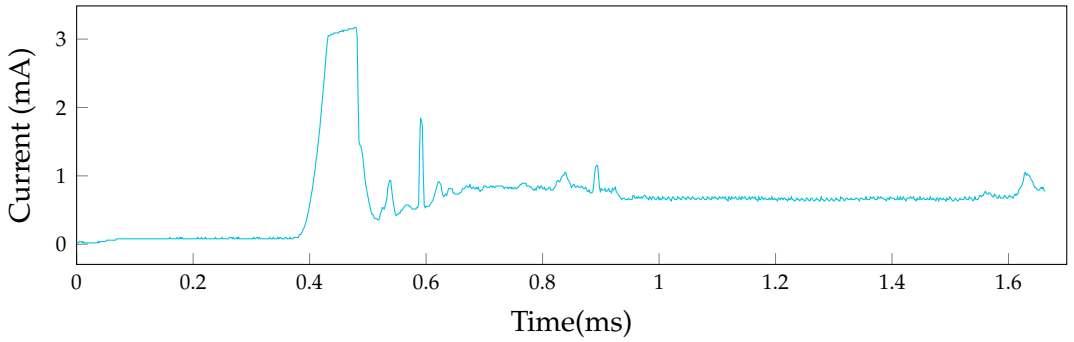


FIGURE 4.7: Current consumption during hardware boot.

4.4.3 Hardware boot

When powering up the microcontroller, energy is consumed while the hardware boots, before any code is executed. The current draw during hardware boot was similar across power cycles, as observed through oscilloscope measurements of the hardware. Figure 4.7 shows the boot-current trace averaged over 128 power cycles, measured by power-cycling the microcontroller while measuring the current consumption with an oscilloscope. Since the real hardware's operation during boot is undocumented, *Fused*'s PMM replays the current trace of Figure 4.7 during boot, and delays execution for its duration. Most simulators can safely ignore this boot current, because the systems they target rarely reboot; for energy-driven computers, however, reboots can be frequent, and may constitute a significant part of total energy consumption (demonstrated in Figure 4.13).

4.5 Experimental Validation

4.5.1 Computational kernels & benchmarks

To generate inputs to the regression model, and to evaluate simulation accuracy, and check for specific microarchitectural features, a suite of computational kernels and benchmarks was assembled. The computational kernels are small assembler programs written to exercise specific operations of the targeted microcontroller. These operations include: read and write accesses to FRAM, read and write accesses to SRAM, sparse and dense reads from FRAM (to exercise the cache), jumps, and copying data between memories. The computational kernels are not necessarily representative of real workloads, but help in validating correct execution, and in generating data for the power

model. The following list gives a short description of each kernel. The source code for all kernels is available at https://github.com/sivertism/msp430fr5994-microbenchmarks.

- `kF`: A series of `MOV` instructions between variables located in FRAM.
- `kF-raw`: A series of `MOV` instructions that write and then immediately read variables located in FRAM. This kernel exposes whether writes invalidate cache data.
- `kF-wsingle`: A series of `MOV` instructions that write to a single variable located in FRAM.
- `kF-wsinglesparse`: A series of `MOV` instructions that write once to each of five sparsely located variables in FRAM in a loop.
- `LRUTest`: A series of `JMP` instructions executed from FRAM that jump to specific locations to test whether the cache replacement policy is "least recently used".

As representative workloads of embedded systems, several computational workloads from the *BEEBS* [109] online repository were ported. The workloads include common mathematical and data structure operations, and cryptographic workloads such as AES encryption, and SHA256 hashing. The data memory footprint (`data+bss` and `stack`) and code memory footprint (`text+const`), as well of a short description of each workload is shown in Table 4.2. For the remainder of this chapter, the term "workloads" will be used when referring to both kernels and workloads.

Because the current consumption of the target platform depends heavily on memory allocation and access patterns, several permutations of

$$\{CODE, DST\_DATA, SRC\_DATA\} \in \{FRAM, SRAM\},$$

were generated, i.e. permutations of allocating each of the three sections (LHS) to each of the memories (RHS). This is denoted in the benchmark names as CDS, where C is the code section, D is the destination data section, and S is the source data section. A kernel with its code allocated to SRAM, that copies data from FRAM to SRAM would thus get the suffix `SSF`. The stack was allocated to SRAM.

4.5.2 Execution time

To evaluate the simulator's execution time accuracy, the completion time of all workloads was simulated and measured on real hardware. A GPIO pin indicated the start and completion of each test program. The geometric mean percentage error, across all benchmarks, between the simulated and measured completion time, was found to be 0.20%, and the maximum percentage error 1.16%.

TABLE 4.2: List of names and memory footprints for workloads used in the evaluation of *Fused*. Sizes listed are in bytes. Source code for all workloads is available at https://github.com/uoS-EEC/fused-workloads.

Name	stack	data+bss	text+const	Description
aes	512	2068	1044	Encryption
dijkstra	512	140	4420	Networking
fp	512	24	19976	Floating point arithmetic
matmul	512	36	5878	Integer matrix multiply
matmul-tiled	512	4126	4078	Tiled implementation of matmul
memcpy	512	222	3864	c-library memcpy
memcpy_asm	512	420	3564	Hand-crafted memcpy
nettle-sha256	512	4270	1162	Secure hash
newlib-exp	512	4522	3102	Exponential of float number (s)
newlib-log	512	220	3136	Logarithm of float number (s)
newlib-mod	512	2420	16	Modulo of float number (s)
newlib-sqrt	512	24	6130	Square root of float number (s)
sglib-arraybinsearch	512	1320	1908	Binary search
sglib-arrayheapsort	512	4570	2862	Heap sort
sglib-dllist	512	4602	12684	Doubly linked list
sglib-hashtable	512	2104	1100	Hash table
sglib-listinsertsort	512	620	5548	List insert sort
sglib-listsort	512	20	2334	List sort
sglib-queue	512	426	5416	Queue
sglib-rbtree	512	4322	1404	Red-black tree

4.5.3 Cache model

To validate the cache model in *Fused*, the simulated cache miss rate was compared to the real hardware miss rate. Herein, the miss rate is defined as being the ratio of cache misses over accesses, i.e. including both reads and writes. However, there are no direct ways of measuring miss rate on the real platform, so an indirect measurement was used.

To measure the miss rate,the execution time was measured while setting the FRAM cache miss penalty, WS, to 0 and to 15 clock cycles; setting $WS = 15$, causes the CPU to stall for 15 clock cycles on every cache miss, and hence the execution time becomes highly sensitive to the miss rate. The miss rate of a test program can then be calculated as

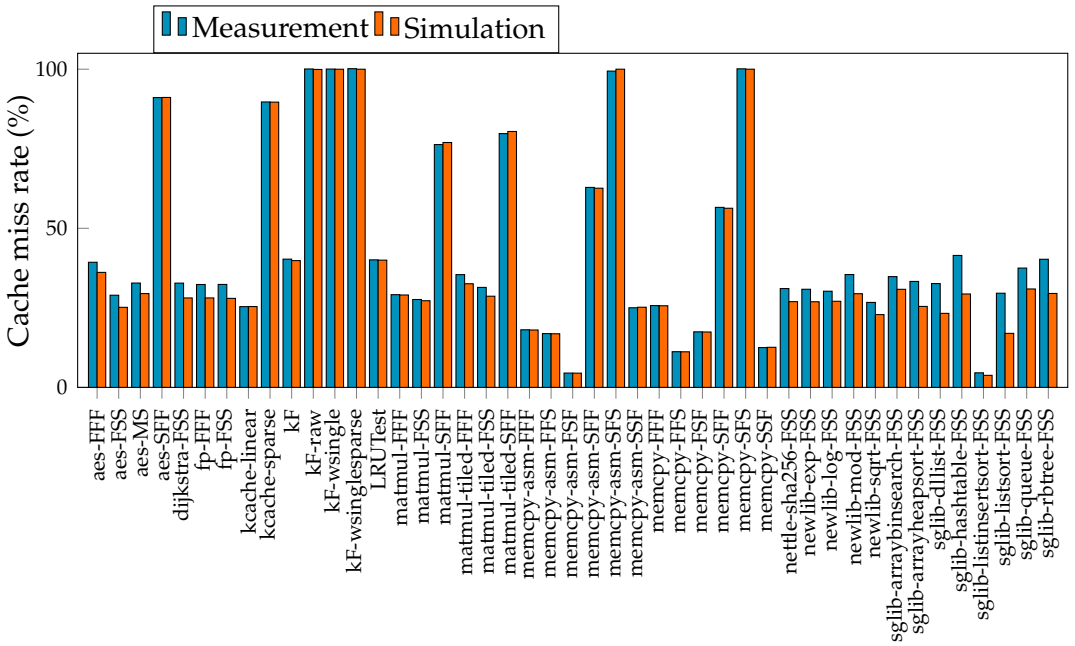


FIGURE 4.8: Cache miss rate across benchmarks with more than 1000 total FRAM accesses.

$$MR = \frac{f_{clk}(t_{WS15} - t_{WS0})}{15} \cdot \frac{1}{n_R + n_W}. \tag{4.3}$$

The first term calculates the total number of cache misses, and the second calculates the access rate. The clock frequency is denoted f_{clk}, t_{WS15} is the execution time when setting $WS = 15$, t_{WS0} the execution time when setting $WS = 0$, and $n_R + n_W$ is the number of read and write accesses to FRAM.

Figure 4.8 shows the measured and simulated miss rate. The geometric mean error between measurement and simulation is 2.69 %. However, the model significantly underestimates the miss rate for certain programs. In the worst case, i.e. for *sglib-listsort-FSS*, the measured miss rate was 30%, but the simulated miss rate was only 17%.

The cache in the *MSP430FR5994* is sparsely described in documentation. The cache is specified as a two-way set associative cache with a total of four lines holding 64 bits each. Writes to FRAM bypass the cache. However, the documentation does not declare which replacement policy and write policy is used.

To ascertain which write policy is used, a micro benchmark, *kF-raw*, that repeatedly writes to and then reads from the same location in FRAM was designed. The cache miss rate was found to be 100 %, meaning that: on a write to FRAM, if the destination data already exists in the cache, it gets invalidated rather than updated. This is often referred to as a write-around policy.

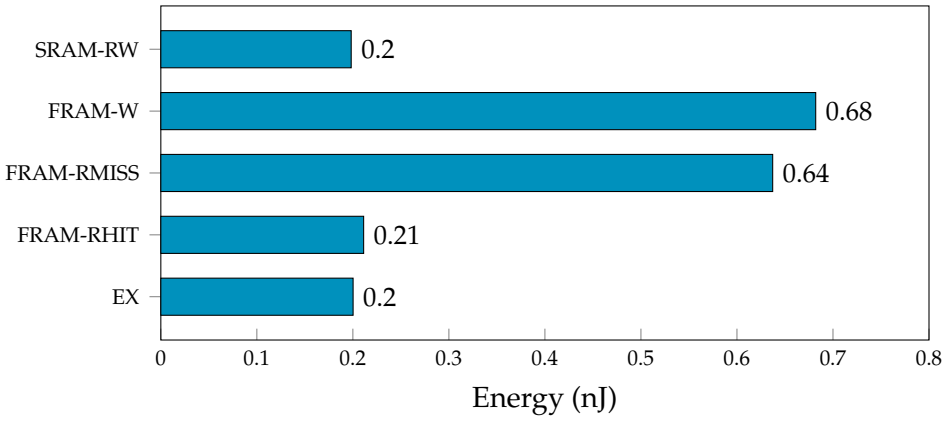


FIGURE 4.9: Energy consumption per occurence of events included in the power model.

The read policy of the cache is not specified in documentation, so LRU, first in first out (FIFO) and least frequently used (LFU) policies were simulated, and the one that best correlated to measurements was selected. Additionally, a micro-kernel, *LRUTest*, was designed to be sensitive to whether the replacement policy is LRU or FIFO (the two most likely candidates). The measurements from the LRU test, and from correlation indicate that the replacement policy is LRU.

The results of this analysis of cache miss rates indicate that the cache used in the *MSP430FR5994* does implement write-around write policy, and a LRU-like read policy, but that there is additional undocumented behavior during reads, that the cache model does not cover.

4.5.4 Power estimation

To run regression and evaluate *Fused*'s power model, a set of 61 workloads were run on the hardware platform. Then 30 ms current traces were recorded, starting from the beginning of each workload (as indicated by a GPIO pin). The measured traces were sampled at 1.25 MHz, and averaged over 128 runs of each workload. Simulations were performed with a time step of 100 µs. The measured traces were then downsampled by averaging, to match the simulation sample rate, before the correlation step; 100 µs was chosen as the sampling rate to allow some degree of temporal misalignment between measurement and simulation.

The estimated energy of each event, obtained through NNLS, is shown in Figure 4.9. Accesses to FRAM are in excess of 3× more costly in energy than those that can be served by the FRAM cache, or SRAM. This implies that, without knowing the cache miss rate of specific applications, power estimates will be severely inaccurate.

FIGURE 4.10: Measured (left bars) and predicted (right bars) current consumption across test programs, sorted according to measured current consumption. The stacked predicted current breaks down the contributions of each feature. The number above each bar denotes the absolute percentage simulation error.

Figure 4.10 shows the measured average current of all workloads on the left bar, and their estimated average current on the right bar. Current is used in place of power because of the targeted platform's on-chip linear voltage regulator. A portion of the workloads were used exclusively for evaluation, in order to evaluate the power model for unseen workloads. The bars for the estimated current also show the contributions attributable to each power model event. In general, FRAM-RMISS and FRAM-W are the strongest two predictors of current consumption, as expected from the device data sheet. The kernels *kF-raw* and *kcache-sparse* are both designed to have very high rates of access to FRAM; the first repeatedly reads and writes to the same FRAM-allocated variable (to test the write policy), the second consists entirely of long jumps that force a cache miss every second clock cycle. On the low-current end are workloads which operate mostly out of SRAM, or have very low miss rates in the FRAM cache. Across all benchmarks, the geometric mean error of the current estimates is 3.4%, and the maximum error is 23.0%.

4.6 Case study: Simulating intermittent computing systems

To demonstrate *Fused*'s ability to simulate truly energy-driven systems, this section presents a case study that models the AES workload running under *ManagedState* from Chapter 3, and compares simulation results to real hardware.

In regards to simulation, an important aspect of *ManagedState* is that, unlike most reactive IC methods, the time and energy consumption of the *suspend* and *restore* operations vary during runtime based on how much state needs to be saved and restored. Furthermore, *ManagedState* adapts its suspend threshold at runtime, depending on the number of pages that need to be saved during the next *suspend* (the adaptive restore threshold was disabled during these experiments, because the external comparator has a constant restore threshold). These complications necessitate simulation over analytical methods.

4.6.1 Experimental setup

A current-limited power source was used to power the hardware platform, and the completion time of a workload that encrypts a 2 kB string using AES was measured on the hardware platform shown in Figure 4.1. This power source setup resembles that of a system connected to a solar harvester, but under more controlled conditions for the purposes of this demonstration. The current was varied from 200 µA to 2 mA in 100 µA steps. To avoid damaging the hardware, the power supply output voltage was limited to 3.59 V. For the following experiments, *ManagedState* was configured to assert a pull-up on the "keep alive" GPIO pin early in the boot process, and to de-assert it at the completion of *suspend*.

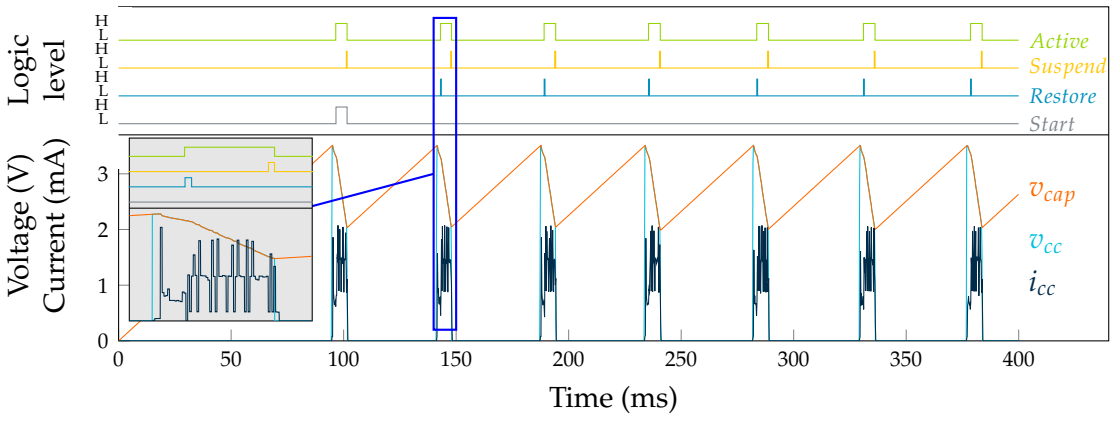


FIGURE 4.11: Simulation trace of a reactive IC system powered by a 200 μA current-limited power supply. The top traces show the logic levels of GPIO pins indicating the operation of the device, and the lower traces show the microcontroller supply voltage (v_{cc}), the storage capacitor voltage (v_{cap}) and the current draw (i_{cc}).

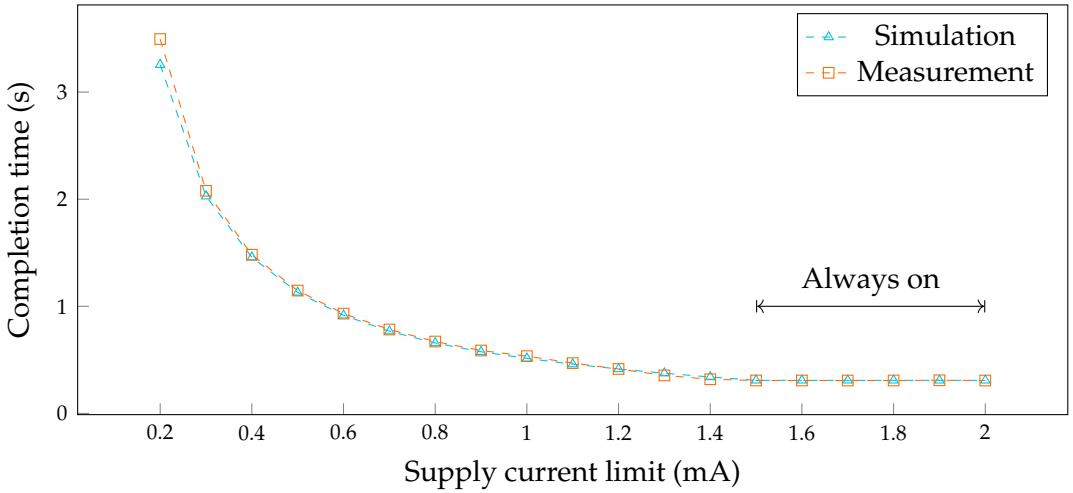


FIGURE 4.12: Completion time of AES workload when running intermittently, powered by a current-limited power source.

4.6.2 Results & analysis

Figure 4.11 shows a sample trace from *Fused*, where GPIO pins indicate when the system was active, the start of the benchmark, when a previous checkpoint was restored, and when state was suspended.

Figure 4.12 shows the measured and simulated completion time of the application, for a range of supply currents, measured as the duration between positive edges on the *start* signal. The completion time increases exponentially with a linear decrease in supply current, mostly due to increased recharging time: as the supply current decreases, on-periods become shorter, and charging periods longer. The maximum error between simulation and measurement was 6.8 %, at the lowest-current measurement point. The mean absolute error was found to be 2.2 %. As indicated on the figure, the platform is continuously on for input currents larger than 1.5 mA.

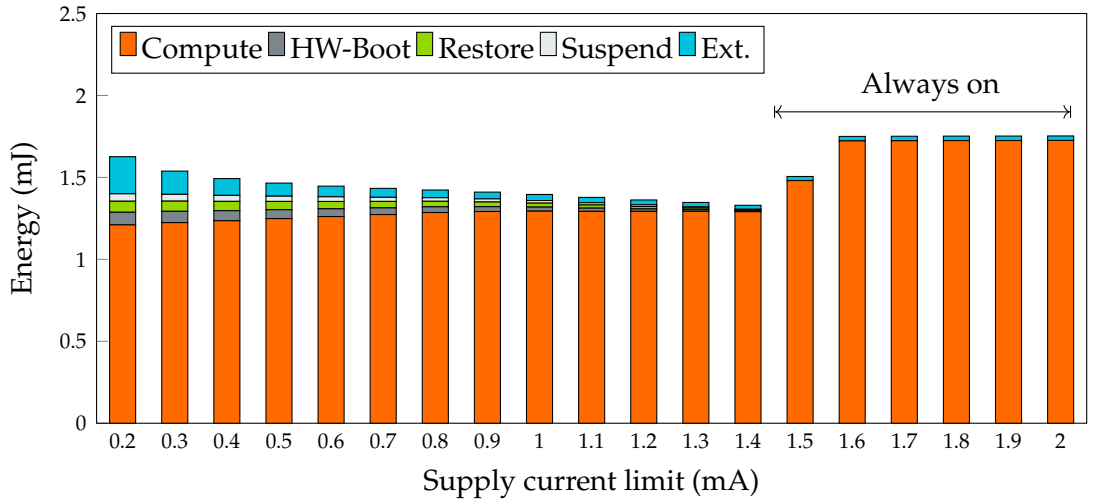


FIGURE 4.13: Full-system energy consumption when running AES encryption intermittently, powered by current-limited power source. The energy consumption is divided into stacked bars for the external circuitry (*ext.*), hardware boot (*HW-Boot*), and the operational phases *restore*, *compute* and *suspend*.

Figure 4.13 shows the simulated energy consumption of the full system, calculated from the simulated current and voltage traces, and divided into components. The energy denoted "compute" is calculated as the energy consumption of the microcontroller while it is active, but not restoring or suspending. The increase in energy consumption at supply current limits exceeding 1.4 mA is caused by an increase in the average supply voltage; beyond 1.6 mA, the power supply is voltage-limited to 3.59 V. The compute energy remains relatively constant for supply current limits below 1.4 mA because the average supply voltage remains nearly constant; this, in turn, is because the comparator has a fixed on-threshold, and *ManagedState* adjusts the *suspend* threshold so that the voltage at the completion of *suspend* remains nearly constant. Hence, the voltage waveform seen in Figure 4.11 remains similar, although temporally stretched/compressed.

The energy overhead of hardware-boot, *suspend* and *restore* grows linearly with the number of power cycles required to complete the workload. Even for the lowest-current measurement point, where one iteration of the workload spans an average of 70 power cycles, the combined energy overhead of *restore* and *suspend* constitute only 6.5 %.

The energy consumption of the external circuitry, however, increases proportionally with total completion time, because it is always powered on. These simulation results indicate that for the *MSP430FR5994* platform, when the supply current is weak, reducing the current consumption of the external circuitry would likely have a larger impact than further optimization of the *suspend* and *restore* operations.

4.7 Discussion

Existing methods of modeling digital circuits are inadequate or impractical for modeling energy-driven computers because they lack flexibility (RTL), accuracy (*gem5*) and closed-loop energy and performance simulation (both). Addressing **Q3**, this chapter presented *Fused*, a full-system simulator for energy-driven computers. Its focus is on closed-loop energy and performance simulation, as well as providing the flexibility needed to explore new hardware and software designs to improve energy-driven computers. Using *Fused*, a developer can rapidly get an accurate picture of the interplay between analog circuitry, digital hardware, and software; thus facilitating IC research.

Fused models execution time with a maximum error of 1.16 % across a broad set of 61 workloads. The power model of *Fused* profiles current consumption of real hardware, and correlates it to simulation events; this resulted in a power model using only five parameters, that achieves geometric mean error of 3.4 %, with a maximum error of 23.0 % across the 61 workloads. Importantly, the power model is explainable; the power consumption is readily attributable to specific functional units and events. e.g. NVM writes and reads.

To evaluate *Fused*'s ability to model truly energy-driven systems, a case study simulated a state-of-the-art intermittent computing system, and validated results against real hardware. *Fused* modeled the completion time of an application running intermittently with a mean and maximum absolute error of 2.2 % and 6.8 %, respectively.

Having shown that *Fused* can model real hardware running intermittently, *Fused* can now be used to model and evaluate new hardware ideas, such as the integration of energy-efficient NVMs and support mechanisms for IC.

It should be noted that *Fused* was used and evaluated within the operating conditions and simulation targets that are most pertinent to the research aims of this thesis. When using *Fused* in research, it is important that the user reviews which variables are part of *Fused*'s execution and power model. For example, the power model used in this evaluation does not consider power drawn from the microcontroller's peripherals; if a user wants to perform research related to the power consumption of peripherals, or using a different microcontroller, she must first update the execution and power model accordingly. Similarly, these experiments were performed at room temperature and with fixed clock frequencies; to make predictions at a wider set of operating conditions, *Fused* must be updated or extended accordingly.

Using *Fused* for modeling and evaluation, the next chapter researches the memory architecture and hardware support for IC, i.e. **Q2**.

Chapter 5

Memory-system support for intermittent computing

5.1 Introduction

Leveraging *Fused* from the previous chapter, this chapter proposes a memory system with hardware-support for IC to address **Q2**.

Whereas the related field of NVPs has proposed several devices that leverage recent NVM developments to enable implicit non-volatility, few works have studied the combination of hardware and software support for IC. To support reliable and efficient IC, both areas need innovation. Utilizing energy-efficient and byte-addressable NVM and specialized circuits is necessary to efficiently perform the core operations of IC. On the other hand, software support is necessary to express application-level mechanisms like atomicity, and to overcome limited hardware resources. By combining appropriate hardware and software support, the result is a device that maintains the programmability of existing software-only methods while reducing software complexity and increasing performance per joule.

This chapter proposes *MEMIC*, a memory subsystem tailored for IC that improves efficiency and reliability while also simplifying IC software. A core tenet of *MEMIC* is to combine volatile and non-volatile memories to get the best of both worlds: volatile memory offers low latency and low per-access energy, whereas non-volatile memory offers persistence, lower leakage power, and potentially higher density. By using an instruction cache, and a data cache tailored for IC, *MEMIC* also simplifies software by removing the need for explicitly loading instructions and data before use. *MEMIC*'s data cache, which implements a software-configurable limit on the size of modified state, also ensures that state is saved before brown-out, regardless of the application's data usage. Naively using a data cache instead of data SRAM would cause bugs when

restarting an aborted FASE. To support FASEs without risking re-execution bugs or resorting to double-buffered checkpoints, *MEMIC* uses a hardware undo-logging module. Both caches reduce the overall energy consumption compared to systems with only NVM, or NVM and SRAM. Unlike many software-based IC methods, *MEMIC* is also compatible with complex industry-standard libraries such as *CMSIS5*, without needing to modify (or even recompile) them.

The main contributions of this chapter are:

- *MEMIC*, a memory system comprising volatile caches, non-volatile main memory, and a hardware undo logger to enable energy-efficient IC that completes workloads 13–39 % faster, using 13–39 % less energy, and operates under conditions where state-of-the-art systems fail.
- Data cache specializations for IC which minimize writes to NVM, and provide an adjustable bound on suspend-energy that is independent of application software.
- Hardware undo logger that provides protection against re-execution bugs when restarting failure-atomic sections.

5.2 Memory-System Support for Reactive Intermittent Computing

This section describes the design of *MEMIC*. It begins by specifying design objectives followed by a description of the target technology, then presents the top-level architecture, followed by detailed descriptions of the features that *MEMIC* implements to achieve the objectives.

5.2.1 Objectives

For the design of *MEMIC*, the following design objectives were set, derived from Chapter 2 and relevant literature [97, 115, 132, 137, 138]:

1. support FASEs, with minimal roll-back cost;
2. software-configurable limit of energy needed to suspend state, i.e. to back up volatile data in NVM;
3. minimal writes to NVM;
4. minimal suspend, restore, and roll-back energy;

5. minimal software complexity.

Firstly, *MEMIC* should support FASEs, as discussed in § 2.1; for the memory system, this implies support for rolling back state in case a FASE is aborted. Secondly, a limit on the energy it takes to suspend is required to guarantee that every suspend operation succeeds despite the finite energy buffer. This limit should be software-configurable such that the same *MEMIC* integrated circuit can be employed across a variety of applications, on a variety of printed circuit boards. Furthermore, this enables resiliency against adverse effects such as capacitor degradation over time and temperature by correspondingly adjusting the limit during deployment. In practice, this limit on the suspend energy can be implemented as a configurable limit on how many modified bytes are held in volatile (cache) memory. When this limit is reached, some modified state has to be saved in NVM before more modified state can be added. The final three objectives are optimization goals. Minimizing writes to NVM as well as suspend, restore, and roll-back overheads leads to better end-performance (i.e. application completion time). Minimizing software complexity eases the adoption of IC into a myriad of applications, ultimately accommodating widespread adoption of battery-less computing devices.

Prior works have demonstrated software implementations of the first two objectives. FASEs can be supported by data versioning or by inserting an extra checkpoint immediately before executing the FASE [12], as discussed in § 2.6. A limit on suspend-energy can be achieved by using software to track and limit modified state, as was done in Chapter 3. However, tracking modified state in software introduces extra software complexity by requiring special annotations, and can significantly degrade both performance and energy efficiency.

Since the core operations of IC (restore, checkpoint, roll-back aborted FASE) consist of various forms of memory access tracking and control, they can be performed much more efficiently and transparently with hardware support than a pure software implementation. *MEMIC* will therefore employ novel hardware support to achieve the design objectives.

5.2.2 Target technology

Fused was, in Chapter 4, developed to target the *MSP430* CPU in order to enable validation against real hardware on a highly relevant platform and show that *Fused* simulations do indeed reflect real-world experiments. However, as *MEMIC* includes proposed hardware changes, the choice of target architecture was reevaluated. The *MSP430* architecture is a proprietary 16-bit CPU architecture available only from a single vendor (*Texas Instruments*) for a limited series of ultra-low-power microcontrollers. The majority of contemporary ultra-low-power microcontrollers are, however, based on the

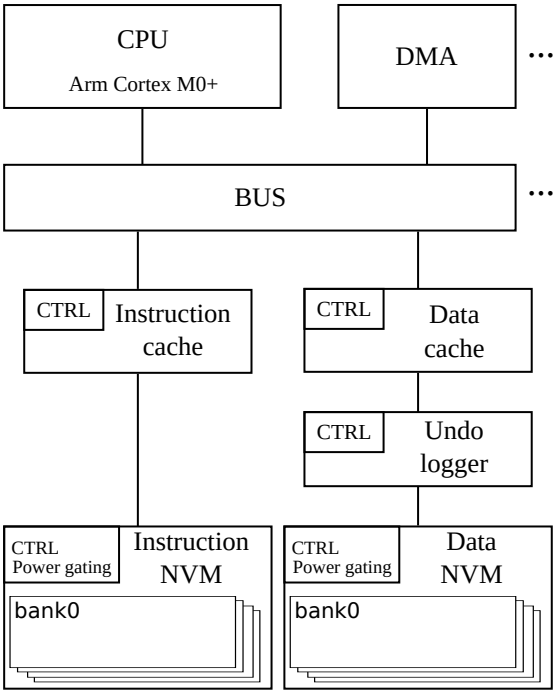


FIGURE 5.1: Top-level architecture of *MEMIC*.

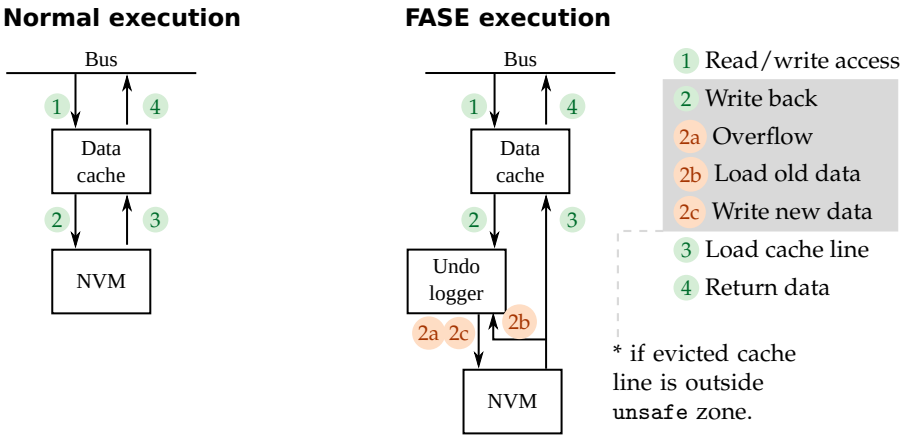
Arm Cortex-M series of 32-bit CPU architectures. These are available through licensing from *Arm*. Microcontrollers using an *Arm Cortex-M* CPU are today available from most microcontroller vendors.

To make *MEMIC* relevant to a broad set of microcontrollers, *MEMIC* therefore targets the *Arm Cortex-M0+* CPU. *Arm Cortex-M0+* has the lowest power and cost among the *Cortex-M* series, and is thus likely the best fit for IC. It should be noted, however, that *MEMIC* does not rely on a specific CPU architecture or specialist instructions, so porting *MEMIC* to a different CPU architecture is likely be straightforward.

On NVM technology, Chapter 2 found that MRAM is the primary candidate to replace flash memory in advanced process nodes for microcontrollers. MRAM is proven to scale to smaller device geometries, and is commercially available from foundries [50], and as standalone memory chips [134]. Microcontrollers using embedded MRAM are also beginning to emerge [5]. *MEMIC* therefore primarily targets MRAM NVM, although the proposed hardware and architecture could likely be used for other NVMs as well.

5.2.3 Top-level architecture

To achieve the design objectives, *MEMIC* leverages existing memory subsystems like caches and an undo-log, specializes these for IC, and arranges them in a synergistic architecture that covers their weaknesses. Figure 5.1 illustrates the top-level architecture,



(a) Data access during normal execution (undo logger disabled).

(b) Data access during FASE execution (undo logger enabled).

FIGURE 5.2: Overview of the *MEMIC* architecture, and data accesses during normal (undo logger disabled) and FASE execution (undo logger enabled). Annotatated steps are numbered in the order they are performed, although not all steps are performed for every memory access.

comprising a volatile instruction cache, a volatile data cache, a hardware undo logger, and non-volatile data and instruction memories. The overall goal is to combine volatile and non-volatile memory in a way that yields good energy-efficiency, by serving most accesses from volatile memory (cache) and carefully preserving state in NVM.

Caching of data and instructions is a well-known method which is used extensively to speed up memory accesses in high-performance computing. In the context of low-power computing, however, caching can also provide reduced energy consumption. The *MSP430FR*-series of low-power microcontrollers, for example, use a read-cache to reduce the energy consumption of their FRAM.

This chapter shows that caching has further benefits in the context of IC. First, they mitigate the overheads of frequently rebooting, by loading data from NVM on demand instead of loading the entirety of data (and possibly instructions) to a separate SRAM during boot. Second, they inherently limit volatile state; a property *MEMIC* leverages and expands upon to enable software-control of the maximum suspend energy.

Employing data caching instead of a separate volatile memory can, however, complicate FASE support because software has less control over which variables are volatile and which are persistent, and must therefore assume that all variables are persistent. *MEMIC* solves this by using a hardware undo-logger. In a conventional hardware architecture, using an undo-logger would cause large overheads, as it logs every write access to NVM. However, in *MEMIC*, the undo-logger is placed behind the data cache, and thus only sees very infrequent write accesses (as shown later in §5.3.8).

MEMIC's caches both use pseudo-random replacement policy, which is a suitable for constrained devices due to its simplicity and robustness against cache-thrashing. The instruction cache is a write-through cache, as writes to instruction memory are assumed to be very infrequent (except during programming). The data cache, detailed in § 5.2.4, is a write-back cache to minimize NVM writes (and thus also pressure on the undo-logger during FASEs). Note that the caches and the undo logger are attached to the NVMs instead of being attached directly to the CPU. This arrangement ensures that memory accesses by peripherals (such as direct memory access (DMA)) are treated the same as CPU accesses, and thus will not break idempotency or state retention. Figure 5.2a and Figure 5.2b show memory operations during normal execution, and FASE execution, respectively; these are detailed in § 5.2.4 and § 5.2.7. The `unsafe` zone allows certain memory accesses to bypass the undo logger, as detailed in § 5.2.7.

MEMIC requires that the platform has *some* energy stored in a capacitor (in the order of 5 µJ), and that it has a supply voltage supervisor that signals a voltage warning when the supply voltage has dropped below a fixed threshold. The voltage warning can be achieved using a single voltage detector, and is usually already part of brown-out detection circuits on microcontrollers.

MEMIC provides software-configurable parameters to ensure portability across hardware platforms (i.e. different end-devices with varying capacitance and power consumption).

The key features implemented by *MEMIC* to achieve objectives 1-5 are as follows:

- a replacement policy that minimizes writes to NVM;
- `MODMAX`: a memory-mapped read/write register that sets a hard limit on the number of modified cache lines;
- **Undo-log:** Logging-support used for cache write-backs during FASE execution;
- `unsafe` **zone**: a region of memory that is excluded from undo logging.

These features are detailed in the following subsections.

5.2.4 Minimizing writes to NVM and limiting volatile state

Due to the relatively high write-energy and low endurance of NVM, design objectives 2 is to minimize writes to NVM. An effective way of achieving that is to employ a write-back data cache upstream of the NVM (shown in Figure 5.1). In contrast to write-through caches, which update both the internal version of a cache line and the downstream memory (NVM), write-back caches only update the internal version and thus

reduce the number of NVM writes. Figure 5.2a shows the memory transactions to and from the cache. Most read or write accesses from the bus (chiefly from CPU or DMA), 1, are served by the cache (cache hit). Writes then only perform step 1, and reads perform 1 and 4 (return data). Read and write hits thus finish in a single clock cycle. However, if the cache does not contain the accessed cache line (cache miss), it has to load it first (3), consuming an extra clock cycle (assuming the NVM can be accessed in one clock cycle). If the cache has to evict a modified line to make room for the new line, a write back (2) occurs before the load (3); this cache miss with write back takes three clock cycles in total.

Cache evictions occur due to aliasing and limited capacity. The number of such evictions generally decreases with increased cache capacity and associativity[1]. Increasing the capacity or associativity, however, also increases power consumption and area. In addition to these well-known trade-offs, the configuration of the data cache also affects suspend energy: flushing a larger cache requires more energy.

An additional approach to reducing the number of write-backs in an associative cache is to bias the replacement policy, which selects the line in the set to be replaced when a cache miss occurs. By biasing the replacement policy such that it prefers evicting unmodified lines, the number of write-backs is reduced. This is achieved by modifying the data cache replacement policy such that, within a set, it always evicts an unmodified line if there is one; modified lines are only evicted if the set is filled with modified lines.

However, if the cache fills up with modified lines, the volatile state may grow larger than can safely be persisted on a power failure. Furthermore, some end-devices may have very stringent limitations on the energy buffer (capacitor). To address this issue, *MODMAX*, a hard limit on the number of modified lines in the data cache is proposed. It guarantees that the volatile state of the data cache never exceeds the limits of the energy buffer. When the total number of modified lines in the cache reaches *MODMAX*, the cache automatically evicts a single[2] pseudo-randomly chosen modified line so as to make it clean. The limit is software-configurable so that individual devices can be tuned for process variation, runtime variation and degradation over time, without requiring a variable suspend voltage threshold. In the simplest case, an appropriate value for *MODMAX* is found for a specific device, and set to a constant value during boot. In more advanced use cases, it can be adjusted throughout deployment. For example, as the storage capacitor degrades over the deployment lifetime of the device, the limits can be reduced correspondingly so that the device remains functional.

[1]Organizing a cache with set-associativity is common practice to reduce the number of conflicts (aliases) by enabling each cache line to have more than one possible storage location in the data array.

[2]Evicting multiple modified lines upon reaching *MODMAX* was also considered, but there is no benefit to spending *N* extra cycles to proactively write back *N* modified lines, versus spending one extra cycle to write back one line every time *MODMAX* is hit. In fact, evicting a single line at a time is preferable because it never writes back more lines than is necessary.

Without *MODMAX*, the cache size would be limited by the minimum expected energy buffer. Or, put the other way, the chosen cache size would impose a minimum capacitor value on the end-device throughout its lifetime.

5.2.5 Suspend and restore during normal execution

When a power failure occurs during normal execution, a checkpoint is taken so that execution can resume from exactly where it left off when power returns. The suspend operation simply entails saving the processor registers to memory, then flushing the cache. This operation is powered by buffered energy alone, as one must assume the worst case where the power source has cut off completely.

When restoring after a power failure, the state is restored simply by loading processor registers. The cache logic automatically loads data as and when they are needed.

5.2.6 Suspend and restore during failure-atomic sections

Conversely, when a power failure occurs during a FASE, execution needs to restart from the beginning of the FASE, hence the current state must be discarded. To ensure that code outside of the FASE never gets re-executed, and to persist writes that occurred before the FASE, a checkpoint must be saved immediately before starting the FASE. When a FASE is aborted, the (volatile) state kept in the CPU registers and the cache is implicitly reset. Data which have been updated in NVM during the FASE, however, must be rolled back before execution can continue in the next on-period. *MEMIC* uses an undo logger to roll back such data, as described in the next subsection. Since the cache and CPU registers are not backed up when a FASE is aborted, the whole energy buffer can be used to apply the undo log, i.e. to roll-back persisted state to the beginning of the FASE.

5.2.7 Undo logging module

Because of possible memory corruption due to write-after-read and repeated-I/O violations (see § 2.3), it is imperative that any data that has been written to NVM during a FASE is rolled back before a FASE is restarted. Ideally, no data would be written to NVM during FASE execution; roll-back would then simply consist of invalidating the cache and loading the checkpointed processor registers. However, due to limited cache capacity, write-backs may occur during FASEs too.

Taking inspiration from task-based IC [92], *MEMIC* employs undo-logging. However, in contrast to prior works, *MEMIC*'s undo-logger is only active while executing FASEs,

is implemented in hardware, and is placed behind a write-back data cache. These are three factors which greatly reduce the energy overhead of the undo-logger, and also reduces its required logging capacity. When disabled, the undo logger simply forwards all writes to NVM without delay.

Figure 5.2b shows memory operations while the undo logger is enabled. Steps 1, 2, 3 and 3 are the same as during normal execution, except that write-backs to locations outside the `unsafe` zone are intercepted by the undo logger. When enabled, and a cache line write back (2) to an address outside the `unsafe` zone (see § 5.2.8) occurs, the undo logger first reads the old cache line from NVM and saves it, along with its address, in an internal volatile memory (2b). Then, the write back is forwarded to NVM (2c), completing the write back. To prevent overflow, the undo logger automatically saves the oldest entry to NVM (2a) when the size of the internal log exceeds a software-configurable threshold. The overflowed entries are saved at a location defined by a memory-mapped register. Software can then load and apply overflowed entries at the start of the next on-period. The overhead caused by the undo logger during cache write-backs is one cache line load (one clock cycle) when not overflowing, and an additional cache line write when overflowing (for a total of two clock cycles).

An alternative to undo logging would be to use a volatile write-back buffer that buffers cache write-backs during FASEs, and only commits the writes to NVM after the FASE has completed. With some added logic, this could have the advantage of reducing the number of NVM writes through write-merging. However, there is a drawback with this technique when it comes to IC: the volatile state at the end of a FASE would then comprise both the modified cache lines and the lines held in the write-back buffer. This, in turn, would increase the necessary amount of energy buffering; when committing a successful FASE, a larger amount of data would have to be written to NVM at a critical point of execution. In contrast, when using an undo logger, the volatile state to be backed up is never larger than the maximum modified cache lines or the maximum undo log size, whichever is bigger. Both the undo log size (threshold) and the limit on modified cache lines are software-configurable.

5.2.8 The `unsafe` zone

The main use case for FASE is likely tasks such as taking sample windows of sensor data (e.g. audio or acceleration) or receiving radio packets, similar in nature to the *sample_window* function in Listing Figure 5.3. These are tasks that read a potentially large amount of data into structures that can readily be made insusceptible to write-after-read and repeated-I/O hazards; i.e. they can safely be overwritten if the FASE is re-executed, without first resetting the state.

```
UNSAFE uint32_t [WINDOW_SIZE] data;

void sample_window(uint32_t *data) {
  for (int i = 0; i < WINDOW_SIZE; ++i;) {
    data[i] = readSensor();
    sleep(10, TIME_US);
  }
}

void main () {
  run_atomic(sample_window, data);
}
```

FIGURE 5.3: A simple sensor-sampling FASE showing the usage of the `unsafe` zone and how FASE can be annotated.

To reduce pressure on the undo log for applications that write excessive amounts of data (approaching the data cache size) during a FASE, the proposed undo logger implements an `unsafe` zone: *a region of memory which bypasses the undo logger*. The term *unsafe* is used to express that the protection normally provided by the undo logger does not apply; i.e. the programmer becomes responsible for avoiding re-execution bugs for variables they choose to allocate to the `unsafe` zone. A pair of memory mapped registers are used to define the base and bound of the `unsafe` zone. The `unsafe` zone is opt-in rather than opt-out, so that an unmodified program will execute correctly, albeit with sub-optimal performance. While specific variables that are allocated to an `unsafe` zone are excluded from the undo log, and thereby not protected against write-after-read and repeated-I/O hazards, all other variables are protected by default. In Listing Figure 5.3, the programmer knows that `data` can be safely ignored from logging, so allocates it to the `unsafe` zone to improve performance.

While this section has assumed manual allocation of data arrays into `unsafe`, methods to automatically detect write-after-read and repeated-IO hazards [137] could be applied to automatically allocate data to the `unsafe` region.

5.3 Evaluation

This section evaluates the performance of *MEMIC* against several relevant baselines. After describing the experimental setup, workloads and baselines, cache configuration is discussed. Then, the instruction cache and the data cache are investigated separately and compared to their relevant baselines. Next, operating conditions for *MEMIC* and the baselines are evaluated. Then follows a case study on a solar-powered IC device running a realistic sensing, computation, and logging workload. Finally, NVM write-endurance is discussed.

TABLE 5.1: Simulation parameters.

Parameter	Value
CPU	Cortex M0+
CPU frequency	1.25 MHz
CPU active current	1.0 µA
Instruction memory size	128 kB (32 banks of 4 kB)
Data memory size	32 kB (8 banks of 4 kB)
Storage capacitor	10 µF
Supply power (P_{supply})	10 µW
Core voltage (V_{core})	1.8 V
On-voltage (V_{on})	2.6 V
Suspend Voltage Threshold (V_{warn})	2.1 V
SRAM read/write	0.33 pJ/b
SRAM leakage (active)	52.9 pA/b
SRAM leakage (retention)	11.2 pA/b
SRAM activate bank	94 pJ

5.3.1 Experimental setup

The evaluation is performed using *Fused* from Chapter 4. Importantly for this evaluation, *Fused* simulates energy and execution in a closed feedback loop, enabling the exploration of the effects of various design choices on the overall performance of a system that is intermittently powered.

The simulation parameters are listed in Table 5.1. A power source that outputs constant power, P_{supply}, is used as a simple model of real energy harvesters, which typically have decreased output current at higher output voltage (and vice versa). In each simulation time step, it adjusts its output current according to load voltage to achieve constant power output. Using a constant power source instead of a specific energy harvester model facilitates analysis (for example, energy can be calculated as the product of time and the power supply setting), and makes the results more readily transferable to a specific setup (energy harvester, capacitor, voltages etc.).

The device turns on when the voltage across the storage capacitor, v_{cap} reaches the on-voltage V_{on}. When v_{cap} discharges below the voltage warning threshold, V_{warn}, an interrupt is issued to trigger a suspend (or abort a FASE). After suspend completes, the device shuts off and charges back up to V_{on}. If v_{cap} drops below V_{core}, the device shuts off regardless of whether a suspend has completed; shut-off without completing suspend results in system failure, and should never occur in a properly configured device.

Dividing memories into banks allows the power gating of inactive banks to reduce leakage power. For example, if an application only uses a fraction of available memory for extended periods of execution, the inactive banks can be powered down. For NVMs

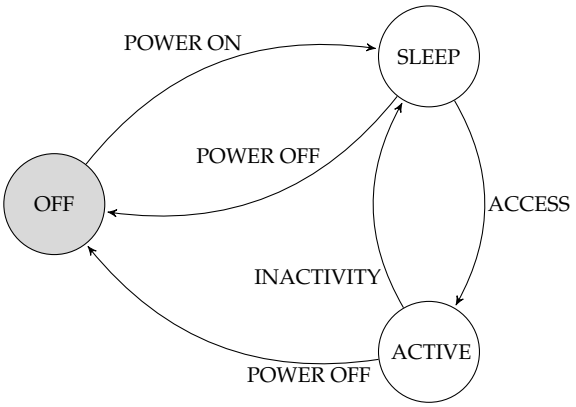


FIGURE 5.4: State machine for power gating inactive memory banks

like MRAM, that could mean powering down a bank completely (as it is non-volatile), or powering down certain (power-hungry) parts of the macro.

To ensure fair evaluation, all SRAM and MRAM memories are divided into banks with automatic retention modes. The caches are not power gated, as they are considered them too small for this method to yield significant benefits.

This is consistent with modern practices for low-power devices. Importantly, it also ensures that our evaluation does not overestimate the leakage power of the baseline methods that employ separate SRAM memories. When comparing a small cache to a large SRAM, the possibility that most of the large SRAM is in fact kept in a low-power retention mode when inactive should not be neglected. The power gating scheme employed for evaluation is shown in Figure 5.4, and is loosely based on prior work [44, 113]. In essence, each bank is individually controlled, and will only become active when accessed, and will again enter a retention mode when an inactivity timer expires. These power mode transitions cost energy, so a trade-off exists between too frequently changing the power modes of the banks, and leaving them on for too long and thus consuming excessive leakage power. In this evaluation, an inactivity timer of 1000 clock cycles, which was found to give a reasonable balance.

The energy consumption of the SRAM and MRAM memories are modeled as read and write energy per accessed bit, leakage current per bit according to operating state (ACTIVE/SLEEP/OFF), and the transition energy between operating modes. The data and tag arrays of the caches are modeled to have the same access energy and leakage current per bit as the SRAM memory in ACTIVE-state. The parameters were obtained using Arm's production MRAM [18] and SRAM [114] compilers, targeting commercial 28 nm FDSOI and 22 nm bulk processes, respectively. While used in our experiments, the specific values of the MRAM parameters are confidential, and are therefore omitted from Table 5.1 (and altered in the public *MEMIC* simulation package). Note however, that since all source code associated with this chapter is released to the public, readers

TABLE 5.2: List of workloads and their code and data footprints.

Workload	Code Size (kB)	Data Size (kB)
bc	3.4	2.05
nn-gru-cmsis	7.12	8.56
matmul	4.46	4.37
fft-q31-cmsis	75.45	2.8
crc	5.17	4.03
aes	5.75	4.04
ar	3.98	2.29
qrencode	13.29	2.08

can readily repeat the presented simulations for the particular parameter values relevant to their target technology. Since this work focuses on the memory system, the CPU power consumption is modeled as a constant current.

5.3.2 Workloads

Table 5.2 lists the workloads used for evaluation, all of which are included in *MEMIC*'s simulation package. These workloads were selected based on being relevant to the domain and on having diverse code and data footprints and memory access patterns. Two of the workloads use the *Arm CMSIS5 library* without modification, showing that *MEMIC* is directly compatible with existing complex code-bases. All workloads except the last are computational workloads which perform computation on data stored in the program binary. The last workload, `nn-gru-cmsis-fase`, which exercises *MEMIC*'s FASE support by sampling sensor data, is described in § 5.3.8.

5.3.3 Baseline

This subsection describes the baseline methods used for evaluation. In regards to instruction memory, *MEMIC* is compared to two baseline configurations, namely *ExecuteInPlace* and *LoadExecute*. *ExecuteInPlace* executes instructions directly from MRAM, which simplifies software and minimizes area. However, the read energy of MRAM is much higher than that of SRAM, so it may not be the most energy efficient solution. *LoadExecute*, on the other hand, executes instructions from a separate instruction-SRAM, using DMA transfers to load instructions from NVM to SRAM during boot. Thus *LoadExecute* decreases read energy significantly. However, *LoadExecute* also increases area and leakage power significantly, as the instruction SRAM must house as many bits as the instruction MRAM. A variation on *LoadExecute* could be to map the most active code to VM and the rest to NVM [70]. However, this requires the developer to statically analyze or profile the code before deployment to determine what to allocate to VM, and thus to make assumptions about how the device will operate in the

future. In the context of IC, execution strongly depends on energy conditions, and so the ideal mapping found during profiling may not match conditions later on.

As the baseline for data memory, *AllocatedState* [9, 129] and *Freezer* [108] are used. Both baselines use a data-SRAM that is loaded from NVM during boot, and checkpointed when power fails. Note that the equivalent of an *ExecuteInPlace* for data is infeasible due to the high write energy of MRAM (in addition to the complexity of rolling back state after a failed FASE[3]). Similarly, methods that map certain portions of variable-sections (stack, `.data`, etc.) to NVM [70] would be ineffective with MRAM, as they increase NVM write-frequency. Increasing the write-frequency on MRAM is highly detrimental to performance as write accesses can require several orders of magnitude more energy than write accesses to SRAM. Note, however, that MRAM is still superior to FRAM, partly because it is compatible with more advanced process nodes, as discussed in § 2.8.

AllocatedState is a software-based IC method based on *Hibernus*[9], that checkpoints and restores all allocated volatile memory (i.e. `.data`, `.bss`, `.stack`) to and from NVM when power respectively fails or recovers. Our implementation of *AllocatedState* transfers data using DMA to improve efficiency. *Freezer* can be described as an optimization of *AllocatedState*, where only modified data is written to NVM when suspending state. It thus provides similar benefits to *ManagedState* (Chapter 3), albeit with hardware support which obviates the need for software annotations and provides better performance. Our implementation of *Freezer* [108] is a memory-mapped peripheral that tracks writes to data-SRAM in order to record which 32 B blocks have been written to. Each block's state is represented by a single *dirty bit* in the write tracker's register file. Software then uses the *dirty bits* to set up DMA transfers of modified blocks when suspending state. When there are multiple contiguous modified blocks, they get aggregated to one large DMA transfer to reduce setup-overhead. Note that all methods evaluated in this chapter push the core CPU registers (`r0-r12`, `lr`, `pc`) to the stack before the stack/data is saved to NVM.

5.3.4 Cache configuration

Caches have various parameters that affect their power, performance and area (PPA), primarily the line width (LW), associativity (AS) and number of sets (NS). The capacity of the cache, is calculated as $LW \cdot AS \cdot NS$.

To find optimal cache parameters, design space exploration was performed using *Fused*. A total of 48 combinations of LW, AS and NS for instruction caches and 79 combinations

[3]FASE support for a system using only non-volatile data memory would require double buffering, which in turn leads to greatly increased number of NVM accesses as data is copied between the buffers.

for data caches of size 1–8 kB were simulated for all the computational workloads in Table 5.2. Each workload was run for a number of iterations such that the total on-time exceeded five seconds, to ensure that each workload required several power cycles to complete. The number of power cycles to complete each workload ranged from 19 to 243 (best configuration on shortest workload to worst configuration on longest workload). For both the instruction and data cache, the cache configuration that yielded the lowest geometric mean completion time across all workloads was chosen for *MEMIC*.

The results are presented for the instruction and data caches in the next two respective subsections. For brevity, the presentation is focused on three distinctive workloads that have very different combinations of instruction and data footprints. The completion times of all workloads are shown later, in Table 5.3 (10 µF columns). Detailed results for all workloads are available in the public dataset.

5.3.5 Instruction Cache

This subsection investigates the relative performance between two baseline methods and different instruction cache configurations (LW, AS and NS). For consistent results, all experiments in this subsection use the baseline *AllocatedState* method for data memory.

Figure 5.5 shows the energy consumption of instruction memory per executed instruction, and the workload completion time, for four different configurations along the horizontal axis. Each plot shows the result for an individual workload. The energy consumption of the SRAM instruction memory (baseline) and the instruction cache are both shown as SRAM (Read/Write/Leakage). The energy consumption of *ExecuteInPlace* comprises NVM read energy and NVM leakage, as no cache or SRAM is used. The relatively high NVM read energy leads to much higher energy consumption, and thus longer charging time than other configurations. *LoadExecute* mitigates the NVM read energy, as each instruction is only read from NVM once (during boot). The drawback with *LoadExecute* is the area and leakage introduced by the SRAM memory, as well as the time and energy taken to load the instructions. For applications with a large instruction footprint, such as the `fft-q31-cmsis` workload, the overall time and energy consumption is dominated by loading instructions, as indicated by the significant increase in on-time and NVM read energy, as well as the increased SRAM leakage due to more SRAM banks being active. Even though only a small portion of the program is executed in each power cycle, *LoadExecute* loads the whole program.

The latter two configurations use an instruction cache in place of the instruction SRAM; the first of them is the best cache configuration for the specific workload, and the second is the best overall configuration (lowest geometric mean completion time across all workloads). In addition to reducing NVM reads compared to *ExecuteInPlace*, the

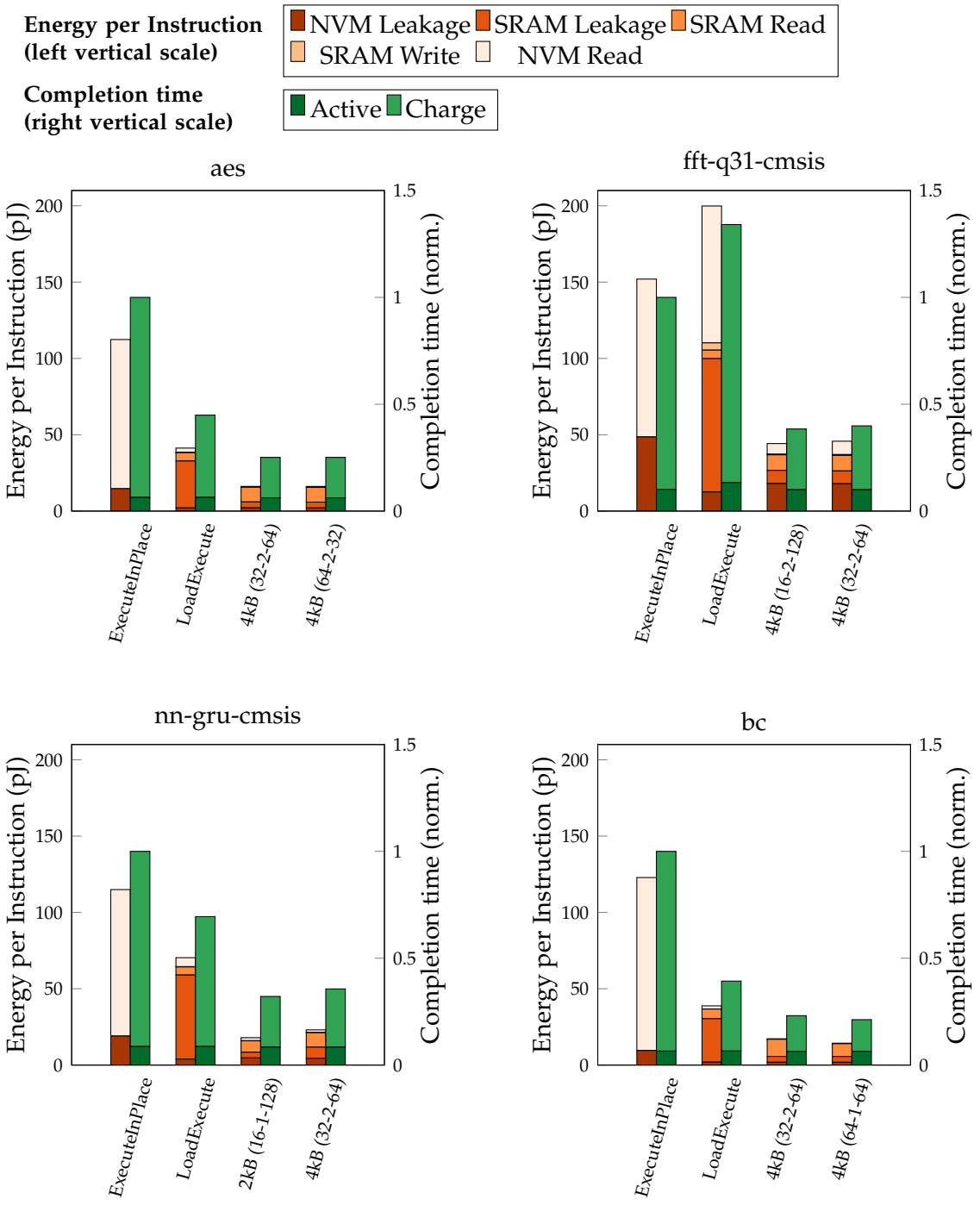


FIGURE 5.5: Instruction access energies per executed instruction (left bars) and workload completion times (right bars, normalized to *ExecuteInPlace*) for different instruction memory architectures. The cache configurations are denoted as "size (Line Width-Associativity-Sets)".

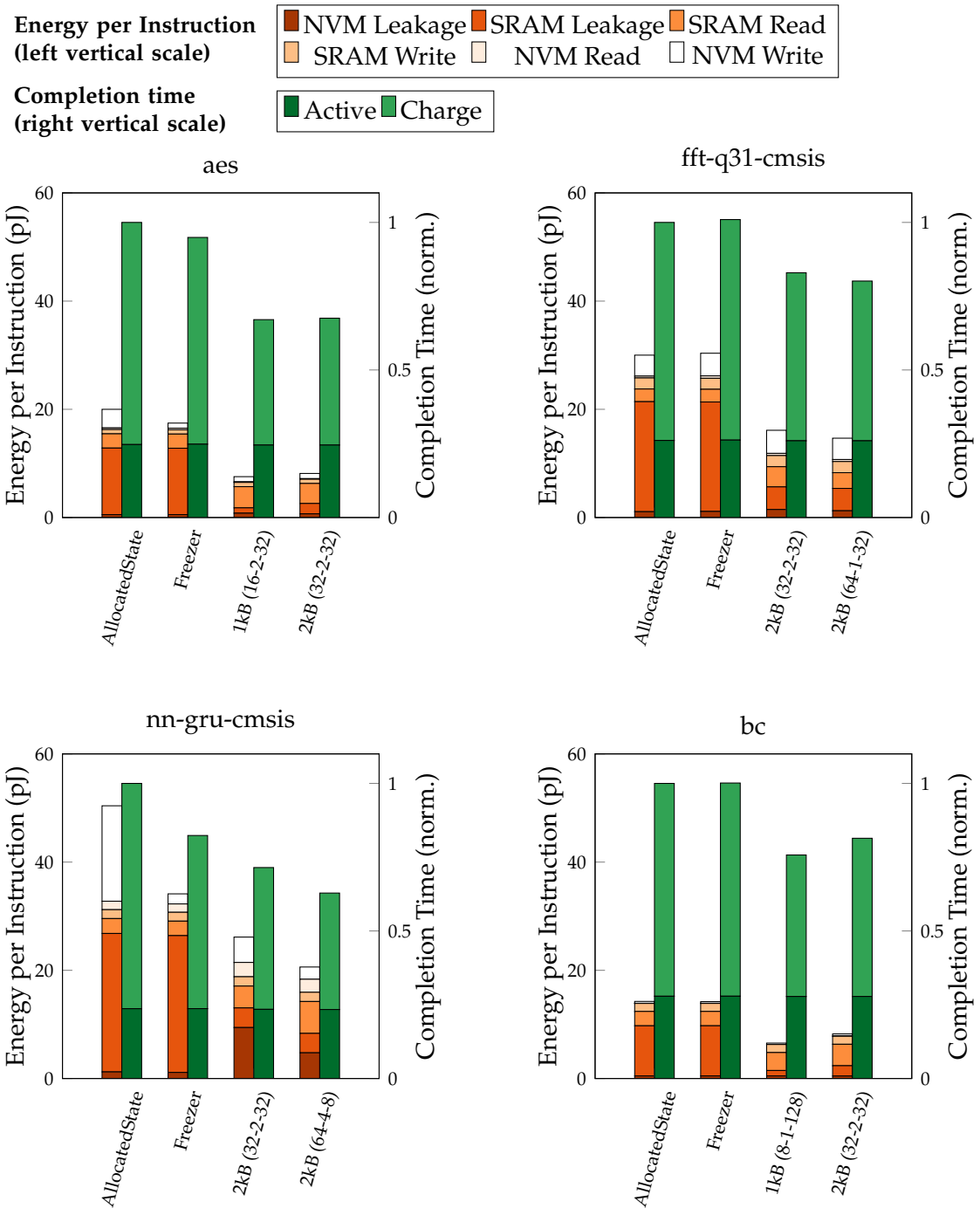


FIGURE 5.6: Data access energies per executed instruction (left bars) and workload completion times (right bars, normalized to *AllocatedState*) for different data memory architectures. The cache configurations are denoted as "size (Line Width-Associativity-Sets)". Note that runtime is dependent on total power consumption, not just the component attributed to data access. Hence a large reduction in data access energy leads to a proportionally smaller reduction in total runtime. All configurations use the 4 kB (16-2-128) instruction cache.

instruction cache solves the instruction loading issue of *LoadExecute*, because the cache mechanism only reads instructions from NVM when they are needed. Additionally, the cached systems reduce area and leakage power compared to *LoadExecute*, because a small cache can cover a large instruction space.

Across all workloads, our results show that a 4 kB instruction cache, arranged as 16 B line width, 2-way set associativity and 128 sets, reduces the workload completion time by 60–77 % (71 % mean) and 41–70 % (49 % mean) compared to the baseline *ExecuteInPlace* and *LoadExecute* configurations, respectively. As these simulations were done using a constant-power supply, these time savings correspond to equal energy-savings. Furthermore, this 4 kB cache uses 1536 b of tag and 32 768 b of data SRAM bit cells, compared to the 1 048 576 b used by the 128 kB SRAM; a reduction of 97 %. Assuming the cache logic has little overhead over the access logic of the large SRAM, this could lead to a substantial area reduction.

5.3.6 Data Cache

In the previous subsection, a 4 kB instruction cache was found to be the best option. This section now investigates the data memory to find the relative performance between *AllocatedState*, *Freezer*, and *MEMIC* configurations using a *data* cache. For fair comparison, and to isolate the effect of data memory, all simulations use the chosen 4 kB instruction cache.

Figure 5.6 shows the energy per instruction for *data* accesses and the total runtime, broken into active and charging, for four configurations along the horizontal axis. Each plot shows the result for an individual workload. The main inefficiencies of *AllocatedState* are caused by the data SRAM, which needs to be large enough to fit all volatile state, and the fact that all volatile state has to be loaded and backed up, regardless of whether it is modified (or even used) during the current on-period. For workloads with a small data footprint and good access locality, such as the `aes` workload, these inefficiencies may be acceptable. However, the NVM write energy (caused by suspend) grows with the data memory footprint, as seen when comparing `aes` and `nn-gru-cmsis` in Figure 5.6. Suspend energy is explored further in § 5.3.7. For workloads with poor access locality or a large active data set (`fft-q31-cmsis` and `nn-gru-cmsis`), leakage energy also grows because more SRAM banks stay active. *Freezer* significantly reduces the average NVM write energy by avoiding back-up of unmodified data for workloads with a significant data footprint where not all data is modified in every power cycle. The worst-case NVM write energy for *Freezer*, however, where all data has been modified, remains the same as for *AllocatedState* (in fact slightly worse due to tracking overhead). And *Freezer* does not improve on the other components of the total energy consumption, such as SRAM leakage. These inefficiencies and inconveniences are alleviated by using a data cache instead of data SRAM.

The data cache has higher energy consumption per access than SRAM, due to cache misses, the overhead of checking tags and, to some degree, loading more data than was requested (i.e. due to the cache line granularity). It can also lead to increased NVM leakage and access energy due to cache misses. The cache leakage energy, however, is much lower than that of the SRAM required by *AllocatedState* and *Freezer*, due to its smaller size.

Across all workloads, the 2 kB data cache, arranged as 32 B line width, 2-way set associativity and 32 sets, reduced the workload completion time by 17–39 % (26 % mean) and 13–39 % (23 % mean) compared to *AllocatedState* and *Freezer*, respectively. This 2 kB data cache uses 320 b of tag and 16 384 b of data SRAM bit cells, compared to the 262 144 b used by the 32 kB data SRAM; a reduction of 94 %.

5.3.7 Operating conditions

This section evaluates the energy required to suspend state and the workload completion times of *MEMIC*, *AllocatedState* and *Freezer* under different operating conditions. Certain systems may have less energy available for suspending state, either because of lower maximum supply voltage (hence also lower V_{warn}), or because the energy buffering capacitance is lower. This evaluation is on capacitor size, but the same analysis can also be solved for voltage. Figure 5.7 shows the suspend energy and minimum required capacitance for three workloads. Two extra configurations are shown in the figure; *MEMIC-MM32* and *MEMIC-MM16* show results for *MEMIC* when the number of modified cache lines is limited to 32 and 16, respectively.

The energy consumed for suspending state, $E_{suspend}$, was calculated from simulation results as follows:

$$E_{suspend} = E_{warn} - E_{done} + E_{supply} \tag{5.1}$$

$$= \tfrac{1}{2}CV_{warn}^2 - \tfrac{1}{2}Cv_{done}^2 + P_{supply}t_{suspend} \tag{5.2}$$

where E_{warn} is the stored energy when the voltage warning is issued, E_{done} is the stored energy when suspend has completed, v_{done} is the capacitor voltage when suspend has completed, $t_{suspend}$ is the time it took to suspend, and C is the energy buffering capacitance.

Based on $E_{suspend}$, the minimum required capacitance, C_{min}, was calculated as follows:

$$E_{avail} = E_{warn} - E_{min} \tag{5.3}$$

$$= \tfrac{1}{2}CV_{warn}^2 - \tfrac{1}{2}CV_{core}^2 \tag{5.4}$$

$$C_{min} = \frac{2E_{suspend}}{V_{warn}^2 - V_{core}^2}, \tag{5.5}$$

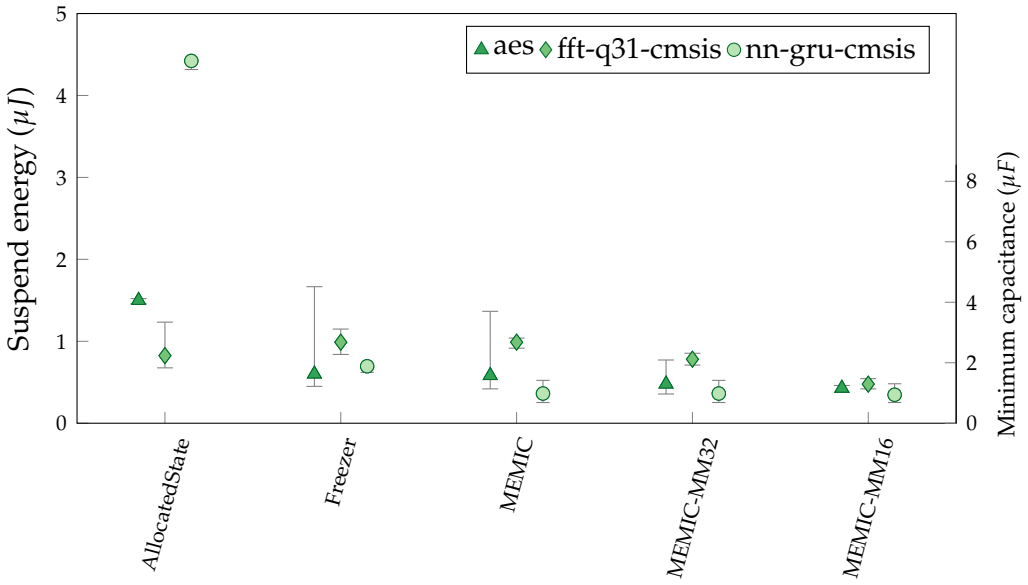


FIGURE 5.7: Energy consumption of suspend (left vertical scale), and the corresponding minimum required capacitance (right vertical scale). The points show the mean value, and the bars show the minimum and maximum values. *MEMIC-MMxx* denotes *MEMIC* with the limit on modified lines (*MODMAX*) set to *xx*. *MEMIC* is the default configuration where *MODMAX* is set to the total number of lines in the cache.

TABLE 5.3: Completion times of all workloads for three capacitor sizes. All configurations use the 4 kB (16-2-128) instruction cache.

	AllocatedState			Freezer			MEMIC		
Workload	2.2 uF	4.7 uF	10 uF	2.2 uF	4.7 uF	10 uF	2.2 uF	4.7 uF	10 uF
qrencode	18.95	16.92	15.93	18.95	16.93	15.95	13.51	12.82	12.19
matmul	fail	10.55	9.43	fail	10.56	9.28	6.28	5.88	5.71
fft-q31-cmsis	0.26	0.21	0.18	0.26	0.21	0.18	0.49	0.17	0.14
crc	fail	5.10	4.51	fail	4.52	4.31	4.97	3.91	3.59
ar	3.84	3.42	3.31	3.74	3.35	3.30	2.87	2.63	2.52
nn-gru-cmsis	fail	fail	1.54	1.76	1.42	1.27	1.24	1.15	1.08
aes	fail	21.26	18.38	fail	18.55	17.44	14.57	13.34	12.47
bc	3.81	3.44	3.51	3.82	3.45	3.51	3.07	2.92	2.86

where E_{avail} is the energy available for suspending and E_{min} is the stored energy when the stored voltage is equal to the minimum operating voltage for the system. Note that the required capacitance is linear with the required energy.

Table 5.3 lists the completion time of all workloads for three capacitor values, comparing *AllocatedState*, *Freezer* and *MEMIC*. All configurations use the 4 kB (16-2-128) instruction cache. *MEMIC*'s *MODMAX* parameter was adjusted according to capacitor size. Empirically, it was found that $MODMAX = 400 \cdot C \cdot 10^6 / LW$ was a reasonable value (i.e. 400 B/µF) in this situation. By comparing Table 5.3 and Figure 5.7, one can see why e.g. `nn-gru-cmsis` fails on *AllocatedState* when the capacitor size is 1.0–4.7 µF, but succeeds at 10 µF: ≈8 µF is the minimum required capacitance. In practice, these results mean that *AllocatedState* needs a larger capacitor and/or a higher suspend voltage threshold to run workloads with a large data footprint. This, in turn,

shows that the application programmer has to re-evaluate the capacitor/suspend voltage threshold whenever the application changes; for a complicated end-device, this dependency between application and electrical properties can substantially complicate development. This also applies for *Freezer*, because, in the worst case where all data has been modified (e.g. a power failure after a long on-period), *Freezer* saves as much data as *AllocatedState*. *MEMIC*, on the other hand, never holds more modified state than is permitted by *MODMAX*, up to a maximum of the size of the data cache, regardless of the application. *MODMAX* enables the end-device to run with a smaller capacitor and/or a lower suspend voltage threshold; both of which can have several other benefits, such as improving the energy efficiency of the energy harvester and reducing conversion loss. As expected, reducing *MODMAX* can result in performance degradation, because the cache's ability to buffer writes is reduced. Compared to *MEMIC*, this performance degradation resulted in less than 2 % increased completion time for most workloads under the *MM32* and *MM16* configurations. For `fft-q31-cmsis`, however, the increase was 66 % and 212 % for *MM32* and *MM16*, respectively. The sharp degradation in `fft-q31-cmsis` is caused by frequent data writes with poor locality.

5.3.8 Case study: Solar-powered sensor node

To evaluate *MEMIC* under realistic conditions, the base simulation model was extended with a PV-cell, a boost regulator, and an accelerometer, as shown in Figure 5.8, and implemented a realistic IoT workload. In each simulation time step, the single-diode PV-cell model [126] takes as input the luminance and the stored voltage v_{in}, to calculate its output current, which charges the 10 µF input capacitor C_{in}. v_{in} is then boosted up to 1.8 V by the boost regulator, with an efficiency of 80 %. The boost regulator is loosely modeled after the *Texas Instruments BQ25570* energy harvesting power management module. When v_{in} exceeds 1.4 V and the boost regulator's OOK (output OK) signal is high, the supply voltage supervisor (SVS) connects the microcontroller and accelerometer to the 1 µF output capacitor C_{out}. Later, when v_{in} discharges below 0.3 V, SVS issues a voltage warning (WARN) which triggers a checkpoint (or FASE abort) in the microcontroller.

The sensing workload, `nn-gru-cmsis-logger`, records a window of 100 three-axis accelerometer values sampled at 1 kHz and uses the sampled data as input to a Gated Recurrent Unit neural network. The accelerometer is a *Fused* model of the *Bosch BMA280*. It communicates over SPI, and implements a 1 kB internal FIFO buffer. Its function and power consumption is modeled based on the device data sheet. The sample window is recorded within a FASE, so that the samples within a window are guaranteed to be continuous. While sampling, the accelerometer buffers data, and the CPU sleeps. When 100 samples (700 B[4]) have been buffered, the accelerometer requests an interrupt

[4]Each sample comprises 6 B of data and a 1 B header.

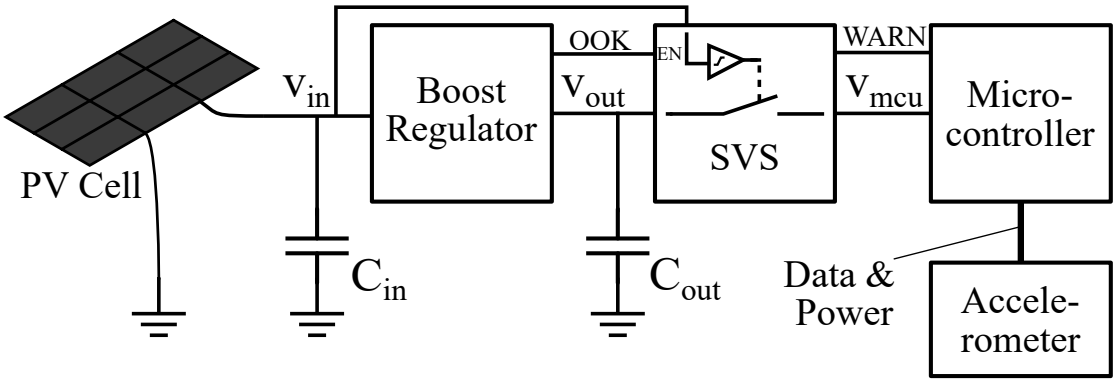


FIGURE 5.8: Simulation model of a solar-powered IC device.

via GPIO, causing the CPU to wake up and read the data into memory; this concludes the FASE. The workload then proceeds to run the neural network intermittently over several power cycles.

Figure 5.9 shows the average completion time of the workload under different lighting conditions (across $20 - 120$ iterations), divided into the time spent charging, actively executing, and sensing while the CPU sleeps. Compared to the baselines, *MEMIC* is able to complete the workload under lower light conditions, and is the only workload to succeed at 800 lux. In the range of 1000–1600 lux, *MEMIC* completes the workload 6–27 % (21 % mean) and 10–31 % (22 % mean) faster than *AllocatedState* and *Freezer*, respectively. At 1600 lux and above, *MEMIC* stops power cycling, as the power input is higher than active power. The same applies for the baseline methods at 1800 lux and above, thus at high light levels all methods complete the workload in approximately the same amount of time (within 1 %).

By further analyzing the simulation data from the PV-cell case study, the overhead and efficacy of the undo-logger under realistic conditions can be evaluated. Across all simulations, up to 0.19 % of all write accesses *during FASE execution* were logged by the undo logger. When not using the `unsafe` zone, this grew to 0.65 %, albeit with insignificant performance overhead (completion time was within 1 %). Despite providing small improvements in this workload, the `unsafe` zone is an easy optimization to use, and can be important in particular workloads that write a large amount of data (approaching the data cache size) inside a FASE.

5.3.9 Endurance

Current MRAMs are limited in endurance, i.e. may fail after a specified number of writes to the same bit-cells. To assess whether *MEMIC* has an impact on NVM endurance, *Fused*'s memory model was instrumented to record the maximum total number of writes to a single location (byte) in data-MRAM. On average, *MEMIC* had 2 %

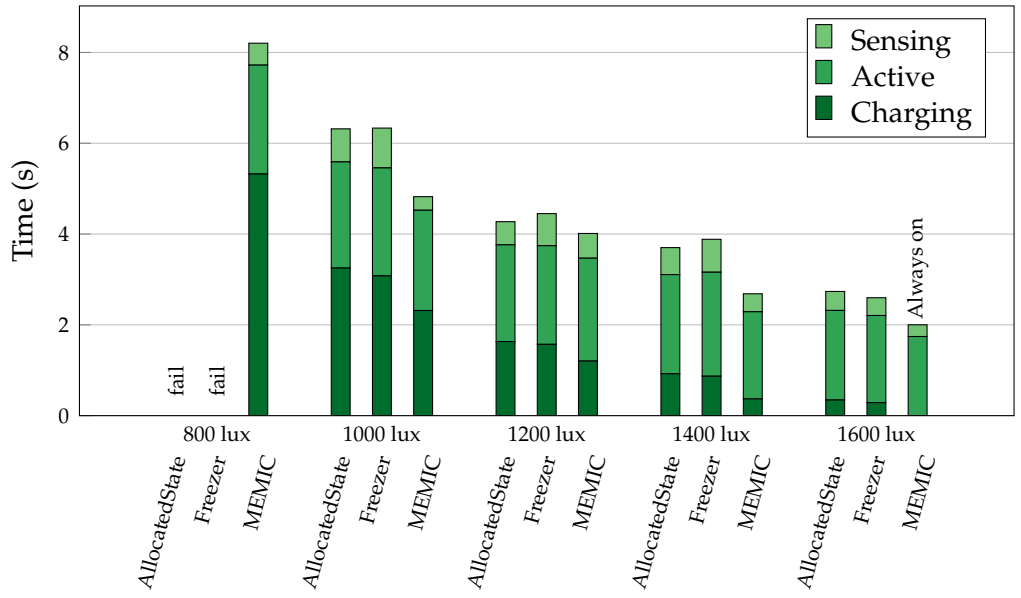


FIGURE 5.9: Average completion time of the `nn-gru-cmsis-logger` workload on the solar-powered IC device (Figure 5.8) under different lighting conditions. The two baseline configurations use the same instruction cache as *MEMIC*.

more writes to the same location per completion of each workload, and thus does not significantly impact write endurance.

5.4 Discussion

Addressing **Q2** by leveraging *Fused*, the simulator proposed in Chapter 4, this chapter proposed a memory architecture for IC with hardware support to optimize the core operations of IC without compromising programmability or robustness. Table 5.4 summarizes the evaluation of *MEMIC* against the correctness criteria, performance goals, and scaling goals from § 2.4. The following paragraphs discusses *MEMIC* in detail.

Instruction caching substantially improves energy efficiency and performance under intermittent operation: partially by reducing access and leakage energy, but also by avoiding unnecessary loading of the entire program during boot, when energy is scarce and on-periods are short. By simulating 48 instruction cache configurations on eight workloads running intermittently, it was found that a 4 kB instruction cache arranged as 16 B line width, 2-way set associativity and 128 sets, reduces workload completion time by 60–77 % (71 % mean) and 41–70 % (49 % mean) compared to the baseline *ExecuteInPlace* and *LoadExecute* configurations, respectively.

By similarly simulating 79 data cache configurations, it was found that a 2 kB data cache arranged as 32 B line width, 2-way set associativity and 32 sets, provided a further reduction of completion time by 17–39 % (26 % mean) and 13–39 % (23 % mean) compared to the relevant baselines *AllocatedState* and *Freezer*, respectively.

TABLE 5.4: *MEMIC* evaluated on the correctness criteria, performance goals, and scalability goals from § 2.4.

Objective/Requirement	MEMIC
C1: Comp. progress	✓
C2: Consistent memory	✓
C3: FASE support	✓
C4: Avoid sisyph. tasks	✓[1]
P1: Forward progress	Fast
P2: Reactive to events	✓[1]
P3: Memory overhead	Checkpoint, undo-log
S1: Required IC expertise	Minor
S2: Software compatibility	✓
S3: Hardware portable	✓

[1]As long as user-created atomic sections do not require more energy than the device can muster in one on-period.

As these simulations were carried out using a constant-power supply, these time savings correspond to equal energy savings. Both caches also present a substantial decrease in area, as they reduce the number of SRAM bit cells by over 90 % as compared to the SRAMs needed by the baseline methods.

Naively using a data cache while operating intermittently, however, would cause idempotency violations and other bugs related to re-execution, because write-backs from the cache could corrupt NVM. *MEMIC* solves this by using a hardware undo-logger that is activated during failure-atomic sections. Our results show that cache write-backs are rare during failure-atomic sections because the cache is flushed immediately before executing them, so the undo-logger is unlikely to need large capacity.

Experiments running the eight workloads with three different capacitor sizes showed that the baseline methods failed to complete some workloads when the capacitor size was smaller than 10 µF, because their operation is conditional on the application's memory usage. In contrast, *MEMIC* successfully ran all workloads for the tested capacitor sizes by using *MODMAX* to adapt to operating conditions. Similarly, when modelling a solar powered IC device, running a realistic logging workload, *MEMIC* was able to complete the workload under lower light conditions, and with better performance (6–31 %) across light conditions ranging from 1000 lux to 1600 lux.

Chapter 6

Conclusions and future work

6.1 Conclusions

Supporting IC on COTS hardware (**Q1**) is important to lower the barrier to entry. Through a systematic review of IC literature, in Chapter 2, reactive IC was found to be the most promising method because of its performance and compatibility with existing software. However, a significant inefficiency in current methods was found. These existing approaches save and restore all allocated state in every power cycle, even though a majority of data can be left unchanged, or even unused, in short on-periods. To alleviate this shortcoming, Chapter 3 presented *ManagedState*, a page-based memory management software layer that provided substantial (26.8–86.9 %) reduction in suspend and restore time for reactive IC on a COTS microcontroller. This, in conjunction with its dynamic suspend and restore voltage thresholds, led to up to 5.3 × faster execution of the benchmarks tested, when on an intermittent supply with short on-periods. However, *ManagedState* requires program annotations to indicate used and modified, and tracking memory usage in software introduces overheads. A better way to gain the benefits of *ManagedState* could be to use hardware support that tracks use and modification without requiring annotations.

Before researching memory architectures and hardware support for IC (**Q2**), however, an appropriate modeling methodology was needed; the thesis therefore addressed **Q3** before **Q2**. Existing modeling tools are inadequate for modeling IC because they lack flexibility (RTL), accuracy (*gem5*), and closed-loop power-performance simulation (both). To address this gap, i.e. **Q3**, *Fused*, a full-system simulator for energy-driven computers was presented in Chapter 4. Its focus is on accuracy and closed-loop power-performance simulation as well as flexibility to explore new designs to improve IC, and energy-driven computers in general. Experimental results, comparing *Fused* to real hardware, showed that *Fused* can execute real application binaries correctly on a simulated intermittent supply. Across a set of 61 workloads, *Fused* modeled the COTS

microcontroller used in Chapter 3 with a geometric mean error of 3.4 % in power estimation and a maximum execution time error of 1.16 %. More importantly for this thesis, which focuses on IC, *Fused* was able to model a state-of-the art IC method (*ManagedState* from Chapter 3) running a workload intermittently on real hardware. Across an input current limit of 0.2–2 mA, *Fused*'s simulated completion time of the workload was within 6.8 % of the value measured experimentally on real hardware. *Fused* thus closes a gap in the modeling and development tools available for IC: prior works have provided methods for power source emulation [47, 58], PCB-level hardware prototyping [61] and online debugging [27], now *Fused* introduces virtual prototyping to enable efficient hardware-software co-design and better introspection. It should be noted though, that *Fused* was developed and tested with the aim of being used to research **Q2**, so if *Fused* is to be used for research beyond this, it's functional and power models must be reviewed and likely extended to support such research. For example, *Fused* was evaluated at constant (room) temperature – studies related to thermal effects would have to expand on *Fused*s models to correctly model the effect of temperature changes.

Using *Fused* for modeling and evaluation, Chapter 5 proposed novel memory system support for IC to address **Q2**. Instruction caching substantially improves energy efficiency and performance under intermittent operation: partially by reducing access and leakage energy, but also by avoiding unnecessarily loading the entire program during boot, when energy is scarce and on-periods are short. By simulating 48 instruction cache configurations on eight workloads running intermittently, it was found that a 4 kB instruction cache arranged as 16 B line width, 2-way set associativity and 128 sets, reduces workload completion time by 60–77 % (71 % mean) and 41–70 % (49 % mean) compared to the baseline *ExecuteInPlace* and *LoadExecute* configurations, respectively.

By similarly simulating 79 data cache configurations, it was found that a 2 kB data cache arranged as 32 B line width, 2-way set associativity and 32 sets, provided a further reduction of completion time by 17–39 % (26 % mean) and 13–39 % (23 % mean) compared to the relevant baselines *AllocatedState* and *Freezer*, respectively.

As these simulations were carried out using a constant-power supply, these time savings correspond to equal energy savings. Each of the caches also present a substantial decrease in area, as they reduce the number of SRAM bit cells by over 90 % as compared to the SRAMs needed by the state-of-the-art baseline methods.

Experiments running the eight workloads with three different capacitor sizes showed that the baseline methods failed to complete some workloads when the capacitor size was smaller than 10 µF, because their operation is conditional on the application's usage of code and data memory. As a consequence, the programmer must carefully consider electrical components when modifying software, or over-provision energy storage and voltage thresholds to account for the worst-case scenario. In contrast, *MEMIC* can adapt to operating conditions by using its software-configurable limit on modified

volatile state, *MODMAX*, and thus successfully ran all workloads for the tested capacitor sizes.

Similarly, when modelling a solar powered IC device, running a realistic logging workload, *MEMIC* was able to complete the workload under lower light conditions, and with better performance (6–31 %) across light conditions ranging from 1000 lux to 1600 lux.

Naively using a data cache while operating intermittently, however, would cause idempotency violations and other bugs related to re-execution, because write-backs from the cache could corrupt NVM. *MEMIC* solves this by using a hardware undo-logger that is activated during failure-atomic sections. The results showed that cache write-backs are rare during failure-atomic sections because the cache is flushed immediately before executing them, so the undo-logger does not need large buffering capacity. The evaluation of *MEMIC* used a set of workloads chosen to be diverse in memory access patters. It can, however, not be excluded that workloads exist for which *MEMIC*s performance is degraded; hardware architects that consider implementing *MEMIC* should therefore evaluate *MEMIC* on their specific use cases first; for example by running their workloads on the publicly available *MEMIC* implementation in *Fused*.

6.2 Future work

There are several interesting and important areas of research in which the work presented herein can be utilized and extended.

The *Fused* simulator proposed in Chapter 4 can be used as a tool for further IC research in many directions not pursued herein. Researchers exploring energy harvesting, power conditioning and energy management could use it to evaluate the effect of improved energy harvesters and associated power conversion, energy storage and energy provisioning strategies. For example, *Fused* could be used to explore better ways of predicting or profiling energy usage and then provision energy to specific FASEs (or tasks in task-based IC) and thus reduce energy wasted on re-executing aborted FASEs.

Researchers working on wireless communication could use *Fused* to enhance their simulation frameworks with better simulation of the application and processing side of each node in a wireless sensor network. Aspects such as the processing cost of their routing protocols, on top of the application and sensing tasks of the node, could be simulated in conjunction with a network simulator such as *OMNet++* [147]. Alternatively, *Fused* could be used in a profiling step to obtain parameters for simulation of wireless sensor networks.

Similarly, researchers working on energy harvesting, power management, power conversion, and more, could use *Fused* to obtain tangible results for how their proposed designs affect the end-application's performance under different specified conditions.

In Chapter 5, the *MEMIC* architecture was proposed. It targeted, and was evaluated on, single-threaded reactive IC applications. However, *MEMIC* accelerates several functions that are fundamental to most IC methods. Functions like saving only modified data when saving state, data versioning (undo-logging in the case of *MEMIC*) and providing guarantees for successful state saving, could be useful to many other proposed IC methods which have until now only been implemented using software and compiler techniques. For example, task-based IC methods rely on executing the application as a series of back-to-back FASEs, and could thus benefit greatly from hardware-implemented undo-logging.

Together with the *MEMIC* architecture and hardware support, researchers could also use *Fused* to evaluate novel applications of IC for IoT and monitoring applications.

Bibliography

[1] H. Aantjes, A. Y. Majid, P. Pawełczak, J. Tan, A. Parks, and J. R. Smith. "Fast Downstream to Many (Computational) RFIDs". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. IEEE INFOCOM 2017 - IEEE Conference on Computer Communications. May 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8056987.

[2] E. Ahmed, I. Yaqoob, A. Gani, M. Imran, and M. Guizani. "Internet-of-Things-Based Smart Environments: State of the Art, Taxonomy, and Open Research Challenges". In: *IEEE Wireless Communications* 23.5 (Oct. 2016), pp. 10–16. ISSN: 1536-1284. DOI: 10.1109/MWC.2016.7721736.

[3] F. Ahmed, C. Kervadec, Y. Le Moullec, G. Tamberg, and P. Annus. "Autonomous Wireless Sensor Networks: Implementation of Transient Computing and Energy Prediction for Improved Node Performance and Link Quality". In: *The Computer Journal* 62.6 (June 1, 2019), pp. 820–837. ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/bxy101.

[4] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. "Efficient intermittent computing with differential checkpointing". In: *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 2019, pp. 70–81.

[5] Ambiq. *Apollo4*. Ambiq. Sept. 11, 2020. URL: https://ambiq.com/apollo4/ (visited on 08/13/2021).

[6] F. Arnaud, P. Zuliani, J. Reynard, A. Gandolfo, F. Disegni, P. Mattavelli, E. Gomiero, G. Samanni, C. Jahan, R. Berthelon, O. Weber, E. Richard, V. Barral, A. Villaret, S. Kohler, J. Grenier, R. Ranica, C. Gallon, A. Souhaite, and P. Cappelletti. "Truly Innovative 28nm FDSOI Technology for Automotive Microcontroller Applications Embedding 16MB Phase Change Memory". Conference. IEDM Conference. Dec. 2018. URL: https://tinyurl.com/2pysthmh (visited on 08/16/2021).

[7] L. Atzori, A. Iera, and G. Morabito. "The Internet of Things: A Survey". In: *Computer Networks* 54.15 (Oct. 28, 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010.

[8] 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". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.12 (2016), pp. 1968–1980. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2016.2547919.

[9] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems". In: *IEEE Embedded Systems Letters* 7.1 (Mar. 2015), pp. 15–18. ISSN: 1943-0663, 1943-0671. DOI: 10.1109/LES.2014.2371494.

[10] G. Berthou, K. Marquet, T. Risset, and G. Salagnac. "MPU-Based Incremental Checkpointing for Transiently-Powered Systems". In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 2020 23rd Euromicro Conference on Digital System Design (DSD). Aug. 2020, pp. 89–96. DOI: 10.1109/DSD51259.2020.00025.

[11] G. Berthou, P.-É. Dagand, D. Demange, R. Oudin, and T. Risset. "Intermittent Computing with Peripherals, Formally Verified". In: *The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES '20: 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems. London United Kingdom: International Foundation for Autonomous Agents and Multiagent Systems, June 16, 2020, pp. 85–96. ISBN: 978-1-4503-7094-3. DOI: 10.1145/3372799.3394365.

[12] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac. "Sytare: A Lightweight Kernel for NVRAM-Based Transiently-Powered Systems". In: *IEEE Transactions on Computers* 68.9 (Sept. 2019), pp. 1390–1403. ISSN: 2326-3814. DOI: 10.1109/TC.2018.2889080.

[13] G. Berthou, K. Marquet, T. Risset, and G. Salagnac. "Accurate Power Consumption Evaluation for Peripherals in Ultra Low-Power Embedded Systems". In: *2020 Global Internet of Things Summit (GIoTS)*. 2020 Global Internet of Things Summit (GIoTS). June 2020, pp. 1–6. DOI: 10.1109/GIOTS49054.2020.9119593.

[14] 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)*. 2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN). Apr. 2017, pp. 209–220.

[15] N. A. Bhatti and L. Mottola. "Efficient State Retention for Transiently-Powered Embedded Sensing". In: *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks* (Graz, Austria). EWSN '16. USA: Junction Publishing, 2016, pp. 137–148. ISBN: 978-0-9949886-0-7. URL: http://dl.acm.org/citation.cfm?id=2893711.2893731 (visited on 02/07/2019).

[16] S. W. C. Bing, D. Balsamo, and G. V. Merrett. "An Energy-Driven Wireless Bicycle Trip Counter with Zero Energy Storage". In: *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (Shenzhen, China). SenSys '18. New York, NY, USA: ACM, 2018, pp. 404–405. ISBN: 978-1-4503-5952-8. DOI: 10.1145/3274783.3275205.

[17] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. "The Gem5 Simulator". In: *ACM SIGARCH Computer Architecture News* 39.2 (May 31, 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718.

[18] E. M. Boujamaa, S. M. Ali, S. N. Wandji, A. Gourio, S. Pyo, G. Koh, Y. Song, T. Song, J. Kye, J. C. Vial, A. Sowden, M. Rathor, and C. Dray. "A 14.7Mb/Mm ^{2} 28nm FDSOI STT-MRAM with Current Starved Read Path, 52Ω/Sigma Offset Voltage Sense Amplifier and Fully Trimmable CTAT Reference". In: *2020 IEEE Symposium on VLSI Circuits*. 2020 IEEE Symposium on VLSI Circuits. Honolulu, HI, USA: IEEE, June 2020, pp. 1–2. ISBN: 978-1-72819-942-9. DOI: 10.1109/VLSICircuits18222.2020.9162803.

[19] A. Branco, L. Mottola, M. H. Alizai, and J. H. Siddiqui. "Intermittent Asynchronous Peripheral Operations". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. SenSys '19. New York, New York: Association for Computing Machinery, 2019, pp. 55–67. ISBN: 9781450369503. DOI: 10.1145/3356250.3360033.

[20] J. S. Broadhead and P. Pawełczak. "Data Freshness in Mixed-Memory Intermittently-Powered Systems". In: *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2021, pp. 3361–3366.

[21] M. Buettner, B. Greenstein, and D. Wetherall. "Dewdrop: An Energy-Aware Runtime for Computational RFID". In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA). NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 197–210. URL: http://dl.acm.org/citation.cfm?id=1972457.1972478 (visited on 11/28/2019).

[22] O. Cetinkaya and G. V. Merrett. "Efficient deployment of UAV-powered sensors for optimal coverage and connectivity". In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2020, pp. 1–6.

[23] C. Chakrabarti and D. Gaitonde. "Instruction Level Power Model of Microcontrollers". In: *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1999 IEEE International Symposium on Circuits and Systems (ISCAS). Vol. 1. May 1999, 76–79 vol.1. DOI: 10.1109/ISCAS.1999.777809.

[24] A. Chen. "A Review of Emerging Non-Volatile Memory (NVM) Technologies and Applications". In: *Solid-State Electronics*. Extended Papers Selected from ESSDERC 2015 125 (Nov. 1, 2016), pp. 25–38. ISSN: 0038-1101. DOI: 10.1016/j.sse.2016.07.006.

[25] W.-M. Chen, P.-C. Hsiu, and T.-W. Kuo. "Enabling Failure-Resilient Intermittently-Powered Systems Without Runtime Checkpointing". In: *Proceedings of the 56th Annual Design Automation Conference 2019 on - DAC '19*. The 56th Annual Design Automation Conference 2019. Las Vegas, NV, USA: ACM Press, 2019, pp. 1–6. ISBN: 978-1-4503-6725-7. DOI: 10.1145/3316781.3317816.

[26] J. Choi, H. Joe, Y. Kim, and C. Jung. "Achieving Stagnation-Free Intermittent Computation with Boundary-Free Adaptive Execution". In: *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). Apr. 2019, pp. 331–344. DOI: 10.1109/RTAS.2019.00035.

[27] A. Colin, G. Harvey, A. P. Sample, and B. Lucia. "An Energy-Aware Debugger for Intermittently Powered Systems". In: *IEEE Micro* 37.3 (2017), pp. 116–125. DOI: 10.1109/MM.2017.48.

[28] A. Colin and B. Lucia. "Chain: Tasks and Channels for Reliable Intermittent Programs". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA 2016*. The 2016 ACM SIGPLAN International Conference. Amsterdam, Netherlands: ACM Press, 2016, pp. 514–530. ISBN: 978-1-4503-4444-9. DOI: 10.1145/2983990.2983995.

[29] A. Colin and B. Lucia. "Termination Checking and Task Decomposition for Task-Based Intermittent Programs". In: *Proceedings of the 27th International Conference on Compiler Construction - CC 2018*. The 27th International Conference. Vienna, Austria: ACM Press, 2018, pp. 116–127. ISBN: 978-1-4503-5644-2. DOI: 10.1145/3178372.3179525.

[30] A. Colin, E. Ruppel, and B. 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* (Williamsburg, VA, USA). ASPLOS '18. New York, NY, USA: ACM, 2018, pp. 767–781. ISBN: 978-1-4503-4911-6. DOI: 10.1145/3173162.3173210.

[31] E. Çürük, K. S. Yıldırım, P. Pawelczak, and J. Hester. "On the Accuracy of Network Synchronization Using Persistent Hourglass Clocks". In: *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems* (New York, NY, USA). ENSsys'19. New York, NY, USA: ACM, 2019, pp. 35–41. ISBN: 978-1-4503-7010-3. DOI: 10.1145/3362053.3363497.

[32] T. Daulby, A. Savanth, G. V. Merrett, and A. S. Weddell. "Improving the Forward Progress of Transient Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.3 (Mar. 2021), pp. 444–452. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.2999913.

[33] R. Dekimpe, P. Xu, M. Schramme, P. Gérard, D. Flandre, and D. Bol. "A Battery-Less BLE Smart Sensor for Room Occupancy Tracking Supplied by 2.45-GHz Wireless Power Transfer". In: *Integration* 67 (July 1, 2019), pp. 8–18. ISSN: 0167-9260. DOI: 10.1016/j.vlsi.2019.03.006.

[34] C. Delgado, J. M. Sanz, and J. Famaey. "On the Feasibility of Battery-Less LoRaWAN Communications Using Energy Harvesting". In: *Proceedings of the 2019 IEEE Global Communications Conference*. IEEE Global Communications Conference. Waikoloa, HI, USA: IEEE, 2019, p. 7.

[35] H. Desai, M. Nardello, D. Brunelli, and B. Lucia. "Camaroptera: A Long-Range Image Sensor with Local Inference for Remote Sensing Applications". In: *ACM Trans. Embed. Comput. Syst.* 21.3 (May 2022). ISSN: 1539-9087. DOI: 10.1145/3510850.

[36] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.7 (July 2012), pp. 994–1007. ISSN: 0278-0070. DOI: 10.1109/TCAD.2012.2185930.

[37] C. D. Donne, K. S. Yildirim, A. Y. Majid, J. Hester, and P. l. Pawe\ lczak. "Backing out of Backscatter for Intermittent Wireless Networks". In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems* (Shenzhen, China). ENSsys '18. New York, NY, USA: ACM, 2018, pp. 38–40. ISBN: 978-1-4503-6047-0. DOI: 10.1145/3279755.3279758.

[38] C. Durmaz, K. S. Yildirim, and G. Kardas. "PureMEM: A Structured Programming Model for Transiently Powered Computers". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (Limassol, Cyprus). SAC '19. New York, NY, USA: ACM, 2019, pp. 1544–1551. ISBN: 978-1-4503-5933-7. DOI: 10.1145/3297280.3299739.

[39] Economist. *The Internet of Things Business Index 2017*. The Economist Intelligence Unit. 2017. URL: https://eiuperspectives.economist.com/sites/default/files/EIU-ARM-IBM%20IoT%20Business%20Index%202017%20copy.pdf (visited on 01/30/2019).

[40] A. El Kouche, L. Al-Awami, and H. Hassanein. "Dynamically Reconfigurable Energy Aware Modular Software (DREAMS) Architecture for WSNs in Industrial Environments". In: *Procedia Computer Science*. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011)

5 (Jan. 1, 2011), pp. 264–271. ISSN: 1877-0509. DOI: 10.1016/j.procs.2011.07.035.

[41] J. Eriksson, A. Dunkels, and N. Finne. "Poster Abstract: MSPsim – an Extensible Simulator for MSP430-Equipped Sensor Boards". In: *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo Session*. European Conference on Wireless Sensor Networks (EWSN). Delft, Netherlands, 2007, p. 2.

[42] Fang Su, Zhibo Wang, Jinyang Li, M.-F. Chang, and Y. Liu. "Design of Nonvolatile Processors and Applications". In: *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC). Tallinn, Estonia: IEEE, Sept. 2016, pp. 1–6. ISBN: 978-1-5090-3561-8. DOI: 10.1109/VLSI-SoC.2016.7753543.

[43] Faycal Ait Aouda, K. Marquet, and G. Salagnac. "Incremental Checkpointing of Program State to NVRAM for Transiently-Powered Systems". In: *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. 2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). Montpellier, France: IEEE, May 2014, pp. 1–4. ISBN: 978-1-4799-5810-8. DOI: 10.1109/ReCoSoC.2014.6861359.

[44] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. "Drowsy Caches: Simple Techniques for Reducing Leakage Power". In: *Proceedings of the Annual International Symposium on Computer Architecture, ISCA* (2002), pp. 148–157. ISSN: 1063-6897. DOI: 10.1109/ISCA.2002.1003572.

[45] M. Furlong, J. Hester, K. Storer, and J. Sorber. "Realistic Simulation for Tiny Batteryless Sensors". In: *Proceedings of the 4th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems - ENSsys'16*. The 4th International Workshop. Stanford, CA, USA: ACM Press, 2016, pp. 23–26. ISBN: 978-1-4503-4532-3. DOI: 10.1145/2996884.2996889.

[46] K. Ganesan, J. San Miguel, and N. Enright Jerger. "The What's Next Intermittent Computing Architecture". In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA). Feb. 2019, pp. 211–223. DOI: 10.1109/HPCA.2019.00039.

[47] K. Geissdoerfer, M. Chwalisz, and M. Zimmerling. "Shepherd: A portable testbed for the batteryless iot". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 2019, pp. 83–95.

[48] K. Geissdoerfer and M. Zimmerling. "Bootstrapping Battery-free Wireless Networks: Efficient Neighbor Discovery and Synchronization in the Face of Intermittency". In: *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 439–455. ISBN: 978-1-939133-21-2. URL: https://www.usenix.org/conference/nsdi21/presentation/geissdoerfer.

[49] Z. Ghodsi, S. Garg, and R. Karri. "Optimal Checkpointing for Secure Intermittently-Powered IoT Devices". In: *Proceedings of the 36th International Conference on Computer-Aided Design* (Irvine, California). ICCAD '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 376–383. URL: http://dl.acm.org/citation.cfm?id=3199700.3199750 (visited on 03/27/2019).

[50] Globalfoundries. *Embedded Memory: eMRAM, eFlash, eSIP, Product Brief*. 2018. URL: https://globalfoundries.com/sites/default/files/product-briefs/pb-emem-14-web.pdf.

[51] G. Gobieski, B. Lucia, and N. Beckmann. "Intelligence beyond the edge: Inference on intermittent embedded systems". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 199–213.

[52] A. Gomez, L. Sigrist, M. Magno, L. Benini, and L. Thiele. "Dynamic Energy Burst Scaling for Transiently Powered Systems". In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2016 Design, Automation Test in Europe Conference Exhibition (DATE). Mar. 2016, pp. 349–354.

[53] A. Gomez, L. Sigrist, T. Schalch, L. Benini, and L. Thiele. "Efficient, Long-Term Logging of Rich Data Sensors Using Transient Sensor Nodes". In: *ACM Transactions on Embedded Computing Systems* 17.1 (Sept. 20, 2017), pp. 1–23. ISSN: 15399087. DOI: 10.1145/3047499.

[54] P. A. Hager, H. Fatemi, J. P. de Gyvez, and L. Benini. "A Scan-Chain Based State Retention Methodology for IoT Processors Operating on Intermittent Energy". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. Design, Automation Test in Europe Conference Exhibition (DATE), 2017. Mar. 2017, pp. 1171–1176. DOI: 10.23919/DATE.2017.7927166.

[55] J. Han and M. Orshansky. "Approximate Computing: An Emerging Paradigm for Energy-Efficient Design". In: *2013 18th IEEE European Test Symposium (ETS)*. 2013 18th IEEE European Test Symposium (ETS). May 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

[56] M. Herlihy and E. B. Moss. "Transactional Memory: Architectural Support For Lock-Free Data Structures". In: *Proceedings of the 20th Annual International Symposium on Computer Architecture*. Proceedings of the 20th Annual International Symposium on Computer Architecture. May 1993, pp. 289–300. DOI: 10.1109/ISCA.1993.698569.

[57] J. Hester, T. Jia, and J. Gu. "Holistic Energy Management with μProcessor Co-Optimization in Fully Integrated Battery-Less IoTs". In: *2018 31st IEEE International System-on-Chip Conference (SOCC)*. 2018 31st IEEE International System-on-Chip Conference (SOCC). Sept. 2018, pp. 7–12. DOI: 10.1109/SOCC.2018.8618523.

[58] J. Hester, T. Scott, and J. 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*. The 12th ACM Conference. Memphis, Tennessee: ACM Press, 2014, pp. 1–15. ISBN: 978-1-4503-3143-2. DOI: 10.1145/2668332.2668336.

[59] J. Hester, L. Sitanayah, T. Scott, and J. Sorber. "Realistic and Repeatable Emulation of Energy Harvesting Environments". In: *ACM Transactions on Sensor Networks* 13.2 (Apr. 24, 2017), pp. 1–33. ISSN: 15504859. DOI: 10.1145/3064839.

[60] J. Hester, L. Sitanayah, and J. Sorber. "Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors". In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems - SenSys '15*. The 13th ACM Conference. Seoul, South Korea: ACM Press, 2015, pp. 5–16. ISBN: 978-1-4503-3631-4. DOI: 10.1145/2809695.2809707.

[61] J. Hester and J. Sorber. "Flicker: Rapid Prototyping for the Batteryless Internet-of-Things". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems - SenSys '17*. The 15th ACM Conference. Delft, Netherlands: ACM Press, 2017, pp. 1–13. ISBN: 978-1-4503-5459-2. DOI: 10.1145/3131672.3131674.

[62] J. Hester and J. Sorber. "The Future of Sensing Is Batteryless, Intermittent, and Awesome". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems - SenSys '17*. The 15th ACM Conference. Delft, Netherlands: ACM Press, 2017, pp. 1–6. ISBN: 978-1-4503-5459-2. DOI: 10.1145/3131672.3131699.

[63] J. Hester, K. Storer, and J. Sorber. "Timely Execution on Intermittently Powered Batteryless Sensors". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (Delft, Netherlands). SenSys '17. New York, NY, USA: ACM, 2017, 17:1–17:13. ISBN: 978-1-4503-5459-2. DOI: 10.1145/3131672.3131673.

[64] J. Hester, N. Tobias, A. Rahmati, L. Sitanayah, D. Holcomb, K. Fu, W. P. Burleson, and J. Sorber. "Persistent Clocks for Batteryless Sensing Devices". In: *ACM Transactions on Embedded Computing Systems* 15.4 (Aug. 2, 2016), pp. 1–28. ISSN: 15399087. DOI: 10.1145/2903140.

[65] M. Hicks. "Clank: Architectural Support for Intermittent Computation". In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). June 2017, pp. 228–240. DOI: 10.1145/3079856.3080238.

[66] T. Instruments. *MSP430FR599x , MSP430FR596x Mixed-Signal Microcontrollers Datasheet*. 2017.

[67] B. Islam, Y. Luo, and S. Nirjon. *Zygarde: Time-Sensitive On-Device Deep Intelligence on Intermittently-Powered Systems*. May 4, 2019. arXiv: 1905.03854 [cs, eess]. URL: http://arxiv.org/abs/1905.03854 (visited on 05/23/2019).

[68] H. Jayakumar, A. Raha, and V. Raghunathan. "QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers". In: *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems. Jan. 2014, pp. 330–335. DOI: 10.1109/VLSID.2014.63.

[69] H. Jayakumar, A. Raha, and V. Raghunathan. "Energy-Aware Memory Mapping for Hybrid FRAM-SRAM MCUs in IoT Edge Devices". In: *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID). Kolkata, India: IEEE, Jan. 2016, pp. 264–269. ISBN: 978-1-4673-8700-2. DOI: 10.1109/VLSID.2016.52.

[70] H. Jayakumar, A. Raha, J. R. Stevens, and V. Raghunathan. "Energy-Aware Memory Mapping for Hybrid FRAM-SRAM MCUs in Intermittently-Powered IoT Devices". In: *ACM Trans. Embed. Comput. Syst.* 16.3 (Apr. 2017), 65:1–65:23. ISSN: 1539-9087. DOI: 10.1145/2983628.

[71] C.-K. Kang, C.-H. Lin, P.-C. Hsiu, and M.-S. Chen. "HomeRun: HW/SW Co-Design for Program Atomicity on Self-Powered Intermittent Systems". In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ISLPED '18. Seattle, WA, USA: Association for Computing Machinery, 2018. ISBN: 9781450357043. DOI: 10.1145/3218603.3218633.

[72] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. "Power Management in Energy Harvesting Sensor Networks". In: *ACM Transactions on Embedded Computing Systems* 6.4 (Sept. 1, 2007), 32–es. ISSN: 15399087. DOI: 10.1145/1274858.1274870.

[73] M. Karimi, H. Choi, Y. Wang, Y. Xiang, and H. Kim. "Real-Time Task Scheduling on Intermittently Powered Batteryless Devices". In: *IEEE Internet of Things Journal* 8.17 (2021), pp. 13328–13342. DOI: 10.1109/JIOT.2021.3065947.

[74] S. Khanna, M. Zwerg, B. Elies, J. Luebbe, N. Krishnasawamy, H. Najar, S. Bellary, W.-Y. Shih, S. Summerfelt, and S. Bartling. "17.4 16MHz FRAM Micro-Controller with a Low-Cost Sub-1μA Embedded Piezo-Electric Strain Sensor for ULP Motion Detection". In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2019 IEEE International Solid- State Circuits Conference - (ISSCC). Feb. 2019, pp. 282–284. DOI: 10.1109/ISSCC.2019.8662391.

[75] M. Kim, J. Lee, Y. Kim, and Y. H. Song. "An Analysis of Energy Consumption under Various Memory Mappings for FRAM-Based IoT Devices". In: *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). Singapore: IEEE, Feb. 2018, pp. 574–579. ISBN: 978-1-4673-9944-9. DOI: 10.1109/WF-IoT.2018.8355212.

[76] H. Koike, T. Ohsawa, S. Ikeda, T. Hanyu, H. Ohno, T. Endoh, N. Sakimura, R. Nebashi, Y. Tsuji, A. Morioka, S. Miura, H. Honjo, and T. Sugibayashi. "A Power-Gated MPU with 3-Microsecond Entry/Exit Delay Using MTJ-Based Nonvolatile Flip-Flop". In: *2013 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. 2013 IEEE Asian Solid-State Circuits Conference (A-SSCC). Singapore, Singapore: IEEE, Nov. 2013, pp. 317–320. ISBN: 978-1-4799-0280-4 978-1-4799-0277-4. DOI: 10.1109/ASSCC.2013.6691046.

[77] V. Kortbeek, K. S. Yildirim, A. Bakar, J. Sorber, J. Hester, and P. Pawełczak. "Time-Sensitive Intermittent Computing Meets Legacy Software". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, Mar. 9, 2020, pp. 85–99. ISBN: 978-1-4503-7102-5. DOI: 10.1145/3373376.3378476.

[78] A. S. Krishnan, C. Suslowicz, D. Dinu, and P. Schaumont. "Secure Intermittent Computing Protocol: Protecting State Across Power Loss". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019*. 2019, pp. 734–739. DOI: 10.23919/DATE.2019.8714997.

[79] S. Lan, Z. Wang, J. Mamish, J. Hester, and Q. Zhu. "AdaSens: Adaptive Environment Monitoring by Coordinating Intermittently-Powered Sensors". In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2022, pp. 556–561. DOI: 10.1109/ASP-DAC52403.2022.9712501.

[80] A. Lee, C.-P. Lo, C.-C. Lin, W.-H. Chen, K.-H. Hsu, Z. Wang, F. Su, Z. Yuan, Q. Wei, Y.-C. King, C.-J. Lin, H. Lee, P. Khalili Amiri, K.-L. Wang, Y. Wang, H. Yang, Y. Liu, and M.-F. Chang. "A ReRAM-Based Nonvolatile Flip-Flop With Self-Write-Termination Scheme for Frequent-OFF Fast-Wake-Up Nonvolatile Processors". In: *IEEE Journal of Solid-State Circuits* 52.8 (Aug. 2017), pp. 2194–2207. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.2017.2700788.

[81] S. Lee, B. Islam, Y. Luo, and S. Nirjon. "Intermittent learning: On-device machine learning on intermittently powered system". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3.4 (2019), pp. 1–30.

[82] S. Lee and S. Nirjon. "Neuro.ZERO: A Zero-Energy Neural Network Accelerator for Embedded Sensing and Inference Systems". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (New York, New York). SenSys '19. New York, NY, USA: ACM, 2019, pp. 138–152. ISBN: 978-1-4503-6950-3. DOI: 10.1145/3356250.3360030.

[83] Q. Li, M. Zhao, J. Hu, Y. Liu, Y. He, and C. J. Xue. "Compiler Directed Automatic Stack Trimming for Efficient Non-Volatile Processors". In: *Proceedings of the 52Nd Annual Design Automation Conference* (San Francisco, California). DAC '15. New York, NY, USA: ACM, 2015, 183:1–183:6. ISBN: 978-1-4503-3520-1. DOI: 10.1145/2744769.2744809.

[84] W. S. Lim, C.-H. Tu, C.-F. Wu, and Y.-H. Chang. "iCheck: Progressive Checkpointing for Intermittent Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020), pp. 1–1. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.3046571.

[85] Y.-C. Lin, P.-C. Hsiu, and T.-W. Kuo. "Autonomous I/O for Intermittent IoT Systems". In: *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). July 2019, pp. 1–6. DOI: 10.1109/ISLPED.2019.8824923.

[86] Q. Liu and C. Jung. "Lightweight Hardware Support for Transparent Consistency-Aware Checkpointing in Intermittent Energy-Harvesting Systems". In: *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. 2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA). Aug. 2016, pp. 1–6. DOI: 10.1109/NVMSA.2016.7547183.

[87] Y. Liu, F. Su, Y. Yang, Z. Wang, Y. Wang, Z. Li, X. Li, R. Yoshimura, T. Naiki, T. Tsuwa, T. Saito, Z. Wang, K. Taniuchi, and H. Yang. "A 130-Nm Ferroelectric Nonvolatile System-on-Chip With Direct Peripheral Restore Architecture for Transient Computing System". In: *IEEE Journal of Solid-State Circuits* (2019), pp. 1–11. ISSN: 0018-9200. DOI: 10.1109/JSSC.2018.2884349.

[88] Y. Liu, Z. Wang, A. Lee, F. Su, C.-P. Lo, Z. Yuan, C.-C. Lin, Q. Wei, Y. Wang, Y.-C. King, C.-J. Lin, P. Khalili, K.-L. Wang, M.-F. Chang, and H. Yang. "A 65nm ReRAM-Enabled Nonvolatile Processor with 6x Reduction in Restore Time and 4x Higher Clock Frequency Using Adaptive Data Retention and Self-Write-Termination Nonvolatile Logic". In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 2016 IEEE International Solid-State Circuits Conference (ISSCC). San Francisco, CA, USA: IEEE, Jan. 2016, pp. 84–86. ISBN: 978-1-4673-9466-6 978-1-4673-9467-3. DOI: 10.1109/ISSCC.2016.7417918.

[89] Y. Liu, Y. Xie, J. Shu, H. Yang, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, and S. John. "Ambient Energy Harvesting Nonvolatile Processors: From Circuit to System". In: *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*. The 52nd Annual Design Automation Conference. San Francisco, California: ACM Press, 2015, pp. 1–6. ISBN: 978-1-4503-3520-1. DOI: 10.1145/2744769.2747910.

[90] Y. Liu, J. Yue, H. Li, Q. Zhao, M. Zhao, C. J. Xue, G. Sun, M.-F. Chang, and H. Yang. "Data Backup Optimization for Nonvolatile SRAM in Energy Harvesting Sensor Nodes". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.10 (Oct. 2017), pp. 1660–1673. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2017.2648841.

[91] B. Lucia, B. Denby, Z. Manchester, H. Desai, E. Ruppel, and A. Colin. "Computational Nanosatellite Constellations: Opportunities and Challenges". In: *GetMobile: Mobile Comp. and Comm.* 25.1 (June 2021), pp. 16–23. ISSN: 2375-0529. DOI: 10.1145/3471440.3471446.

[92] B. Lucia and B. Ransford. "A Simpler, Safer Programming and Execution Model for Intermittent Systems". In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA). PLDI '15. New York, NY, USA: ACM, 2015, pp. 575–585. ISBN: 978-1-4503-3468-6. DOI: 10.1145/2737924.2737978.

[93] G. Lukosevicius, A. R. Arreola, and A. S. Weddell. "Using Sleep States to Maximize the Active Time of Transient Computing Systems". In: *Proceedings of the Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems* (Delft, Netherlands). ENSsys'17. New York, NY, USA: ACM, 2017, pp. 31–36. ISBN: 978-1-4503-5477-6. DOI: 10.1145/3142992.3142998.

[94] K. Ma, X. Li, J. Li, Y. Liu, Y. Xie, J. Sampson, M. T. Kandemir, and V. Narayanan. "Incidental Computing on IoT Nonvolatile Processors". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-50 '17*. The 50th Annual IEEE/ACM International Symposium. Cambridge, Massachusetts: ACM Press, 2017, pp. 204–218. ISBN: 978-1-4503-4952-9. DOI: 10.1145/3123939.3124533.

[95] 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: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). Burlingame, CA, USA: IEEE, Feb. 2015, pp. 526–537. ISBN: 978-1-4799-8930-0. DOI: 10.1109/HPCA.2015.7056060.

[96] K. Maeng, A. Colin, and B. Lucia. "Alpaca: Intermittent Execution Without Checkpoints". In: *Proc. ACM Program. Lang.* 1 (OOPSLA Oct. 2017), 96:1–96:30. ISSN: 2475-1421. DOI: 10.1145/3133920.

[97] K. Maeng and B. Lucia. "Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing". In: 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). 2018, pp. 129–144. URL: https://www.usenix.org/conference/osdi18/presentation/maeng (visited on 03/03/2019).

[98] K. Maeng and B. Lucia. "Supporting Peripherals in Intermittent Systems with Just-in-Time Checkpoints". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA). PLDI 2019. New York, NY, USA: ACM, 2019, pp. 1101–1116. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314613.

[99] K. Maeng and B. 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. London, UK: Association for Computing Machinery, June 11, 2020, pp. 1005–1021. ISBN: 978-1-4503-7613-6. DOI: 10.1145/3385412.3385998.

[100] A. Maioli, L. Mottola, M. H. Alizai, and J. H. Siddiqui. "On Intermittence Bugs in the Battery-Less Internet of Things (WIP Paper)". In: *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems - LCTES 2019*. The 20th ACM SIGPLAN/SIGBED International Conference. Phoenix, AZ, USA: ACM Press, 2019, pp. 203–207. ISBN: 978-1-4503-6724-0. DOI: 10.1145/3316482.3326346.

[101] A. Y. Majid, M. Jansen, G. O. Delgado, K. S. Yıldırım, and P. Pawełczak. *Multi-Hop Backscatter Tag-to-Tag Networks*. Jan. 29, 2019. arXiv: 1901.10274 [cs]. URL: http://arxiv.org/abs/1901.10274 (visited on 02/03/2019).

[102] H. R. Mendis and P.-C. Hsiu. "Accumulative Display Updating for Intermittent Systems". In: *ACM Trans. Embed. Comput. Syst.* 18 (5s Oct. 2019), 72:1–72:22. ISSN: 1539-9087. DOI: 10.1145/3358190.

[103] G. V. Merrett and B. M. Al-Hashimi. "Energy-Driven Computing: Rethinking the Design of Energy Harvesting Systems". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, May 5, 2017. URL: https://eprints.soton.ac.uk/404363/.

[104] Mimi Xie, Chen Pan, Jingtong Hu, C. Yang, and Y. Chen. "Checkpoint-Aware Instruction Scheduling for Nonvolatile Processor with Multiple Functional Units". In: *The 20th Asia and South Pacific Design Automation Conference*. 2015 20th Asia and South Pacific Design Automation Conference (ASP-DAC). Chiba, Japan: IEEE, Jan. 2015, pp. 316–321. ISBN: 978-1-4799-7792-5. DOI: 10.1109/ASPDAC.2015.7059024.

[105] T. Moreau, J. San Miguel, M. Wyse, J. Bornholt, A. Alaghi, L. Ceze, N. Enright Jerger, and A. Sampson. "A Taxonomy of General Purpose Approximate Computing Techniques". In: *IEEE Embedded Systems Letters* 10.1 (Mar. 2018), pp. 2–5. ISSN: 1943-0663. DOI: 10.1109/LES.2017.2758679.

[106] M. Nardello, H. Desai, D. Brunelli, and B. Lucia. "Camaroptera: a batteryless long-range remote visual sensing system". In: *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems* (New York, NY, USA). ENSsys'19. New York, NY, USA: ACM, 2019, pp. 8–14. ISBN: 978-1-4503-7010-3. DOI: 10.1145/3362053.3363491.

[107] M. Natsui, D. Suzuki, A. Tamakoshi, T. Watanabe, H. Honjo, H. Koike, T. Nasuno, Y. Ma, T. Tanigawa, Y. Noguchi, M. Yasuhira, H. Sato, S. Ikeda, H. Ohno, T. Endoh, and T. Hanyu. "12.1 An FPGA-Accelerated Fully Nonvolatile Microcontroller Unit for Sensor-Node Applications in 40nm CMOS/MTJ-Hybrid Technology Achieving 47.14μW Operation at 200MHz". In: *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2019 IEEE International Solid- State Circuits Conference - (ISSCC). Feb. 2019, pp. 202–204. DOI: 10.1109/ISSCC.2019.8662431.

[108] D. Pala, I. Miro-Panades, and O. Sentieys. "Freezer: A Specialized NVM Backup Controller for Intermittently-Powered Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020), pp. 1–1. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.3025063.

[109] J. Pallister, S. Hollis, and J. Bennett. *BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms*. Aug. 23, 2013. arXiv: 1308.5174 [cs]. URL: http://arxiv.org/abs/1308.5174 (visited on 01/25/2019).

[110] C. Pan, M. Xie, S. Han, Z.-H. Mao, and J. Hu. "Modeling and Optimization for Self-Powered Non-Volatile IoT Edge Devices with Ultra-Low Harvesting Power". In: *ACM Trans. Cyber-Phys. Syst.* 3.3 (Aug. 2019), 32:1–32:26. ISSN: 2378-962X. DOI: 10.1145/3324609.

[111] D. Patterson and J. Hennessy. *Computer Organization and Design ARM Edition*. 1st. Morgan Kaufmann, 2016. 720 pp. ISBN: 978-0-12-801733-3. URL: https://www.elsevier.com/books/computer-organization-and-design-arm-edition/patterson/978-0-12-801733-3.

[112] S. S. Philip, R. Passerone, K. S. Yildirim, and D. Brunelli. "Intermittent Computing Emulation of Ultra-Low-Power Processors: Evaluation of Backup Strategies for RISC-V". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022). DOI: 10.1109/TCAD.2022.3169108.

[113] P. Prabhat, B. Labbe, G. Knight, A. Savanth, J. Svedas, M. J. Walker, S. Jeloka, P. M.-Y. Fan, F. Garcia-Redondo, T. Achuthan, and J. Myers. "27.2 M0N0: A Performance-Regulated 0.8-to-38MHz DVFS ARM Cortex-M33 SIMD MCU with 10nW Sleep Power". In: *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*. 2020 IEEE International Solid- State Circuits Conference - (ISSCC). Feb. 2020, pp. 422–424. DOI: 10.1109/ISSCC19947.2020.9063136.

[114] G. Rangarajan. *Arm Delivers a Comprehensive Physical IP Platform for Optimized SoCs with TSMC 22nm ULP/ULL Process Technology*. Arm Delivers a Comprehensive Physical IP Platform for Optimized SoCs with TSMC 22nm ULP/ULL Process Technology. Oct. 2018. URL: https://community.arm.com/developer/ip-products/system/b/soc-design-blog/posts/arm-delivers-comprehensive-physical-ip-platform-for-optimized-socs-tsmc-22nm-ulp-ull.

[115] B. Ransford and B. Lucia. "Nonvolatile Memory Is a Broken Time Machine". In: *Proceedings of the Workshop on Memory Systems Performance and Correctness* (Edinburgh, United Kingdom). MSPC '14. New York, NY, USA: ACM, 2014, 5:1–5:3. ISBN: 978-1-4503-2917-0. DOI: 10.1145/2618128.2618136.

[116] B. Ransford, J. Sorber, and K. Fu. "Mementos: System Support for Long-Running Computation on RFID-Scale Devices". In: *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVI. New York, NY, USA: ACM, 2011, pp. 159–170. ISBN: 978-1-4503-0266-1. DOI: 10.1145/1950365.1950386.

[117] C. C. Rheinländer and N. Wehn. "Adaptive Transient Computing for Power-Neutral Embedded Devices". In: *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). July 2019, pp. 63–68. DOI: 10.1109/PATMOS.2019.8862063.

[118] A. Rodriguez, D. Balsamo, Z. Luo, S. P. Beeby, G. V. Merrett, and A. S. Weddell. "Intermittently-Powered Energy Harvesting Step Counter for Fitness Tracking". In: *2017 IEEE Sensors Applications Symposium (SAS)*. 2017 IEEE Sensors Applications Symposium (SAS). Mar. 2017, pp. 1–6. DOI: 10.1109/SAS.2017.7894114.

[119] A. Rodriguez Arreola, D. Balsamo, A. K. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, and G. V. Merrett. "Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation". In: *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems* (Seoul, South Korea). ENSsys '15. New York, NY, USA: ACM, 2015, pp. 3–8. ISBN: 978-1-4503-3837-0. DOI: 10.1145/2820645.2820652.

[120] A. Rodriguez Arreola, D. Balsamo, G. Merrett, and A. Weddell. "RESTOP: Retaining External Peripheral State in Intermittently-Powered Sensor Systems". In: *Sensors* 18.2 (Jan. 10, 2018), p. 172. ISSN: 1424-8220. DOI: 10.3390/s18010172.

[121] S. Ruffini, L. Caronti, K. S. Yildirim, and D. Brunelli. "NORM: An FPGA-based Non-volatile Memory Emulation Framework for Intermittent Computing". In: *ACM JOURNAL ON EMERGING TECHNOLOGIES IN COMPUTING SYSTEMS* 1 (2022) (2022), pp. 1–19. DOI: 10.1145/3517812.

[122] E. Ruppel and B. Lucia. "Transactional Concurrency Control for Intermittent, Energy-Harvesting Computing Systems". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA). PLDI 2019. New York, NY, USA: ACM, 2019, pp. 1085–1100. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314583.

[123] N. Sakimura, Y. Tsuji, R. Nebashi, H. Honjo, A. Morioka, K. Ishihara, K. Kinoshita, S. Fukami, S. Miura, N. Kasai, T. Endoh, H. Ohno, T. Hanyu, and T. Sugibayashi. "A 90nm 20MHz Fully Nonvolatile Microcontroller for Standby-Power-Critical Applications". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014 IEEE International Solid- State Circuits Conference (ISSCC). San Francisco, CA, USA: IEEE, Feb. 2014, pp. 184–185. ISBN: 978-1-4799-0920-9 978-1-4799-0918-6. DOI: 10.1109/ISSCC.2014.6757392.

[124] D. Samyde, S. Skorobogatov, R. Anderson, and J. Quisquater. "On a New Way to Read Data from Memory". In: *First International IEEE Security in Storage Workshop, 2002. Proceedings.* First International IEEE Security in Storage Workshop, 2002. Proceedings. Dec. 2002, pp. 65–69. DOI: 10.1109/SISW.2002.1183512.

[125] J. San Miguel, K. Ganesan, M. Badr, and N. E. Jerger. "The EH Model: Analytical Exploration of Energy-Harvesting Architectures". In: *IEEE Computer Architecture Letters* 17.1 (Jan. 1, 2018), pp. 76–79. ISSN: 1556-6056. DOI: 10.1109/LCA.2017.2777834.

[126] A. Savanth, A. Weddell, J. Myers, D. Flynn, and B. Al-Hashimi. "Photovoltaic Cells for Micro-Scale Wireless Sensor Nodes: Measurement and Modeling to Assist System Design". In: *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems - ENSsys '15*. The 3rd International Workshop. Seoul, South Korea: ACM Press, 2015, pp. 15–20. ISBN: 978-1-4503-3837-0. DOI: 10.1145/2820645.2820653.

[127] R. L. Shinmoto Torres, R. Visvanathan, S. Hoskins, A. Van den Hengel, and D. C. Ranasinghe. "Effectiveness of a Batteryless and Wireless Wearable Sensor System for Identifying Bed and Chair Exits in Healthy Older People". In: *Sensors* 16.4 (Apr. 2016), p. 546. DOI: 10.3390/s16040546.

[128] P. Singla, S. S. Singh, and S. R. Sarangi. "Flexicheck: An adaptive checkpointing architecture for energy harvesting devices". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 546–551.

[129] S. T. Sliper, D. Balsamo, N. Nikoleris, W. Wang, A. S. Weddell, and G. V. Merrett. "Efficient State Retention Through Paged Memory Management for Reactive Transient Computing". In: *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA). DAC '19. New York, NY, USA: ACM, 2019, 26:1–26:6. ISBN: 978-1-4503-6725-7. DOI: 10.1145/3316781.3317812.

[130] S. T. Sliper, D. Balsamo, A. S. Weddell, and G. V. Merrett. "Enabling Intermittent Computing on High-Performance Out-of-Order Processors". In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ENSsys '18. New York, NY, USA: ACM, 2018, pp. 19–25. ISBN: 978-1-4503-6047-0. DOI: 10.1145/3279755.3279759.

[131] S. T. Sliper, O. Cetinkaya, A. S. Weddell, B. Al-Hashimi, and G. V. Merrett. "Energy-Driven Computing". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2164 (Feb. 7, 2020), p. 20190158. DOI: 10.1098/rsta.2019.0158.

[132] S. T. Sliper, W. Wang, N. Nikoleris, A. S. Weddell, and G. V. Merrett. "Fused: Closed-Loop Performance and Energy Simulation of Embedded Systems". In: *Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'20). Boston, MA, USA: IEEE, Apr. 2020.

[133] S. T. Sliper, W. Wang, N. Nikoleris, A. S. Weddell, A. Savanth, P. Prabhat, and G. V. Merrett. "Pragmatic Memory-System Support for Intermittent Computing using Emerging Non-Volatile Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022). DOI: 10.1109/TCAD.2022.3168263.

[134] *Spin-Transfer Torque MRAM Technology — Everspin*. URL: https://www.everspin.com/spin-transfer-torque-mram-technology (visited on 08/13/2021).

[135] F. Su, Y. Liu, Y. Wang, and H. Yang. "A Ferroelectric Nonvolatile Processor with 46 *M*s System-Level Wake-up Time and 14 *M*s Sleep Time for Energy Harvesting Applications". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.3 (Mar. 2017), pp. 596–607. ISSN: 1549-8328. DOI: 10.1109/TCSI.2016.2616905.

[136] F. Su, Y. Liu, X. Sheng, H. G. Lee, N. Chang, and H. Yang. "A Task Failure Rate Aware Dual-Channel Solar Power System for Nonvolatile Sensor Nodes". In: *ACM Trans. Embed. Comput. Syst.* 18.4 (July 2019), 33:1–33:21. ISSN: 1539-9087. DOI: 10.1145/3320270.

[137] M. Surbatovich, L. Jia, and B. Lucia. "I/O Dependent Idempotence Bugs in Intermittent Systems". In: *Proc. ACM Program. Lang.* 3 (OOPSLA Oct. 2019), 183:1–183:31. ISSN: 2475-1421. DOI: 10.1145/3360609.

[138] M. Surbatovich, L. Jia, and B. Lucia. "Automatically Enforcing Fresh and Consistent Inputs in Intermittent Systems". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 851–866. ISBN: 9781450383912. URL: https://doi.org/10.1145/3453483.3454081.

[139] M. Surbatovich, B. Lucia, and L. Jia. "Towards a Formal Foundation of Intermittent Computing". In: *Proc. ACM Program. Lang.* 4.OOPSLA (Nov. 2020). DOI: 10.1145/3428231.

[140] C. Suslowicz, A. S. Krishnan, D. Dinu, and P. Schaumont. "Secure Application Continuity in Intermittent Systems". In: *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*. 2018 Ninth International Green and Sustainable Computing Conference (IGSC). Oct. 2018, pp. 1–8. DOI: 10.1109/IGCC.2018.8752145.

[141] J. Tan, P. Pawełczak, A. Parks, and J. R. Smith. "Wisent: Robust Downstream Communication and Storage for Computational RFIDs". In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524574.

[142] A. Teverovsky. "Insulation Resistance and Leakage Currents in Low-Voltage Ceramic Capacitors With Cracks". In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 4.7 (July 2014), pp. 1169–1176. ISSN: 2156-3985. DOI: 10.1109/TCPMT.2014.2318178.

[143] P. Thanigai and W. Goh. *Application Report: MSP430 FRAM Quality and Reliability*. SLAA526A. Texas Instruments, 2014.

[144] A. Torrisi, K. S. Yildirim, and D. Brunelli. "Reliable Transiently-Powered Communication". In: *IEEE Sensors Journal* 22.9 (2022), pp. 9124–9134. DOI: 10.1109/JSEN.2022.3158736.

[145] E. Valea, M. D. Silva, M. Flottes, G. D. Natale, S. Dupuis, and B. Rouzeyre. "Providing Confidentiality and Integrity in Ultra Low Power IoT Devices". In: *2019 14th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*. 2019 14th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS). Apr. 2019, pp. 1–6. DOI: 10.1109/DTIS.2019.8735090.

[146] J. Van Der Woude and M. Hicks. "Intermittent Computation Without Hardware Support or Programmer Intervention". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA). OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 17–32. ISBN: 978-1-931971-33-1. URL: http://dl.acm.org/citation.cfm?id=3026877.3026880 (visited on 03/03/2019).

[147] A. Varga. "The omnet++ discrete event simulation system". In: *In ESM'01*. 2001. URL: https://www.omnetpp.org.

[148] T. D. Verykios, D. Balsamo, and G. V. Merrett. "Selective Policies for Efficient State Retention in Transiently-Powered Embedded Systems: Exploiting Properties of NVM Technologies". In: *Sustainable Computing: Informatics and Systems* (July 25, 2018). ISSN: 2210-5379. DOI: 10.1016/j.suscom.2018.07.003.

[149] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi. "Hardware-Validated CPU Performance and Energy Modelling". In: *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Belfast: IEEE, Apr. 2018, pp. 44–53. ISBN: 978-1-5386-5010-3. DOI: 10.1109/ISPASS.2018.00013.

[150] D. Wang, K. Tsui, and Q. Miao. "Prognostics and Health Management: A Review of Vibration Based Bearing and Gear Health Indicators". In: *IEEE Access* 6 (2018), pp. 665–676. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2774261.

[151] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang. "A 3us Wake-up Time Nonvolatile Processor Based on Ferroelectric Flip-Flops". In: *2012 Proceedings of the ESSCIRC (ESSCIRC)*. ESSCIRC 2012 - 38th European Solid State Circuits Conference. Bordeaux, France: IEEE, Sept. 2012, pp. 149–152. ISBN: 978-1-4673-2213-3 978-1-4673-2212-6 978-1-4673-2211-9. DOI: 10.1109/ESSCIRC.2012.6341281.

[152] Z. Wang, Y. Liu, A. Lee, F. Su, C.-P. Lo, Z. Yuan, J. Li, C.-C. Lin, W.-H. Chen, H.-Y. Chiu, W.-E. Lin, Y.-C. King, C.-J. Lin, P. Khalili Amiri, K.-L. Wang, M.-F. Chang, and H. Yang. "A 65-Nm ReRAM-Enabled Nonvolatile Processor With Time-Space Domain Adaption and Self-Write-Termination Achieving 4x Faster Clock Frequency and $>$ 6x Higher Restore Speed". In: *IEEE Journal of Solid-State Circuits* 52.10 (Oct. 2017), pp. 2769–2785. ISSN: 0018-9200, 1558-173X. DOI: 10.1109/JSSC.2017.2724024.

[153] H. Williams, X. Jian, and M. 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*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 69–84. ISBN: 9781450371025. URL: https://doi.org/10.1145/3373376.3378478.

[154] H. Williams, M. Moukarzel, and M. Hicks. "Failure sentinels: ubiquitous just-in-time intermittent computation via low-cost hardware support for voltage monitoring". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 665–678.

[155] J. de Winkel, V. Kortbeek, J. Hester, and P. Pawełczak. "Battery-Free Game Boy". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4.3 (Sept. 2020). DOI: 10.1145/3411839.

[156] S. C. Wong, S. T. Sliper, W. Wang, A. S. Weddell, S. Gauthier, and G. V. Merrett. "Energy-Aware HW/SW Co-Modeling of Batteryless Wireless Sensor Nodes". In: *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. ENSsys '20. New York, NY, USA: Association for Computing Machinery, Nov. 16, 2020, pp. 57–63. ISBN: 978-1-4503-8129-1. DOI: 10.1145/3417308.3430272.

[157] T. Wu, L. Zhang, H. Yang, and Y. Liu. "An Extensible System Simulator for Intermittently-Powered Multiple-Peripheral IoT Devices". In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems* (Shenzhen, China). ENSsys '18. New York, NY, USA: ACM, 2018, pp. 1–6. ISBN: 978-1-4503-6047-0. DOI: 10.1145/3279755.3279756.

[158] Y. Wu, Z. Wang, Z. Jia, Y. Shi, and J. Hu. "Intermittent Inference with Nonuniformly Compressed Multi-Exit Neural Network for Energy Harvesting Powered Devices". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020 57th ACM/IEEE Design Automation Conference (DAC). July 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218526.

[159] M. Xie, C. Pan, Y. Zhang, J. Hu, Y. Liu, and C. J. Xue. "A Novel STT-RAM-Based Hybrid Cache for Intermittently Powered Processors in IoT Devices". In: *IEEE Micro* 39.1 (Jan. 2019), pp. 24–32. ISSN: 0272-1732. DOI: 10.1109/MM.2018.2890257.

[160] M. Xie, M. Zhao, C. Pan, H. Li, Y. Liu, Y. Zhang, C. J. Xue, and J. Hu. "Checkpoint Aware Hybrid Cache Architecture for NV Processor in Energy Harvesting Powered Systems". In: *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). Oct. 2016, pp. 1–10.

[161] Y. Xu, H. G. Lee, X. Chen, B. Peng, D. Liu, and L. Liang. "Puppet: Energy Efficient Task Mapping For Storage-Less and Converter-Less Solar-Powered Non-Volatile Sensor Nodes". In: *2018 IEEE 36th International Conference on Computer Design (ICCD)*. 2018 IEEE 36th International Conference on Computer Design (ICCD). Oct. 2018, pp. 226–233. DOI: 10.1109/ICCD.2018.00042.

[162] K. S. Yildirim and P. Pawelczak. "On Distributed Sensor Fusion in Batteryless Intermittent Networks". In: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). May 2019, pp. 495–501. DOI: 10.1109/DCOSS.2019.00096.

[163] Yizi Gu, Y. Liu, Y. Wang, H. Li, and H. Yang. "NVPsim: A Simulator for Architecture Explorations of Nonvolatile Processors". In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2016 21st Asia and South Pacific

Design Automation Conference (ASP-DAC). Macao, Macao: IEEE, Jan. 2016, pp. 147–152. ISBN: 978-1-4673-9569-4. DOI: 10.1109/ASPDAC.2016.7428003.

[164] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester. "InK: Reactive Kernel for Tiny Batteryless Sensors". In: *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (Shenzhen, China). SenSys '18. New York, NY, USA: ACM, 2018, pp. 41–53. ISBN: 978-1-4503-5952-8. DOI: 10.1145/3274783.3274837.

[165] M. Zhao, Q. Li, M. Xie, Y. Liu, J. Hu, and C. J. Xue. "Software Assisted Non-Volatile Register Reduction for Energy Harvesting Based Cyber-Physical System". In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* (Grenoble, France). DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 567–572. ISBN: 978-3-9815370-4-8. URL: http://dl.acm.org/citation.cfm?id=2755753.2755881 (visited on 02/11/2019).

[166] D. Zhou, K. Qiu, X. Shi, and Y. Liu. "Dual-Threshold Directed Execution Progress Maximization for Nonvolatile Processors". In: *Proceedings of the 15th ACM International Conference on Computing Frontiers* (Ischia, Italy). CF '18. New York, NY, USA: ACM, 2018, pp. 224–227. ISBN: 978-1-4503-5761-6. DOI: 10.1145/3203217.3203263.

[167] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. O. Eversmann. "An 82μA/MHz Microcontroller with Embedded FeRAM for Energy-Harvesting Applications". In: *2011 IEEE International Solid-State Circuits Conference*. 2011 IEEE International Solid-State Circuits Conference. Feb. 2011, pp. 334–336. DOI: 10.1109/ISSCC.2011.5746342.