UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science
Smart Electronic Materials and Systems

Checkpointing Strategies for Efficient Reactive Intermittent Computing Systems

by

Timothy S. Daulby

BEng Electronic Engineering

ORCiD: 0000-0002-7047-6181

A thesis for the degree of
Doctor of Philosophy

January 2023

University of Southampton

Abstract

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

Doctor of Philosophy

Checkpointing Strategies for Efficient Reactive Intermittent Computing Systems

by Timothy S. Daulby

The Internet of Things is a growing field, with 1 trillion devices forecast to be collecting, sending and showing information. These devices exist on a scale from the ultra-power constrained to large, data-rich, mains connected devices. Powering the ultra-constrained devices is challenging, particularly when mass, size and cost must be minimised.

Energy harvesting presents a way of increasing the lifetime of battery powered devices. A more sustainable approach is removing any form of energy storage, significantly reducing the mass, size and cost. Unfortunately, energy harvesters have low power density, temporal variation and unpredictability, making running directly from the supply difficult with traditional computing schemes.

A new paradigm, intermittent computing, overcomes intermittency by storing information in non-volatile memory ahead of power-failure. Reactive checkpointing is one such approach that monitors the supply voltage, and interrupts execution to store volatile data before the supply voltage drops below the minimum operating voltage of the processor.

As intermittent computing matures from proof-of-concept to deploy-able systems, adaptation for system-on-chip implementation is crucial to understand the limitations of existing schemes and application agnostic strategies. By profiling power and memory accesses in RTL and gate-level simulations, as well as analysing memory-usage across benchmarks, design criteria can be established.

All intermittent computing schemes suffer performance from the added time and energy overheads of saving state. Two methods for reducing these overheads for any benchmarks and conditions are presented: PowerNapping and Expedit. These give improvements of up to 46.8% and 84.4% in completion time across a range of benchmarks.

Contents

List of Figures

List of Tables

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. None of this work has been published before submission

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

Acknowledgements

I first of all want to extend my sincere thanks to my supervisors Dr. Alex S. Weddell and Professor Geoff V. Merrett who patiently supported me with guidance and expertise. I would also like to thank my arm supervisor Dr. Anand Savanth for generously offering his expertise and support, particularly during my internship at Arm Research. All of my supervisors have done a fantastic job of helping me to develop and improve my research work, and to persist through all the challenges.

This research wouldn't have been possible without the opportunity to work with Arm Research, thank-you to all those that created an excellent experience. Thank-you also to arm and EPSRC, whose funding made this research possible. I thank Alberto, Theo, Domenico, Sivert, Jie and Ed for their companionship in our intermittent computing research, and Ulises and Sultan for our long, distracting chats. I am grateful to Jonas Svedas and Graham Knight for their help in obtaining netlist simulation data, and Sahan Gamage for explaining the SPI controller. Thanks also to my mother, Marie for proofreading a dissertation she didn't understand a word of.

Finally, I thank all of those who somewhere along the way believed. My wife, parents, family, colleagues and friends whose love and support encouraged me to keep going. I couldn't have done it without you.

And whatever you do, whether in word or deed, do it all in the name of the Lord Jesus, giving thanks to God the Father through him.

Colossians 3:17

To Emily, who graduated from acquaintance to wife
in less time

Abbreviations

ADC analog-to-digital converter

CEM cold-chain equipment monitoring

COTS commercial-off-the-shelf

DFS dynamic frequency scaling

DVS dynamic voltage scaling

EH energy harvesting

FFT fast-fourier transform

FRAM ferroelectric RAM

IoT Internet of Things

LPM low power mode

NVFF non-volatile flip-flop

NVM non-volatile memory

NVP non-volatile processor

PV photovoltaic

RAM random access memory

RF radio frequency

RTL register transfer level

SoC System-on-Chip

TEG thermo-electric generator

VM volatile memory

Nomenclature

T_λ	Time spent sleeping or shutdown before restoring
a	the modified ideality factor
E_α	Energy required to transfer RAM
E_β	Energy required to transfer registers
E_δ	Energy stored in capacitor in excess of energy needed for V_{min}
E_σ	Energy required to transfer system state
$E_{complete}$	Energy required to complete the current task
I_0	Dark saturation current
I_{mp}	Maximum power point current
I_{pv}	Photo generated current
I_{sc}	Short circuit current
k	Boltzmann constant
n_α	Size of random access memory (RAM)
n_β	Size of registers
n_{ia}	Number of interruptions that are avoidable
n_{iu}	Number of interruptions that are unavoidable
P_{active}	Power consumption of the MCU
$P_{harvest}$	Harvester Power Supply
P_{mp}	Maximum power point power
P_{sleep}	Power consumption of the MCU in low power mode
q	Elementary charge on an electron

R_p	Shunt resistance
R_s	Series resistance
T	Device simulation temperature
T_ϕ	Time spent on forward progress
T_a	Time overhead introduced by the intermittent computing algorithm
T_h	Time required to hibernate
T_{on}	Total system on-time
T_r	Time required to restore
T_{sw}	Time overhead of sleeping and waking
V_h	Hibernation threshold voltage
V_t	Voltage required to save state
V_{cap}	Voltage across the capacitor
V_{cc}	Voltage at the common collector
V_{max}	Maximum operating voltage
V_{min}	Minimum operating voltage
V_{mp}	Maximum power point voltage
V_{oc}	Open circuit voltage
V_r	Restore threshold voltage
V_s	Sleep voltage threshold
V_{th}	Thermal voltage

Chapter 1

Introduction

The Internet of Things (IoT) is a fast expanding and developing field, ranging from connected homes to concepts of smart cities with a unity of physical world and cloud achieved by a proliferation of interconnected sensor devices [5].

The IoT is built upon devices ranging from mains connected appliances such as smart fridges, to ultra-constrained "fit and forget" sensor devices. These ultra-constrained embedded sensor devices are experiencing a significant growth in uptake [38]. They present the opportunity to monitor parameters such as pressure, temperature and vibration remotely. Figure 1.1 shows a typical edge device which is able to sense, process and sendreceive data.

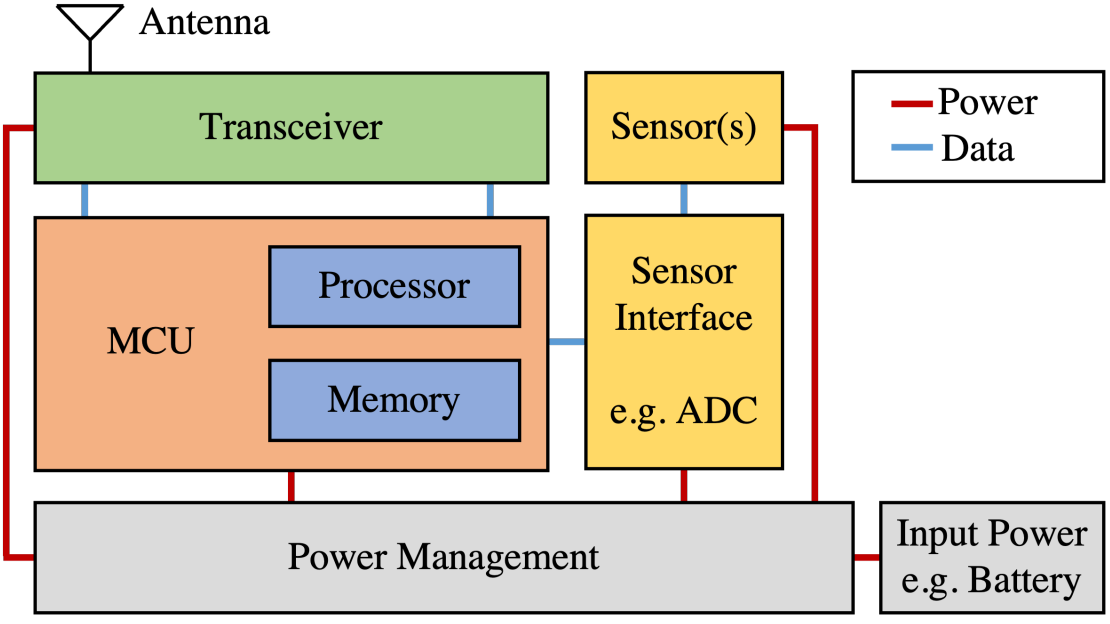


FIGURE 1.1: Example of a wireless IoT sensor device

To maximise the potential deployment opportunities for these devices, they must be autonomously powered and long-running. Some deployments necessitate a small size and mass. Examples of these scenarios include biomedical implants [80], radio frequency identification and structural monitoring.

Powering these devices poses a significant challenge [54]. Relying purely on batteries incurs high maintenance costs with the necessary frequent replacements or charging. Alternatively the batteries must be much larger, which is impractical, and negates many of the benefits of these small devices.

This has prompted developments in equipping IoT devices with energy harvesting (EH). EH could enable 10+ year lifetimes whilst maintaining a small size, but there are a number of challenges to overcome. EH sources have low power density as well as high temporal variation and unpredictability. Furthermore, different kinds of harvester have different supply profiles. Solar and thermoelectric harvesters typically have a slow-varying DC output, whereas wind and vibration harvesters typically have an AC output. In addition, the stability of these sources varies according to their deployment environment. The available energy can vary significantly from device to device, even with identical design due to the nature of distributed deployment.

If the challenges of energy harvesting (EH) sources could be overcome, the benefits of cheap, low-maintenance, truly long-running sensor networks are clear. Our changing environment and climate could be accurately monitored with a much greater number of distributed sensors. Seismic and volcanic activity monitoring could be monitored and modelled with far more data points. Crops could be accurately monitored remotely and simultaneously from anywhere in the country. All supply-chain monitoring could be done in real-time with information such as temperature variation accurately recorded [58].

The motivation for this research direction is clear, and many authors have investigated the means of incorporating EH into IoT devices. Energy neutral computing [92] matches the system energy demand to the incoming supply over a given time period by buffering energy. However these systems depend on rechargeable batteries which suffer from degraded efficiency over time due to charge-cycling [103] or they instead incorporate supercapacitors which increase the size, mass, and cost. Intermittent (or transient) computing is a new paradigm; powering devices from harvested energy without batteries or supercapacitors. They operate directly from the supply; processing, storing and sending data only when power is available. Their limitation is the increased overhead, either hardware or software, resulting in an increase to the time or energy required by the application to complete.

1.1 Intermittent Computing Systems

Intermittent computing systems accept the frequent loss of power resulting from minimal energy buffering, and incorporate techniques for sustaining useful computation across power failures, mitigating the intermittency of the EH sources they are powered from. There are 3 main ways this is achieved in the literature:

1. Periodically copying all volatile data to an non-volatile memory (NVM) in an approach known as checkpointing. The position of these checkpoints are either determined at runtime (reactive approaches) or compile time (static approaches).
2. Ensuring that tasks are completed atomically (i.e. in one power cycle), such that progress can be made in short bursts, or rolled-back safely. These tasks must carefully handle shared variables. This is known as a task-based approach.
3. Fully non-volatile systems where state is always maintained. This requires non-volatility down to flip-flop level, and these devices are referred to as non-volatile processors (NVPs).

All ensure system state is contained in NVM before power failures so that computational progress can be sustained across power cycles. Each approach incurs a time and energy overhead, either from checkpointing, re-execution or task-switching. The smaller these overheads, the more energy and time can be dedicated to forward progress, that is computation beneficial to the progress of the main application being executed.

Checkpoints are set at compile time by a number of schemes [67, 89] which can be done without additional hardware, but often results in re-execution and unnecessary checkpoints. Reactive checkpointing schemes [8, 7, 55, 101] instead monitor supply voltage, such that failures can be detected and state saved before dropping below the minimum operating voltage of the processor. This approach is the most failure resilient since complete state can be saved at any point in the execution, however, it incurs a large time and energy overhead if the checkpoints are not optimised.

Task-based approaches have a much smaller overhead than reactive, often saving only certain registers to maintain consistency, however they put a significant burden on the programmer to define appropriately sized tasks, and often result in large amounts of energy being spent on re-execution.

Non-volatile processors (NVPs) [69, 71] typically have non-volatile flip-flops (NVFFs) which can be backed up in parallel. This leads to checkpoint times less than 10µs, which is more than 180x faster than the 1.8ms of a leading checkpointing approach [7]. These devices are not yet commercially available, and have a larger on-chip footprint than ordinary processors, increasing costs.

By removing the storage buffer, the size, mass and cost of the device can be reduced, and if the intermittency can be handled by these schemes, perpertual sensing is a realisable goal. SoftBank (owner of UK's arm Holdings) CEO Masayoshi Son stated that "In the next twenty years, a trillion IoT devices are coming... We are on the brink of an information revolution that will redefine all industries." [104]. Intermittent computing is sure to play a key role in the pursuit of this ambition. Unfortunately many of the existing systems fail to achieve sufficient forward progress or energy efficiency to sense, process and transmit data on the constrained intermittent energy budgets. To increase the performance of these devices, the energy and time overheads must be reduced and the efficiency of the design improved. A number of research questions have been identified in pursuit of this in Section 1.3 which, when answered, should move the research closer to a wider uptake of intermittent computing systems and present a valuable contribution to the field.

1.2 Research Justification

The field of intermittent computing has matured significantly during the course of my research, from tens of articles in total to over 100 articles published in 2021 alone across all three techniques. Reactive checkpointing approaches that checkpoint in response to a falling voltage are one of the least common, with few articles published using this technique. [89, 8, 7, 55, 73, 101]

To explain the premise of a reactive approach, the checkpointing approach, of which reactive is a subset, must first be understood in more detail. At a checkpoint, data is copied from volatile memory (VM) and registers, to a NVM before power failure, incurring a time and energy overhead. This data is then copied back when restoring system state when power returns. Reactive schemes trigger a checkpoint in response to a declining voltage.

Hibernus [8] is an example of a reactive scheme where the supply voltage is monitored, checkpointing only when it drops below a set threshold (hibernate threshold) close to the minimum operating point of the processor, as shown in Figure 1.2. When power increases beyond a restore voltage threshold, the program state is restored and execution is resumed.

There are drawbacks to reactive intermittent computing approaches such as inconsistent peripheral operation, especially given the task cannot detect if its been interrupted or the large checkpointing overhead, however there are also significant benefits. It is possible to abstract the complexity and contain it within interrupt routines, thereby removing the burden from the programmer. They also eliminate re-execution[1], and the associated overheads.

[1] Re-execution is repeating processing that was lost due to power failure occurring whilst data was held only in VM.

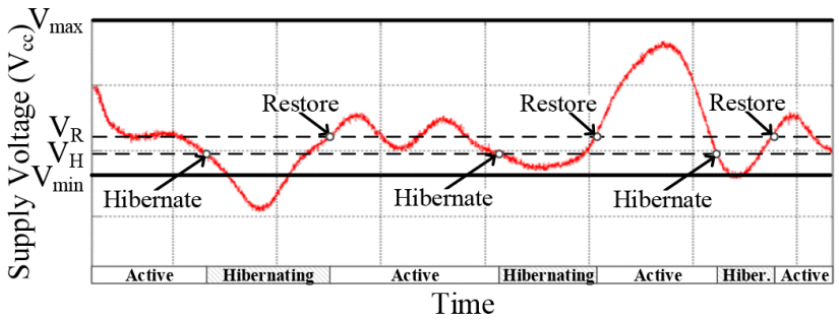


FIGURE 1.2: Operation of Hibernus in response to intermittent supply voltage taken from [8].

A key motivation of intermittent computing schemes is small mass, size and cost. To manufacture these devices at scale and cheaply, System-on-Chip (SoC) implementations are a natural progression, yet few designs exist in the literature, and only a handful of evaluations of this kind are performed in simulation. By taking a broad view of the whole system including energy harvester, software application characteristics, NVM choice, etc... the assumptions about intermittent computing can be investigated and future schemes developed in an informed way. Alternatively, implementing a reactive approach on a SoC would be expected to give further insight. Intermittent computing approaches combine such a wide range of harvesters, sensors and applications that thorough evaluation is challenging, and many schemes apply either few benchmarks or few supply profiles. This presents opportunities to explore the impact of a wider range of benchmarks and EH profiles on the efficacy of intermittent computing approaches.

Balsamo et al, [7] incorporated further runtime energy awareness to set the hibernate/restore threshold dynamically, however few works have built upon this foundation, and there exist many opportunities to improve these reactive systems by leveraging this enhanced energy awareness. By recognising not only the availability of harvested energy, but also the characteristics of the harvester, there is scope for new designs and improvements to reactive intermittent computing systems.

In summary, intermittent computing research should look ahead to SoC implementation and recognise the remaining opportunities for performance increase that exist in the energy-awareness of these devices, and through detailed evaluation of the whole system encompassing application, energy harvester and NVM memory technology.

1.3 Research Questions

The questions addressed in this research are:

1. To what extent is memory important for SoC based intermittent computing? As the field matures from proof-of-concept to commercial SoC implementation, the contents and design of memory become increasingly important. To answer this question an intermittent computing approach is implemented on register transfer level (RTL) simulation for cycle-accurate memory profiling, as well as exploring NVM technologies and a range of benchmarks.

2. How can intermittent computing designs be optimised for high-power, yet discontinuous power sources? Some supplies, such as piezoelectric generators, provide periods of high power for computation, but intermittently. When energy is available it can be used aggressively, but when power fails the buffered energy should be used optimally for increased forward progress. This is justified, opportunities are explored and a scheme experimentally validated.

3. How can checkpoint overheads be minimised for slowly-varying, but low-power, sources? An energy harvester, such as a photovoltaic (PV) cell, generating a slowly-varying current supply that is less than the active consumption of a device will cause frequent interruptions as the storage buffer is depleted. Reducing the current consumption, by placing the CPU in a low power mode (LPM) instead of checkpointing, allows supply recovery. A scheme is developed and validated based on this concept that avoids the energy and time overheads of saving state.

1.4 Contributions

The contributions of the research in this thesis can be summarised as:

1. **A DMA-based intermittent computing scheme that handles data transfer between volatile memory (VM) and non-volatile memory (NVM) when checkpointing.** The approach, Expedit, reduces the time and energy overheads of a state-of-the-art approach by up to 85.6%, and giving up to a 84.4% improvement in completion time as a result by using a DMA rather than the CPU to handle data transfer.
2. **Implementation, analysis and evaluation of a PowerNapping approach** This approach avoids checkpoints by instead using a sleep-state. This results in an 88.3% reduction in time and energy overheads demonstrated over a state-of-the-art reactive checkpointing approach, leading to up to 46.8% improvement in execution time.

The contributions listed above have enabled intermittent computing systems to become significantly more energy efficient, by minimising the overheads of checkpointing. This creates an improvement in the forward progress allowing the schemes to be implemented

on more constrained devices, such as SoC implementations with a smaller capacitance. Finally, both the approaches are instrumented within interrupt routines, allowing the complexity to be abstracted from the developer, further enabling the uptake of intermittent computing devices. The source code for both these schemes is available at https://github.com/UoS-EEC/Improved-Transient-Computing.git

1.5 Publications

The following manuscripts were published as a direct result of the work presented in this thesis:

Tim Daulby, Anand Savanth, Alex S Weddell, and Geoff V Merrett. Comparing NVM Technologies through the Lens of Intermittent Computation. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ENSsys '20, pages 77–78, New York, NY, USA, November 2020. Association for Computing Machinery. ISBN 978-1-4503-8129-1. doi: 10.1145/3417308.3430268. URL https://doi.org/10.1145/3417308.3430268

Tim Daulby, Anand Savanth, Geoff V. Merrett, and Alex S. Weddell. Improving the Forward Progress of Transient Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):444–452, March 2021. ISSN 1937-4151. doi: 10.1109/TCAD.2020.2999913

1.6 Thesis Structure

Chapter 2 explains the necessary subsystems of an IoT long-running sensor system, followed by a review of the existing state-of-the-art in intermittent computing systems. Energy harvester characteristics and applications are also explained. Intermittent computing is implemented on RTL simulation for a large range of benchmarks giving insight for future designs, along with other data such as NVM technologies in Chapter 3. Chapter 4 Explains the benefits of energy awareness and presents Expedit for faster, more energy efficient checkpointing. Chapter 5 explores intermittent computing systems in response to a consistent low-power supply and presents the PowerNapping scheme. Chapter 6 gives the conclusions that can be drawn from this thesis, and suggestions for future work.

Chapter 2

A Review of Intermittent Computing Systems

In pursuit of increased lifetimes and performance for pervasive IoT sensor devices, energy harvesting (EH) is present in much of the latest research. Furthermore the removal of batteries from these devices has prompted a field of research typically referred to as “battery-less”, “transient” or “intermittent” computing to reflect the intermittent nature of the resulting supply. This chapter presents a review of the subsystems that make up these devices, as well as a detailed survey of the emerging approaches to overcoming this intermittency in the supply.

2.1 EH-IoT Sensor Architecture

There are a range of architectures for perpetually powered sensors for the IoT, however, there are a few fundamental subsystems that are essential to these devices. These can be seen in Figure 2.1. This section provides a background to the research, explaining the operation of each subsystem and introducing some challenges and opportunities. These will be viewed primarily through the lens of intermittent computing

2.1.1 Microcontroller

The Microcontroller (MCU) contains the CPU, registers, random access memory (RAM), clock and I/O control unit. They can be integrated with additional hardware into a complete SoC, or discrete MCUs can be combined with off-the-shelf components for faster design, build and testing.

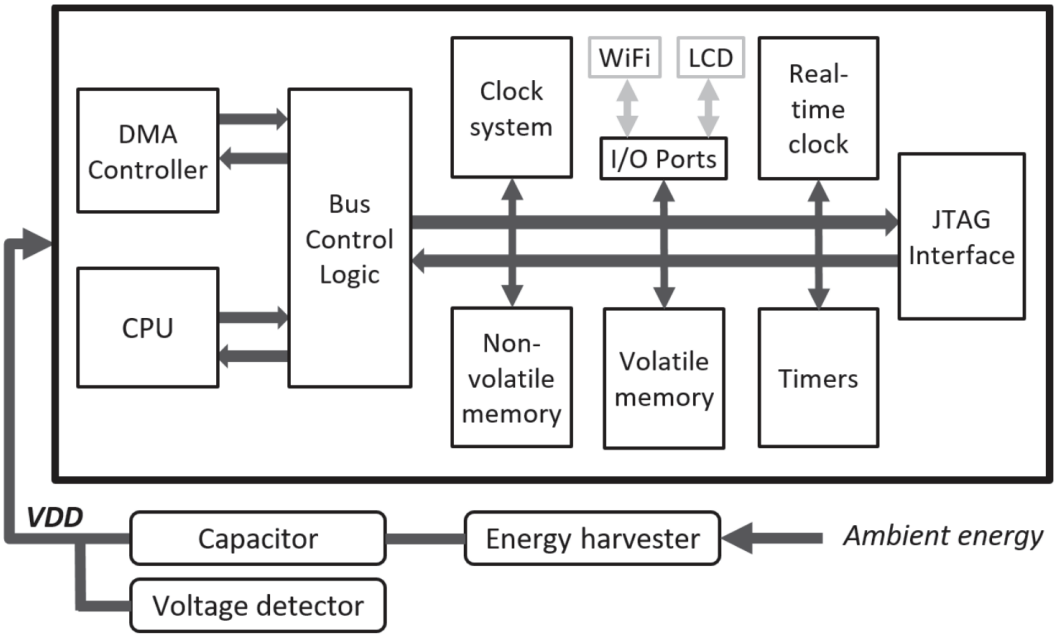


FIGURE 2.1: Example of a perpetually powered sensor architecture taken from [22]

CPU

The CPU is critical in low-power systems, providing opportunities for a range of energy efficiency savings, either in the hardware design, or in software. Designing a more efficient CPU is far beyond the scope of this thesis, however, reducing its active time will lower the energy consumption. When in sleep mode, the CPU consumes significantly less power. There are typically a number of low power modes (LPMs) with a trade-off between power consumption and wake up time. Many energy efficient sensor devices are built with 16-bit CPUs, such as the MSP430, for their reduced complexity and resultant energy efficiency. However, the Cortex-M0 from arm demonstrates that 32-bit architectures can have competitive energy performance, whilst enabling greater complexity and performance. The use of both of these architectures are considered in this research as alternative, but equally viable platforms. Perpetual EH devices, and particularly battery-less ones are still in their infancy, and broadening the scope of the research in this way should reach a wider range of needs.

RAM RAM contains the program information, registers, and cache. RAM is volatile and so data is lost on power failure. The advantage of this VM is that the speed, density, number of read/write cycles and price are all improved over existing non-volatile solutions. This can be dynamic-RAM (DRAM) or static-RAM (SRAM).

Flash Flash memory is one of the most widely used NVMs. Found in USB drives, SSDs, SD cards, etc... It has a finite endurance and a slow erase/write process, but is cheap and compact. It is also still the predominant memory in embedded devices [10].

FRAM Ferroelectric RAM (FRAM) is similar to DRAM but instead of using a capacitor, it has a thin film of ferroelectric material, storing the state as electric polarisation. It is a form of NVM with faster read/write speeds than flash but is much more expensive.

Table 2.1 gives typical read/write speeds and power consumption for each of these memory technologies.

TABLE 2.1: Read/write speeds and power consumption for popular memory technologies

Memory	Read	Write	
	Speed	Energy (pJ/bit)	Speed
SRAM	20-40ns [110]	0.01[18]	50ns[48]
DRAM	60-100ns[110]	>1[18]	<10-50ns[82]
Flash	50ns[82]	<0.01[49]	0.1-1ms[82]
FRAM	20-80ns[79]	0.03[49]	10-50ns[79][82]

In an FRAM-enabled MCU, MSP430FR5739 [111], Jayakumar et al. [56] observed that the FRAM access latency is 3x longer compared to the on-chip SRAM, motivating a hybrid FRAM-SRAM approach. Another concern with FRAM is that it does not currently integrate well with CMOS [94], leading authors to consider alternative technologies such as spin-RAM.

2.1.2 DMA Controller

A direct memory access (DMA) controller is a hardware device, integrated into many MCUs, that moves data directly between peripherals and memory. This is typically used for applications where a high data transfer rate is necessary, or a short latency is required. They reduce the strain on the CPU at the expense of including additional hardware.

The DMA requests permission from the CPU to take over the bus, data and address lines. It is responsible for generating addresses and keeping track of the number of bytes moved [48]. They are much simpler than a CPU and have lower power requirements. Traditionally their benefit is faster transfer speeds, however, in an embedded system, there is potential for energy savings. Figure 2.2 shows the operation of a DMA system.

The CPU first enables the DMA controller, initialising with a source address, destination address and size of transfer. The DMA controls this transfer, before returning an acknowledgement to the CPU. There are three main modes of operation for a DMA:

Burst/ Block Transfer This is the fastest mode of operation, where data is transferred as a block, interrupting the CPU and buses until completion. The processor is held by a “Hold” signal until the transfer is complete and the DMA handles address incrementation. If there is a large transfer this can cause problems for the CPU.

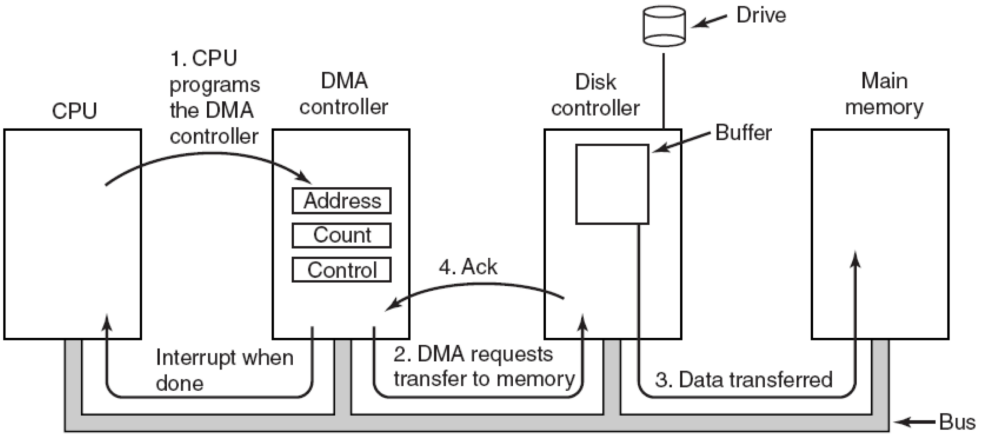


FIGURE 2.2: Operation of a DMA transfer taken from [109]

Cycle Steal/ Single Transfer In this mode only a single data value is transferred for each DMA request assertion, making it slower than block transfer. The DMA controller releases the buses after every bit so that the CPU is not held.

Transparent/ Hidden DMA Transfer This is the slowest mode of operation. In this mode, the DMA waits until the buses are not in use by the CPU and then takes control of the lines. This is transparent to the CPU. In this way, the processor is not interrupted [59].

2.1.3 Sensors

To perform sensing, an input transducer must be used. This converts some physical parameter into an electrical signal. There are a large range of transducers, a selection of which are listed in Table 2.2.

Different applications will require different sensors. The power consumption and data rate of these transducers can vary by orders of magnitude, making this one of the most important considerations when designing for specific applications. This Thesis presents general approaches with benchmarks that stress the CPU, and would form the building blocks of applications, but does not investigate optimisations relating to the use of sensors.

2.1.4 Transceiver

While size, weight, cost, and power are considered some of the key considerations for IoT; frequency, modulation protocols and intelligence are also important [28]. ZigBee/IEEE802.15.4 is an example of a low power personal area network (PAN) technology designed for low data rates (250kb/s). The range is around 20m, making it suitable for indoor IoT networks, or further if using mesh networks. This is just one of many examples of

TABLE 2.2: Common Input Transducers reproduced from [32]

Physical Parameter	Input Transducer
Light Level	Light Dependant Resitor (LDR) Photodiode Photo-transistor Solar Cell
Temperature	Thermocouple Thermistor Resistive Temperature Detectors
Humidity	Capacitive humidity sensor Resistive humidity sensor Thermal conductivity humidity sensor
Force/Pressure	Strain Gauge Pressure Switch
Position	Potentiometer Encoders Reflective/Slotted Opto-switch Linear Variable Differential Transformer (LVDT) Inductive Proximity Position Sensor
Speed	Tacho-generator Reflective/Slotted Opto-coupler Doppler Effect Sensors
Sound	Carbon Microphone Piezo-electric Crystal

low power transceiver protocols for the IoT. Practical implementations range from stand-alone transceivers to complete SoC integration. Table 2.3 shows the overheads for a range of transmission protocols.

TABLE 2.3: Overheads for a range of transmission protocols reproduced from Rodriguez [90]

Parameter	Bluetooth	UWB	ZigBee	Wi-Fi
V_{cc} (V)	1.8	3.3	3.0	3.3
Transmit (mA)	57	227.3	24.7	219
Receive (mA)	47	227.3	27	215
Payload (bytes)	339	2044	102	2312

Transceivers have a high energy consumption. Their use can be minimised by pre-processing of the data; both compression and selection. In addition their use can be matched to periods of high energy availability in devices without real-time data requirements. Bakar et al. propose a heuristic approach where the data transmission can be varied in a number of ways according to information about the incoming supply. This includes reducing the transmission power, incorporating additional compression to the data being sent during moderate energy state or finally only inferring information from the data to transmit even fewer packets at lowest power [6].

2.2 Energy Harvesting

Energy harvesting (EH) is the process of capturing energy from the surrounding environment that would otherwise be lost. EH presents a promising solution to power availability issues for sensing systems, but also brings a number of key problems. These include temporal variation, unpredictability and poor efficiency [91]. Figure 2.3 shows the power characteristics for a range of sources.

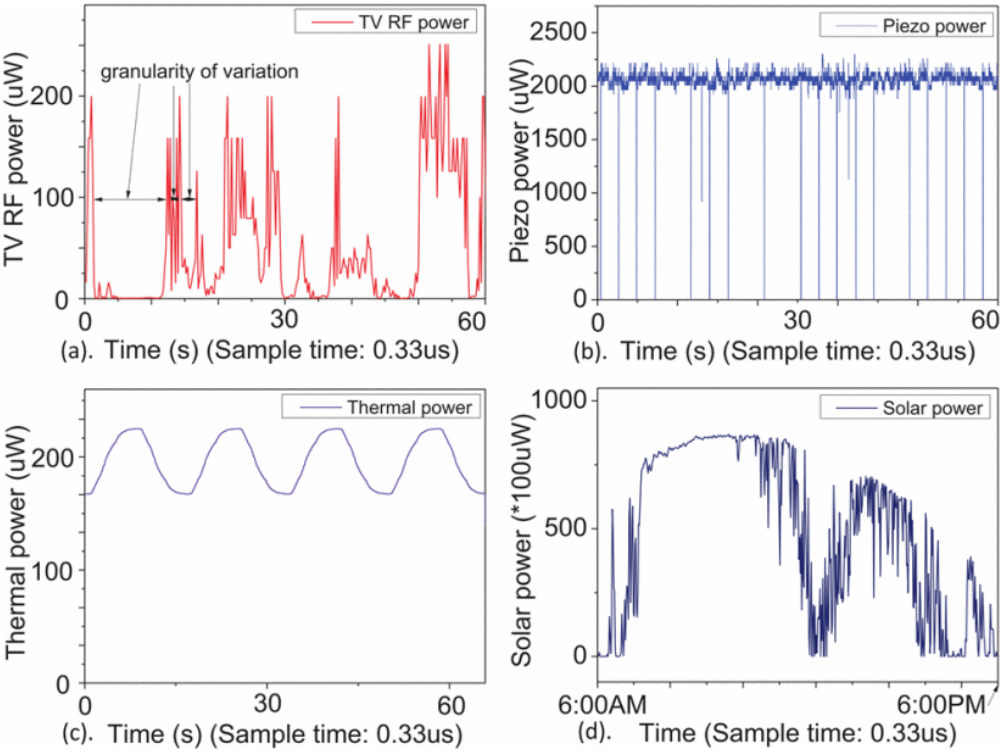


FIGURE 2.3: Power traces a)TV RF b) Piezo c) Thermal d) Solar taken from [70]

The achievable power and its characteristics have been determined for a range of sources. The following sources are referenced by the literature as important for low-power embedded sensors. Other harvesting techniques include flow-based (turbines), wind-based and hydro-based. These are often too bulky for IoT applications.

By understanding the characteristics of a range of harvesters and the applications they may be appropriate for, opportunity opens up to design intermittent computing schemes that extract greater efficiency from the source and further improve forward progress. This energy-informed design is a key focus of the work presented in this Thesis.

2.2.1 Photovoltaic Energy Harvesting

Photovoltaic (PV) harvesters convert light energy (photons) into electrical energy. The voltage is maximal at 0 current, this is known as the open circuit voltage, V_{oc}. Conversely,

current is maximal at 0 voltage, this is known as the short circuit current, I_{sc}. Along the PV cell's I-V curve, there exists a point of maximum power. The voltage and current at this maximum power point are abbreviated to V_{mp} and I_{mp}. This can be better understood by observing Figure 2.4.

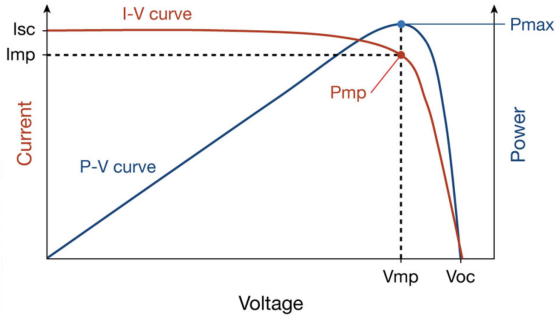


FIGURE 2.4: Graph showing IV and PV characteristics of a PV cell taken from [40]

PV cells have the highest power density of any harvester [12]. Some degree of solar energy is available in most environments, although different harvesters respond differently to these environments.

PV has been in use for wireless sensors for over a decade [88] and given a suitable environment can already achieve perpetual sensing. The greatest limitation is temporal variance with little to no power being collected at night. One interesting development for IoT is indoor PV cells that are designed specifically to convert artificial light. These would be suitable for venues such as hospitals, stadiums and industrial buildings where there is typically more light available [99].

2.2.2 Thermo-Electric Energy Harvesting

Thermal energy can be found in almost any environment and much of it goes to waste. There are three thermoelectric effects for conversion between electrical and thermal energy, namely the Seebeck, Peltier and Thomson effects [13]. The most relevant of these being the Seebeck effect which produces a voltage as a result of a temperature gradient.

Thermo-electric generators (TEGs) produce a linear current-voltage characteristic, and so the maximum power point can be tracked. Figure 2.5 shows an example I-V and P-V characteristic.

TEGs are rarely used in the literature due to their poor conversion efficiency (5-6%) [99], but were demonstrated as an effective energy neutral environmental monitor for server farms by Rossi et al. [92], with minimum 5 min intervals for measurement and transmission of environmental parameters (temperature, humidity, light intensity, self supply voltage, and air safety through a MOX gas sensor). This demonstrates that energy harvesters can be beneficial given the correct deployment environment. The characteristics of the chosen harvester will always be an environment-based design decision.

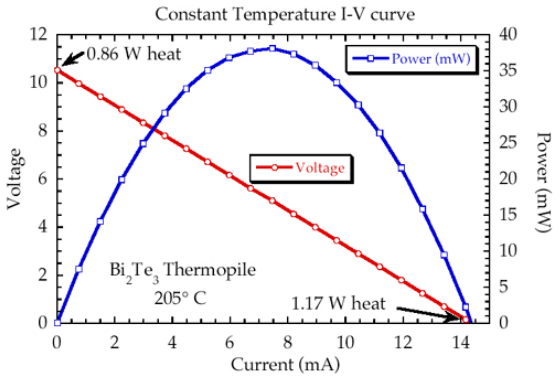


FIGURE 2.5: I-V and P-V characteristic of a thermoelectric generator taken from [20]

2.2.3 RF Harvesting

Received radio waves can be converted into useful power through radio frequency (RF) harvesting antennae. Their largest setback is the need for a comparatively large antenna for useful power to be generated. This, in combination with the need for correct orientation of the antenna, makes it impractical in many situations. This antenna can also only access a close bank of frequencies [31]. There are few situations where this is of great value for IoT devices. Power received is in the range of less than 1µW without a dedicated short range transmitter.

As an ambient source, RF is limited, but as a method of wireless power transfer can be quite effective. This transfer of energy can be separate to the communications or sent as a combined signal [99]. This power transmission as part of a wireless sensor network, prompts the consideration of power management on a system scale and no longer on a single device [30]. This research concentrates on the efficiency of a single device, and so leaves a number of open challenges in this field, such as EH receiver design, energy arrival rate, minimum numbers of dedicated energy sources, scheduling of energy transmission and multi-path energy routing.

2.2.4 Kinetic Energy Harvesting

Kinetic EH is a process of converting kinetic energy into electrical energy. There are many sources of kinetic energy, however unlike solar and thermal, the design of the harvester often operates best at resonance with the frequency it is harvesting.

A common method of generating energy is using inertial generators. These systems have both an inertial frame and suspended inertial mass. Kinetic energy produces a relative displacement that is converted to electrical energy by a number of means.

Piezoelectric generators typically produce a high voltage and low current. Figure 2.6 shows I-V characteristics for a piezoelectric vibration harvester. The output impedance is typically

high (>100kΩ). Electromagnetic generators, unlike piezoelectric, have a high current with a low voltage (typically <1V). Electrostatic generators produce high output voltages (>100V) but, as with piezoelectric, have limited current-supplying capability [13].

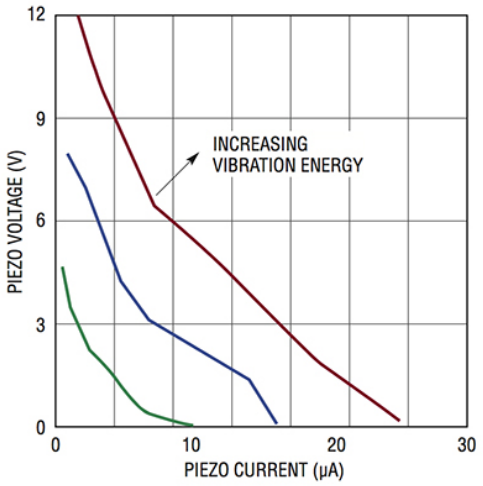


FIGURE 2.6: IV curve for a piezoelectric harvester taken from [34]

2.2.5 Ambient vs External

Many of these sources occur naturally such as wind, electromagnetic radiation (solar and RF), vibration, thermal. These can be considered ambient sources, but external sources provide new application directions. Human-based for example with locomotion, body heat and even blood flow. Other examples include mechanical based: vibrations, pressure, stress-strain or energy transfer. Energy can be transferred through wire, microwave beam, magnetic resonance, reflected solar, RF energy or laser beam [60].

2.2.6 Summary

Table 2.4 shows an estimation of the power to expect to generate from a number of sources. It was intended to make a more comprehensive estimation for emerging technologies, however the amount of power is so dependent on the size of the harvester and the nature of the source, i.e light intensity that it is challenging to generalise. There are a range of review papers that give a good review of the state-of-the-art [35, 99], but as their tables illustrate, there are many variables in predicting power generation. Without modelling a specific device, the power cannot be precisely anticipated.

Energy harvesters constitute a significant proportion of the overall node volume [35], and so whilst some harvesters provide power that could be considered abundant for low power sensors, they are likely to be prohibitively large. Therefore for size-constrained devices,

Source	Power Density
Photovoltaic	Outdoors (direct sun): $15 mW/cm^2$ Outdoors (cloudy day): $0.15 mW/cm^2$ Indoors: $< 10 \mu W/cm^2$ [15][21][93][116]
Thermoelectric	Human: $30 \mu W/cm^2$ Industrial: $1-10 mW/cm^2$ [50][116]
Piezoelectric	$250\ \mu W/cm^3$ $330\ \mu W/cm^3$ (shoe inserts) [21][93]
Electromagnetic	Human motion: $1-4 \mu W/cm^3$ [85][108] Industrial: $306 \mu W/cm^3$ [14], $800 \mu W/cm^3$ [108]
Electrostatic	$50-100 \mu W/cm^3$ [113]
RF	GSM 900/1800MHz: $0.1 \mu W/cm^2$ WiFi 2.4GHz: $0.01 \mu W/cm^2$ [15]

TABLE 2.4: Harvested power from common energy sources reproduced from [12]

low-power, intermittent, unpredictable supplies have to be expected. There are some characteristics that are consistent, and devices often have a maximum power point. Exploiting this with sufficient energy management will increase the power given by the supply. Section 4.2 further explores the use of energy harvesters, and how their characteristics can be generalised and exploited in design.

2.2.7 Energy Management

There are four primary ways energy harvesters can be used in a system. Direct power, battery-free (supercapacitors), battery recharging or battery activation. Different harvesters perform better under particular strategies. Particularly those that require impedance matching to operate at a maximum power point.

Ideally a system could power itself from multiple sources, but as identified by Weddell et al. [120] there is no golden bullet solution. Multi-source systems either require certain harvesters to be used or certain interface circuits to be used. Most are not energy aware and do not automatically detect hardware. One solution may be smart harvesters that contain microprocessors to interface directly with each other and the embedded device.

Ultimately a device that is maximally efficient in using available power, but does not extract the maximum efficiency from the harvester is not overall efficient. Protocols must be adapted to maximise the information processing for a given energy availability instead of pure efficiency savings [99]. Further improvements could come from efficient (i.e. low processing cost) prediction techniques and simulation environments that extend to all aspects of EH for IoT devices.

2.2.7.1 Energy Storage

Energy storage is a key component of many existing autonomous sensor designs, however there are some key limitations imposed by them. Firstly, they increase the mass, cost and size of the overall system. There are many applications where these are critical factors. They also suffer from inefficiency when charging and discharging due to the resistance in the path.

Batteries There are two classes of battery, primary and secondary. Primary batteries have a single charge and run until failure, whereas secondary batteries can be recharged and used many times. Non-EH autonomous systems would most likely use primary batteries that are manually replaced, however EH necessitates secondary if they are used at all. The most prominent issue with secondary batteries for long term use is the reduction in capacity with charge cycling. This eventually leads to total failure.

Super-capacitors Super-capacitors have a greater energy density than standard electrolytic capacitors, but retain many of their traits. They do not have the same charge cycling issues as batteries, but suffer capacitor inefficiencies. The first of these is that only a fraction of a capacitors capacity can be used to store energy deliverable to the load. Secondly there is a cold-start period where charge must be built up in the capacitor until $V_{cap} \geq V_{min}$. Finally energy is lost due to leakage across the internal resistance. There are some advantages to using capacitors. If the EH source provides less than the minimum operating current at the minimum load voltage, the device will not be able to operate without charge storage. Also, capacitor discharge is predictable, unlike the sudden loss of power associated with harvesting sources. This allows minimum time intervals to be utilised by the system's control algorithms. Jiang et al. present an interesting approach where a super-capacitor serves as a buffer to reduce the impact of charge-cycling, but in tandem with a battery to reduce the limitations of single storage schemes [123]. This is a good solution, however does nothing to reduce the size, mass and cost of the system.

2.2.7.2 Input Power Conditioning

Impedance matching is essential for extracting the maximum power from an EH source, either for storage or direct use. Correct impedance matching is known as the maximum power point. There are a number of ways to determine this point, either statically or dynamically. In systems where maximum energy or power is needed, maximum power point tracking (MPPT) will ensure the harvester is load matched as accurately as possible, without excessive consumption from the MPPT scheme.

2.2.7.3 Output Power Conditioning

The output power needs to be regulated for the CPU and other system components to function. This should be done with minimal waste. High efficiency is key, but there are some more interesting approaches in combination with this. Reciprocal conversion schemes, where a single conversion scheme provides both input and output conditioning, reduce the complexity and power consumption. Savanth et al. present such a scheme with selective direct operation, such that within certain thresholds, no conversion is applied to an incoming DC voltage from a PV source [97].

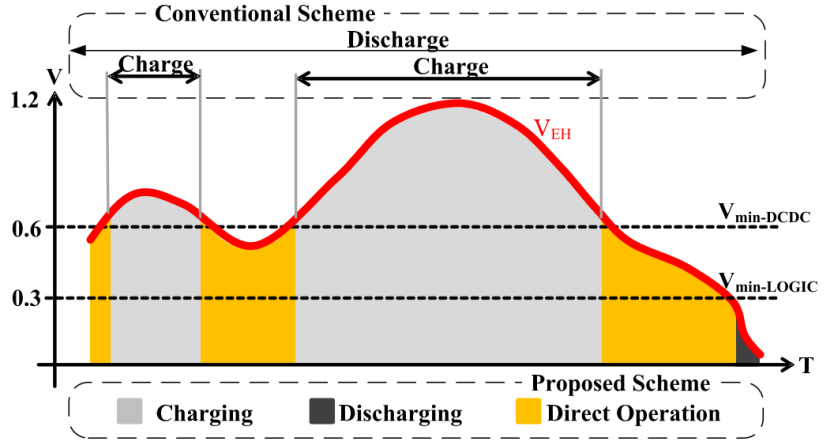


FIGURE 2.7: Conceptual representation of reciprocal conversion taken from [97]

2.3 EH Sensor Approaches

Providing sufficient energy to devices for the internet of things is perceived to be a significant barrier to deployment [23, 54]. A range of solutions are found in the literature. Earlier solutions all rely on batteries and/or supercapacitors to provide long-running computation.

2.3.1 Event-Driven

Batteries are a tried and tested means of supplying power in mobile applications. It makes sense, therefore, that the earliest devices were entirely battery powered. Systems such as the Mica node [47] and the Telos mote [87] enabled some of the earliest sensor network systems, requiring only AA batteries to sustain save-and-send sensing applications.

These devices are event-driven, in that they spent the majority of their time in a low-power mode, waking only to perform sensing, processing or sending tasks. The projected lifetime of these devices is proportional to the duty-cycling. At a 1% duty cycle, Telos can last for almost 3 years with 2 AA batteries. For comparison, the lifetime of the Mica2 mote is 1.5

years and the MicaZ mote is 1 year [86]. For tasks that require a higher duty-cycle or longer-term operation, batteries are not sufficient without a significant increase to their size.

There have been developments since in reducing the energy consumed by elements such as the transceiver [76], read/write to memory, idle CPU, etc. but all of these benefits can transfer to a system that also includes EH. Energy harvesting allows battery energy to be augmented, not simply depleted.

2.3.2 Energy-Neutral

Energy-neutral devices aim to perpetually power devices by matching the average load consumption to the available supply over a given time frame. This is achieved through duty-cycling and/or energy buffering. Duty-cycling can be reducing the load of the processor when energy is scarce, though this is not appropriate for time critical applications. Energy buffering can be done with batteries or supercapacitors.

While Moore's law has continued a little longer, battery technology has not kept up [57]. The energy density of existing battery technology is insufficient for long-term sensing, and the capacity is further reduced by frequent charge-cycling. This motivates researchers to move away from batteries as a source of power for IoT. Some energy-neutral approaches are able to run using super-capacitors to hold charge between power availability.

Hester et al. [41] suggest an approach that uses an adaptive capacitance. This increases the size of the buffer when additional energy is needed for a task. It also decreases the energy buffer for smaller tasks to reduce the cold-start time.

2.4 Intermittent Computing

Designing sensor devices with a low mass, size and cost results in a highly energy constrained device. Without storage such as batteries or supercapacitors these devices must adapt to the characteristics of the energy they are able to harvest. The key problem is sustaining computation across multiple power cycles. The intermittency of EH sources leads to frequent power loss, and CPU shutdown.

Intermittent computing (also known as transient computing) is a new paradigm of systems that have no additional energy storage and are instead able to maintain computation across power cycles through the use of NVM. There are a few ways that the problem of intermittency is addressed in the literature. NVPs, task-based programming models and checkpointing strategies. The ideal case for these devices is a CPU retaining all information on every power loss, therefore being able to resume immediately, with no overheads. Maeng et al. [74] translate this into correctness requirements (C1-3) and performance goals (G1-3) that help to summarise the research direction:

• C1: A program must preserve progress despite losing volatile state on power failures.

• C2: A program must have a consistent view of its state across volatile and non-volatile memory.

• C3: A program must respect high-level atomicity constraints (e.g., sampling related sensors together).

• G1: Applications should place as few restrictions on the processor and memory hardware architecture as possible.

• G2: Applications should be tune-able at design time to use the energy storage capacity efficiently.

• G3: Applications should minimise runtime overhead, memory footprint, restore overhead, and re-executed code.

For the purposes of this research it is important to understand the methods for overcoming intermittency, and the corresponding overheads.

2.4.1 Checkpointing Approaches

The concept of intermittent systems began with checkpointing. This involves polling of the supply voltage to decide if a system snapshot should be taken. If so, the system state (contents of RAM, processor and peripheral registers) are copied into an NVM before power fails. Thus, after a supply interruption, the system state can be restored and the program continues from the point where it was interrupted, instead of re-initialising and executing from the start of the main function. Checkpoints enable computations to continue across power cycles, but introduce additional overhead in the save and restore.

Mementos One of the earliest approaches was Mementos [89]. This scheme polls the supply voltage to estimate the energy remaining in the capacitance of the system. There are 3 triggers for this listed below. They use a WISP RFID [124] with 10µF capacitor.

Loop-latch In this mode a trigger is placed at each loop latch, resulting in an energy check on every loop of the program.

Function-return In this mode a trigger is placed after each call instruction, leading to an energy check every time a function returns.

Timer-aided This mode reduces the frequency of checks by triggering a flag at set time intervals. Combined with loop-latch, or function-return, this only energy checks if they trigger and the flag is raised. It is lowered again for the next trigger point.

Further triggers can be inserted manually. An analog-to-digital converter (ADC) reads the supply voltage and compares this to a set threshold. If found to be below, a state-save is initiated. Mementos is a double-buffering approach, meaning checkpoints are saved in alternate locations. This means that if power failure occurs during checkpointing, a valid snapshot remains.

Mementos was the pre-cursor to many other checkpointing based intermittent computing systems, but had a number of drawbacks. There are a high number of checkpoints due to a conservatively high threshold voltage. ADC demand a significant power-consumption in relation to low-power embedded processors. Recent work shows that because mementos resumes computation after saving system state, there is a significant risk of state inconsistency [67]. This is because the compiler does not distinguish between writes to VM and NVM, meaning partially executed, or re-executed code may generate inconsistencies. This is often referred to as an idempotence violation and is illustrated by Figure 2.8

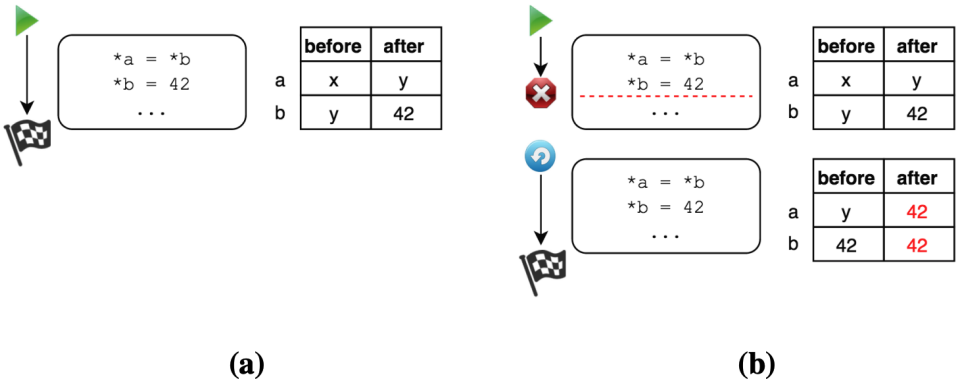


FIGURE 2.8: The same code executed with and without power failure to demonstrate idempotence violations taken from [122]

As you can see continuous execution in Figure 2.8a concludes with a=y and b=42, however when the first two lines are executed and then re-executed, as in Figure 2.8b, the non-volatile state of b has been changed, and a concludes with a wrong result.

Hibernus Hibernus [8] reduced the number of checkpoints and removed the problem of state-inconsistency. Hibernus instead of polling the voltage, initialises a voltage interrupt to trigger a single checkpoint per power cycle. It relies on FRAM to capture system state using only the energy stored in a system's decoupling capacitance. Figure 1.2 shows the operation of Hibernus.

Hibernus sets two thresholds, one for hibernating (V_h) and the other for restoring (V_r). Hibernating involves storing the system state (RAM, Core Registers and General Purpose

Registers) into NVM, and entering a LPM. Restoring is when the system is able to restore system state from NVM and resume useful computation, resuming typically occurs after power failure.

There are two key equations for setting these thresholds. Equation 2.1 shows the amount of energy (E_σ) required to transfer system state to NVM.

$$E_\sigma = n_\alpha E_\alpha + n_\beta E_\beta \tag{2.1}$$

Where n_α and n_β are the sizes of the RAM and registers (in bytes) respectively and E_α and E_β are the energy amounts required to copy each byte to NVM (J/byte). Equation 2.2 shows the amount of energy (E_δ) stored in the system capacitance (C) above the minimum required before system shut down for a given voltage (V_t).

$$E_\delta = \frac{V_t^2 - V_{min}^2}{2} \cdot \sum C \tag{2.2}$$

Where V_{min} is the minimum voltage of the processor. The system is able to save a complete snapshot, even on instant total power failure, provided the condition $E_\delta > E_\sigma$ is met. V should be adjusted to ensure this, giving $V_h = V$. If this condition is not satisfied even if $V_t = V_{max}$, where V_{max} is the maximum operating voltage of the processor, then the capacitance must be increased with an external capacitor for stability to be maintained.

Hibernus reduced the time wasted on checkpointing, but does not utilise any of the energy between checkpointing and restoring. Overall the energy saving was higher. The key issue with Hibernus is that it requires prior measurement of the system properties to set V_h and V_r. This lack of runtime adjustment could create instability if there is a change to system properties.

V_h is set for a worst case scenario where every byte must be re-written, and energy fails immediately. When these conditions aren't met, significant energy is wasted by this 'safe' value of V_h.

CTPL Texas Instruments introduced a compute through power loss function [2] to their FRAM devices. This is similar to Hibernus with a fixed V_h of 2.6V and an ADC sampling the supply at 1kHz. It saves state in 1.12ms, faster than Hibernus, but the threshold is higher than necessary, resulting in wasted computation time. It uses a timer interrupt to decide whether to resume computation if the supply doesn't fail after entering LPM.

QuickRecall Jayakumar et al. [55] present a method similar to that of Hibernus, but with an emphasis on a unified NVM. RAM is forgone, replaced entirely by FRAM. This means that there is less information to be transferred, and therefore a smaller overhead.

The drawback of QuickRecall is the need for a unified FRAM memory. Increasing the use of FRAM increases the overall system cost. It has been shown by Rodriguez et al. [91] that QuickRecall also has a higher current and energy consumption for low interruption frequencies. This is due to the use of FRAM over conventional RAM. Accesses to NVM use more time (3x for MSP430 [56]) and energy (2-2.5x [74]). At higher frequencies the reduced state save overheads give QuickRecall lower total energy consumption.

Hibernus++ Balsamo et al. improved upon their initial work with Hibernus++ [9] by introducing runtime adjustment of its thresholds. When the system is first powered on, it runs a calibration routine that sets V_h. This is shown in figure 2.9.

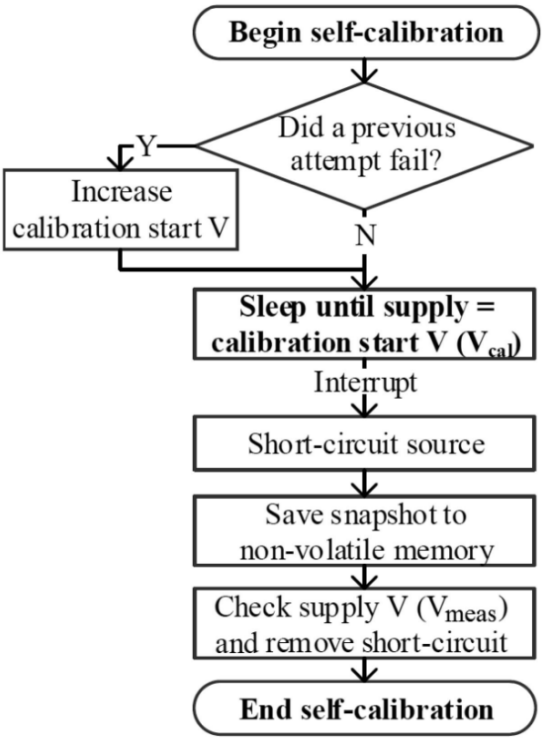


FIGURE 2.9: Calibration routine for setting thresholds in Hibernus++ taken from [7]

If a snapshot fails to complete, V_h is increased by 0.1V, and the application is restarted. Hibernus++ comes close to predicting power failure with hibernations only occurring when the voltage drops close to that of system failure. However there are still situations where Hibernus++ does not perform well. One example is low current EH sources where the current is lower than the active current consumption, leading to frequent hibernations. The PowerNapping approach described in Chapter 5 is built on Hibernus++, so additional details of its operation are given there.

Added sleep state It was suggested by Lukosevicius et al. [68] that energy could be saved in Hibernus++ by introducing a third threshold where the system enters sleep, without saving state. For supplies where the harvested energy is greater than sleep, but lower than active,

the system would recover without failure. However the sleep threshold is set much higher than V_h. This leads to wasted computation any time the system does require hibernation. For highly intermittent sources, the additional sleep state would incur a high penalty. Little experimental validation was included, and the results that were included did not line up with the simulation results given. Further to this, many of the presented equations were found to be incorrect.

Power-Neutral Approach Power neutral systems aim to match the system load to the available supply in order to maximise forward progress at a given point in time. In battery-less systems energy can be aggressively consumed when it is available, because otherwise it is wasted. There are two prominent techniques for this; dynamic voltage scaling (DVS) and dynamic frequency scaling (DFS). DVS is, for a given performance, reaching the minimum possible supply voltage for the system. DFS is where the clock frequency is similarly minimised depending on required performance or available power. With a finite supply such as a battery, this is desirable, however these same techniques can be applied where, instead of minimising voltage or frequency for a set performance, we maximise performance for a set input power. Balsamo et al. [9] present a methodology for power neutral DFS. It is based on Hibernus++, but with an added DFS algorithm. This algorithm effectively checks if the voltage supply is a set amount higher or lower than the system voltage. It then increases or decreases the frequency of the system respectively.

Ratchet Ratchet, by Van Der Woude et al. [122] removes the need for user task boundary definitions by implementing compiler modifications. These modifications track the idempotency of tasks. Idempotency means that the system state remains the same after one or several calls. Idempotency is broken by overwriting to a memory location that has previously been read. Roll-back at this point would cause the read value to have changed, altering the result of the execution. By inserting a checkpoint between the read and write the original section is separated into two idempotent sections. Registers that are updated are known as 'live-in' registers, and these are tracked by the compiler anyway. Only these need to be saved when checkpointing. This is because ratchet allocates all of its state in NVM, which has a higher time/energy overhead. Like Mementos, Ratchet also uses double-buffering, requiring additional memory footprint.

Clank Reducing the number of checkpoints, and the time and energy overheads for checkpointing intermittent computing systems is critical for increased forward progress. Ratchet [122] demonstrated an idempotency tracking approach to greatly reduce the size of each checkpoint, and therefore the time and energy overhead. However, there were a large number of checkpoints, especially for applications with frequent idempotency violations. Clank [44] built upon this work by introducing hardware to complement the software approach.

Idempotence violations occur when a write occurs at the same address as a read. Clank uses hardware to track the addresses which are read-first and write-first. If a memory write access occurs for an address in the read-first buffer, there is an idempotence violation. Instead of checkpointing each time this occurs, Clank saves the address in a volatile Write-back buffer. If any of the buffers are full this triggers a checkpoint. Figure 2.10 shows the in-hardware buffers and management logic of Clank. Buffers are queried and updated in parallel.

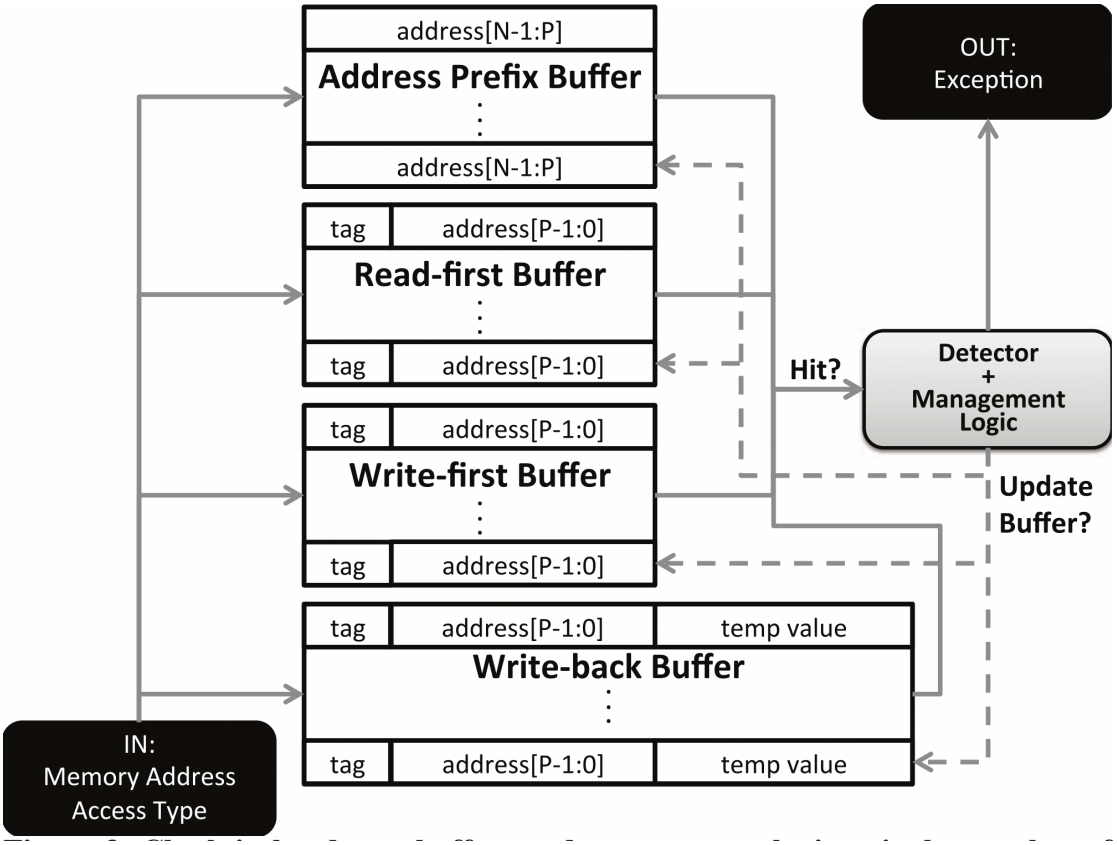


FIGURE 2.10: Clank in-hardware buffers and management logic taken from [44]

Clank checkpoints are not energy-aware and instead occur only when enough idempotency violations occur to fill the buffer. Clank may produce idempotent sections larger than power-on time in which case, no forward progress will be made. If a checkpoint wasn't saved in the last power-cycle a watchdog timer is enabled. This creates an arbitrary checkpoint to ensure forward progress. If a checkpoint still fails to save, the value of time is halved. This is then disable again once a checkpoint is saved.

A second watchdog timer is also included that targets maximised performance. Each checkpoint has a time and energy overhead. Re-executing code not saved since the last checkpoint also has a time and energy overhead. Given the ideal case where there are no program-induced checkpoints it is possible to calculate the value that balances the overhead of re-execution with checkpointing overhead to ensure that the cost of re-execution does not exceed that of checkpointing.

ENZYME ENZYME is very similar to the work of Hibernus, but introduces a frequency modulator and additional pre-characterisation algorithms. Enzyme is designed for consistent low-power sources. It uses the same added sleep state, but uses an algorithm to determine the optimal sleep and wake voltages to operate in the range where the power regulator is most efficient. This benefit of ENZYME would not translate to a highly intermittent source for the same reasons as the added sleep state. Also ENZYME uses an ADC to monitor the voltage. An ADC uses significantly more energy than a voltage reference and comparator (310µW vs 17µW+130µW [8]). Some peripheral devices bottleneck the maximum operation speed, such that operating the CPU at a faster clock frequency, which uses more energy, yields no additional forward progress. By identifying the optimal frequency for each task, the CPU's clock frequency can be changed at task boundaries, potentially reducing power consumption. They discover that ADC sampling, for example, gives an optimal frequency of 4MHz on the MSP430FR6989.

Increased Uptake Balsamo et al. [10] identify that most existing IoT systems contain flash, not FRAM and that drivers need to be compatible with OS APIs. They state that compiler level modifications or hardware-assisted solutions are not ideal for a large range of platforms. They demonstrate a working version of Hibernus on two mbed platforms, with flash memory. Mbed OS is an open-source operating system (OS) that has features to develop and connect IoT platforms based on arm Cortex-M microcontrollers. Using flash and the mbed platform requires increased capacitance of 4.7mF and 4.9mF in line with equation 2.2. These are large capacitors, demonstrating one of the many reasons why flash is not a good choice for intermittent computing systems.

Allocated/Managed-State Sliper et al. [101] present allocated and managed state approaches, referred to in this thesis as iclib, which is the name of the open source repository. They identify that it is inefficient state retention to save the entirety of VM on every power cycle. Allocated state tracks the dynamic memory, and only transfers this on power failure. Managed state goes one step further and implements a lightweight page based memory manager that tracks active and modified regions of memory. They demonstrate an improvement in suspend and restore time by 26.8-86.9% when memory locality is good. When memory locality is poor, the overheads of the scheme can exceed 6X.

2.4.2 Task-Based Approaches

An alternative to checkpointing, is breaking the code into tasks that can be completed within each energy period. Some approaches are energy aware and ensure that there is enough before initiating a task. Others break the code into small, self-contained tasks that has the effect of reducing the roll-back when computation is interrupted.

EMU Gomez et al. [36] identified that for data rich sensors, high amounts of energy were needed for tasks such as saving to flash, or reading from a camera. Saving to an SD card for example requires a high initialisation energy (~10mJ) for each access. Images can instead be buffered in FRAM, with an energy burst to move multiple images. They implement an "Energy Management Unit" (EMU) to buffer low-energy input for high energy bursts using boost/buck conversion. Because of this they require a capacitor of 1.4-4.3mF. By building up energy in this capacitance they can ensure that a task will complete.

DINO Lucia et al. [67] created a programming model known as DINO (Death is not an option). This ensures data-consistency by ensuring tasks are atomic and data is consistent at task boundaries. Registers and the stack are still check-pointed at these task boundaries, which are defined by the programmer. This introduces a time and energy overhead at every boundary, but significantly smaller than systems that save the entire RAM such as Hibernus. DINO need not be energy aware since each task is self-contained and does not manipulate non-volatile data used by other tasks. Instead it uses static multi-versioning, creating copies of modified data to keep volatile and non-volatile state consistent. This means that interrupting a task will not lead to inconsistent state, unlike in Mementos. A significant problem with DINO is that tasks must either be made small to achieve forward progress at low power, or large to reduce the overhead associated with having many checkpoints.

Chain Colin et al. [25] built a task model that relies on data channels. Separate input and output channels alongside volatile-only task variables guarantee consistency, without any checkpoint. They introduce syntax for the programmer to segment the code into these tasks, and the channels to pass information between tasks. The burden is on the programmer to follow the syntax and ensure that variables are moved through the correct channels. This is the only mechanism by which variables can be passed between tasks, and have any access to NVM.

Alpaca Alpaca [74] from Maeng et al. is a more recent approach. Alpaca dynamically versions non-volatile data like DINO and the data is selectively versioned by a compiler-based idempotence analysis, similar to Ratchet, but there are no checkpoints, and a unified FRAM memory is not required. Tasks in Alpaca are atomic and can be re-executed after many interruptions, as if they had run once in a continuous system. Tasks are similar to that of Chain [25], but the memory model is different. Alpaca has task-shared data in NVM and task-local data, which is initialised by that task, in VM. Alpaca copies task-shared data into task-local copies for execution, and commits the updated copies back at the end of a task. In this way Alpaca can execute computation in atomic tasks, re-starting from the most recent task upon power failure, with no coherency errors. The drawbacks of Alpaca are similar to DINO, where tasks must be sized small enough to achieve forward progress, but copying and re-writing all task-shared variables to NVM is not dissimilar to checkpointing,

incurring overhead. Upon every power-failure there is a privatisation and re-execution overhead. While much of the operation is abstracted from the programmer, there is still a burden on the programmer to determine task boundaries. There is a higher burden than DINO, but lower than Chain. Reactive schemes, such as Hibernus++, require no additional code from the programmer.

Chain, Alpaca, Ratchet and DINO all emphasise a need for a system to automate, or at least help create task-boundaries, which remains a manual task.

Mayfly Mayfly [43] is a significant paper which focuses on the idea that sensor data ages, and may lose its relevance as priorities change and opportunities are missed. It is a task scheduling runtime and language which better combines timing with sensing goals. Mayfly programs are a series of connected tasks with timing constraints attached to the data generated by each task. Remanence timekeepers [42] maintain time across power failures, maintaining the clock with less than 20nA current draw. If the power outage is too long, all time sensitive data will be rolled back since accuracy cannot be guaranteed. Certain applications necessitate timekeeping, and so Mayfly is an important contribution, however it suffers the same problems as other task-based approaches.

2.4.3 Non-Volatile Processors

Beyond checkpointing and task-based approaches there is redesigning processors to be power-failure resilient. These systems typically incorporate NVFFs, and small but frequent checkpoints.

Wang et al. [118] present the first NVP [106]. It incorporates a non-volatile flip-flop design with parallel read and write as shown in Figure 2.11. A detailed explanation of the operation is given in [117]. Parallel read/write is significantly faster than serially backing up data as the checkpointing approaches do. They state a 7µs back-up and 3µs restore. Back-up is triggered by a voltage detection circuit. The energy consumed is 14.3pJ/bit back-up and 5pJ/bit restore.

Bartling et al. [11] present an even faster device with sleep/wake up times of <400ns. Energy cost of 2.2pJ/b sleep, 0.66pJ/b wake. The improvement is achieved by using non-volatile logic arrays to commit data to ferro-electric capacitors (Fe-Caps), instead of at an individual flip-flop level.

Sakimura et al. [94] state that the 8MHz clock frequency achieved by Bartling is insufficient for application requirements, and that the FeRAM has poor integration with standard CMOS. They instead propose a spintronics-based nonvolatile integrated circuit. This SpinRAM macro consists of 2 transistor memory cells and 1 magnetic-tunnel-junction memory cell. This MTJ gains memory persistence through the magnetisation of a 'free' layer

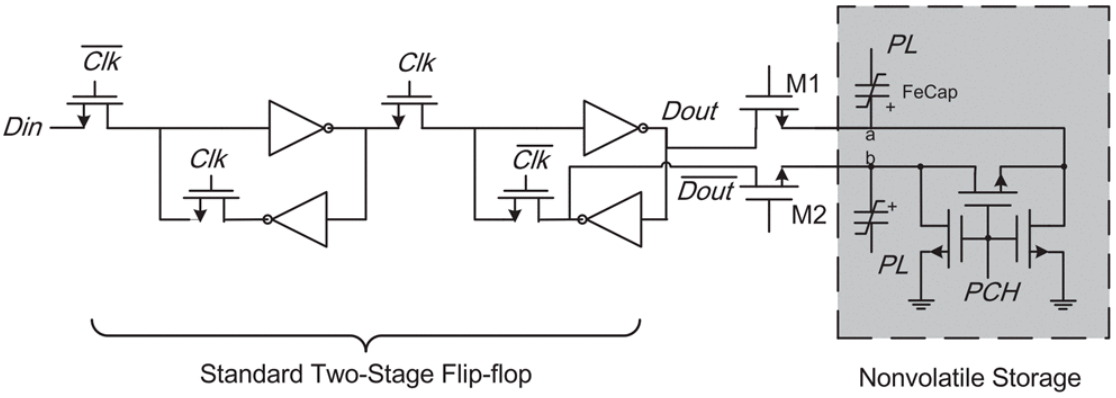


FIGURE 2.11: Non-volatile flip flop design taken from [118]

between a reference and sense layer. They achieve 20MHz operation with 120ns wakeup time.

Liu et al. [66] explain that a large parallel NVM backup generates a large in-rush current, inducing an IR drop on the processor, which deteriorates system stability. They instead propose a 3-layer distributed back-up control architecture. This remains 26.3% faster than a sequential method, without exceeding a maximum tolerable current limit.

Su et al. [105] concentrate on improving the wake-up and sleep time for the whole system, improving the voltage detection circuit, so that it does not become a bottleneck. They demonstrate a 46µs wake-up from power off and sleep within 14µs. This is an extension of the 7µs back-up, 3µs restore processor from wang [118].

2.4.4 Predictive Approaches

Some authors have attempted to include machine-learning, artificial intelligence, or other methods of prediction into their state-saving approaches. This is challenging due to the unpredictable supply, and the hardware constraints of the devices. Any performance improvements must be achieved without a significant impact on the power consumption since this is a key requirement of intermittent computing devices with their typically low energy availability. Many of these approaches include prediction so that scheduling can be added into the task-management. Prioritising mandatory tasks when energy is scarce, or maximising use of available energy by re-ordering tasks instead of running applications start to finish.

Flexi-check Singla et al. [100] create a checkpointing approach that uses a quadratically constrained linear program (QCLP) solver to determine an optimal checkpointing algorithm. This requires an ambient energy profile, and significant assumptions about the consistency of energy harvesters. The energy in a certain time frame is calculated and stored in a look-up table during a training phase, with a corresponding action. This can be execute,

wait or checkpoint. They show significant time and energy improvements for ideal sources in simulation, however the scheme relies on accurate prediction of the source characteristics. For real harvesters and systems it is not expected to yield such effective results, because the sources are less predictable than the ideal case.

Zygarde Islam et al. [52] also apply learning to energy availability, determining whether a source is unpredictable, predictable but generates insufficient energy, or predictable with medium confidence and generates sufficient energy. They then incorporate scheduling to improve the performance of a dynamic neural network based on the expected energy, executing the essential tasks, and adding optional tasks when more energy is available. The prediction depends on a measure of predictability of the supply which they name as the η-factor. This is estimated offline, and therefore could significantly vary in a deployed environment. Much like Flexi-check, Zygarde is expected to show greater performance in research than in a real EH environment.

Spendthrift Spendthrift from Ma et al [72] builds upon the idea in [9] of power-neutral computing by utilising a single neural network to apply machine learning to resource and frequency scaling techniques to better match the load of the system to the available supply. This technique when applied to an NVP improved forward progress 2.08x over static frequency and load.

2.4.5 Physical Deployment

Afanasov et al. [3] implement a intermittent computing approach relying solely on thermal and kinetic sources, harvesting energy from temperature gradients and structural vibrations. They opt for a task-based approach to implement sensing, data processing, and communication. They then set the device for a period of 2 years. The deployment demonstrated an improvement in the data quality over a battery approach, with the “fit and forget” benefits of EH powered device being truly demonstrated when they weren’t able to access the experiment for 3 months due to Covid-19, and the battery system failed, but the EH system continued. Figure 2.12 gives a useful breakdown of the energy consumption of the main elements

2.4.6 Summary

Intermittent computing enables IoT devices to operate directly from EH supplies, ensuring progress despite frequent power failure, or low-power sources. There are many schemes presented here, Table 2.5 gives an overview of key differences. It is clear from the literature that there is no expansive solution. Different schemes show benefits for different harvesters, and different applications.

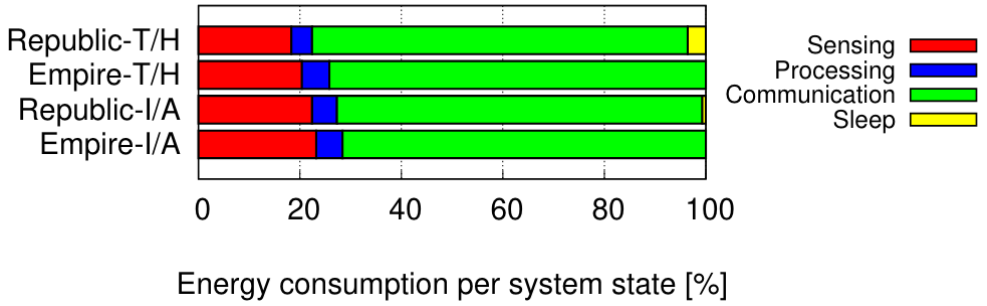


FIGURE 2.12: Breakdown of percentage energy consumption depending on functionality. Taken from [3]

Forward progress emerges as a key metric of comparison between schemes. It is most often demonstrated as a reduction in complete times of various bench-mark tests. It is increased by ensuring the time and energy overheads of each scheme are minimised and that power is used when it is available.

Reactive approaches such as Hibernus++ incur no additional work for the programmer, but have high energy and time overheads to save state. This is because the entire RAM is saved to NVM.

Task-based approaches put the burden on the programmer to define tasks that execute atomically. Tasks that are too large will not complete with low-power sources, or will incur high re-execution cost on power interruption. Tasks that are too small will incur frequent checkpoint and/or task transition energy and time costs. A system that can automate the task-boundary definitions well, would significantly improve this scheme.

Non-volatile processors (NVPs) are more similar to reactive approaches but can move data quickly, due to the parallel read/write capability. There remain problems with in-rush current peaks, that will only increase as the NVM size is expanded. These systems require more research before any commercial availability. The additional components for NVFFs are expensive both in terms of cost and on-chip footprint.

Overall there is a growing trend away from checkpointing approaches, and toward failure resilient atomic tasks. This is because of the favourable removal of overheads, however with high interruption frequencies task-based approaches may fail to make any progress with tasks consistently interrupted. This is a risk that reactive checkpointing approaches do not suffer from, generating interest in further improving checkpointing schemes. They also present the opportunity to contain all added complexity within interrupt routines triggered by hardware interrupts. This allows any software application to be run intermittently without any adjustments. Given the relatively low uptake, unfamiliarity in the IoT space and scarce deployment of intermittent computing schemes to-date, this reduced complexity should be a significant appeal.

Both schemes have merit, and in future combined approaches may prove to be the most effective.

TABLE 2.5: Comparison of leading intermittent computing approaches

System	Task Based	CP	Idempotency aware	Voltage Aware	Energy Aware	HW required
Mementos [89]		✓			✓	
QuickRecall [55]		✓		✓		✓
Hibernus [8]		✓			✓	✓
CTPL [2]		✓		✓		
DINO [67]	✓	✓				
Hibernus++ [7]		✓			✓	✓
Chain [25]	✓					
Ratchet [122]		✓	✓			
Clank [44]		✓	✓			✓
Alpaca [74]	✓		✓			
ENZYME [84]		✓		✓		✓
Flexi-check [100]		✓		✓	✓	✓

2.5 Evaluation Platforms

This thesis concentrates primarily on the improvement of reactive intermittent computing schemes, but to validate this improvement a test platform is required. There are several platforms that occur frequently such as the MSP430FR5739 experimenter board [111].

TABLE 2.6: Evaluation platforms used in leading intermittent computing research

Scheme	**Software Evaluated**
Mementos	MSPsim [33]
QuickRecall	MSP430FR5739
Hibernus	MSP430FR5739
Hibernus++	MSP430FR5739
DINO	WISP hardware platform [95] and MSP430FR5969
Ratchet	Thumbulator [45]
Chain	WISP5 energy-harvesting platform [95]
Alpaca	TI MSP430FR5969 microprocessor on MSPTS430RGZ48C project board and WISP
Flexi-check	Tejas simulator [96]
ENZYME	MSP430FR5969
Clank	Thumbulator and FPGA running Cortex-M0

Table 2.6 shows that the majority of intermittent computing approaches use an MSP430 experimenter board or adaption thereof. This is an obvious choice due to the low-power consumption and inclusion of an on-board FRAM memory. Chapters 4 and 5 evaluate the proposed schemes on an MSP430FR5739 and MATLAB mathematical simulation, however

Chapter 3 instead uses Fused simulation [102] and RTL simulation of a Cortex-M0+ and so these approaches are described in detail here. The MSP430FR5739 platform allows practical validation and comparison with existing research, whilst the inclusion of RTL simulation supports future research into transitioning intermittent computing to SoC platforms. The schemes presented in this thesis are intended to be compatible with a wide-range of software and hardware, further justifying the inclusion of a range of evaluation platforms.

2.5.1 MATLAB Mathematical Simulation

If the power consumption of a system and power input from a source can be characterised, then their interaction can be modelled using equations such as Equation 2.2. Power input can be modelled as an ideal source, however because real sources are used in the evaluation of schemes presented in this thesis, a more complex harvester model is desired to complement this.

PV cells can be modelled as using a single diode model as in Equation 2.3. Equivalent circuit is shown in Figure 2.13.

$$\mathrm{I} = \mathrm{I}_{pv} - \mathrm{I}_0(e^{q(\mathrm{V}+\mathrm{IR_S})/akT} - 1) - \frac{(\mathrm{V}+\mathrm{IR_S})}{\mathrm{R_P}} \tag{2.3}$$

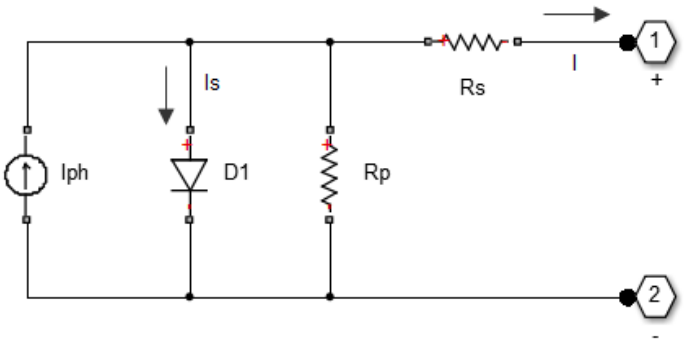


FIGURE 2.13: Equivalent circuit of a PV cell using a single diode model including series R_s and shunt R_p resistances taken from [81]

Data sheets do not include values for $\mathrm{R_S}$ and $\mathrm{R_P}$, so these must be determined. Selecting three points, short circuit (0, I_{sc}), MPP (V_{mp},I_{mp}), and open circuit (V_{oc} , 0), and calculating the remaining values in Equation 2.3 from the information provided in data sheets PV cells can be modelled from manufacturers specification. To determine R_s, R_p first calculate the photo generated current, I_{pv}, using Equation 2.4 to calculate the nominal value, and Equation 2.5 to account for the variation in temperature and intensity.

$$\mathrm{I}_{pv,n} = \frac{\mathrm{R}_p + \mathrm{R}_s}{\mathrm{R}_p}\mathrm{I}_{sc,n} \tag{2.4}$$

$$I_{pv} = (I_{pv,n} + K_I \Delta T)\frac{G}{G_n} \tag{2.5}$$

Calculating the dark saturation current is suggested by many authors to be done using Equation 2.6 [16], where $I_{0,n}$is given by Equation 2.7.

$$I_0 = I_{0,n}(\frac{T_n}{T})^3 exp[\frac{qE_g}{ak}(\frac{1}{T_n} - \frac{1}{T})] \tag{2.6}$$

$$I_{0,n} = \frac{I_{sc,n}}{exp(V_{oc,n}/aV_{th,n}) - 1} \tag{2.7}$$

However Equation 2.8, [114] gives a more accurate value over a wider range of temperatures by utilising K_V and K_I.

$$I_0 = \frac{I_{sc,n} + K_I \Delta T}{exp((V_{oc,n} + K_v \Delta T)/aV_{th}) - 1} \tag{2.8}$$

To calculate values for R_P and R_S an iterative approach can be used, adjusting incrementally until $P_{mp,m} = P_{mp,e} = V_{mp}I_{mp}$, since there is only one pair of resistances that achieve this. By setting $P_{mp,m} = P_{mp,e}$, Equation 2.9, is obtained which can be solved to give Equation 2.10.

$$P_{mp,m} = V_{mp}\{I_{pv} - I_0[exp(\frac{q}{kT}\frac{V_{mp} + R_s I_{mp}}{aN_S}) - 1] - \frac{V_{mp} + R_S I_{mp}}{R_p}\} = P_{mp,e} \tag{2.9}$$

$$R_p = V_{mp}(V_{mp} + I_{mp}R_S)/\{V_{mp}I_{pv} - V_{mp}I_0 exp[\frac{(V_{mp} + I_{mp}R_S)}{N_S a}\frac{q}{kT}] + V_{mp}I_0 - P_{mpp,e}\} \tag{2.10}$$

R_S should be slowly incremented starting at 0, and R_p recalculated until the peak power is equal to the experimental MPP. Equation 2.11 provides a good starting approximation for R_p.

$$R_p = \frac{V_{mp}}{I_{sc,n} - I_{mp}} - \frac{V_{oc,n} - V_{mp}}{I_{mp}} \tag{2.11}$$

The peak power is obtained by calculating I for a range of V values using Equation 2.3, and calculating P = I × V at each.[114]. Once series and shunt are known, the I-V and P-V characteristics of the cell can be determined from Equation 2.3. This quick-modelling allows initial assumptions to be evaluated and tested before any software is written, or intermittent computing schemes adapted.

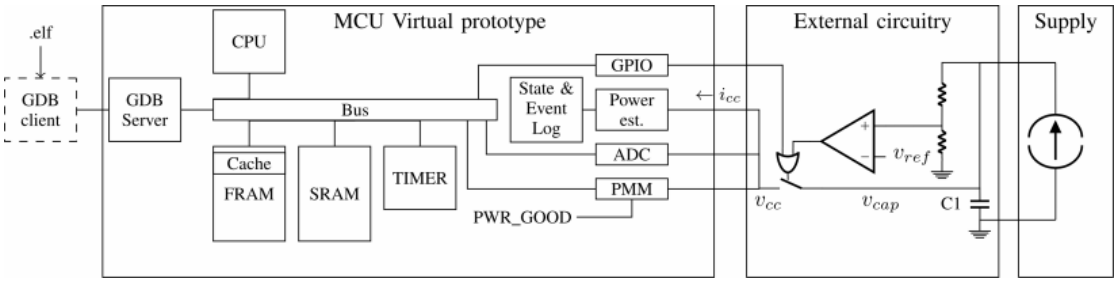


FIGURE 2.14: Model architecture of fused

2.5.2 Fused Simulation

Fused [102], is an open source full-system simulator. Fused models execution, power consumption, and power supply in a closed loop. It uses SystemC for digital and mixed-signal simulation to model a microcontroller and mixed signal circuitry, enabling hardware-software codesign.

Fused includes a high-level power modelling methodology, whereby events recorded during simulation are correlated to power measurements of real hardware to extract features for power modelling. Figure 2.14 shows the architecture of Fused. CPU models for the Cortex-M0 and the MSP430 instruction set architectures that execute unmodified binaries are both available.

It also has a power management module monitoring the supply voltage, and controlling the core voltage within the microcontroller. This is necessary to simulate the intermittent systems in this research. It is turned on when the supply voltage reaches approximately 1.88 V, and turned off when it drops below approximately 1.80 V. A reset to default values for all internal registers and VM occurs at this point. This enables intermittent computing to be simulated on Fused.

The power estimator computes the current consumption of the microcontroller based on the state and event counts within the current time step, and with a near-fixed voltage provided to the CPU, the power is easily obtained.

Fused was designed with intermittent computing in mind, and iclib [101] is already developed to work with Fused. Fused is able to give much more accurate power estimates than the MATLAB quick-modelling, but requires the source code to be completed and compiled is therefore a later-stage simulation tool. It allows adjustments and measurements to be made more quickly than a real-hardware platform or on-chip simulation.

2.5.3 RTL and Gate-level Simulation

Register transfer level (RTL) simulation is a model of a synchronous digital circuit design that focuses on the flow of signals between hardware registers and logical operations

performed on those signals. It is implemented in Verilog (or similar) for high-level circuit designs. From these gate-level netlist models can be derived that allow cycle-accurate power modelling.

The SoC design used in this research is based around a Cortex-M0+ CPU model from arm, with the design compiled by members of the arm research team I performed an internship with. This research contributes software and testbench to perform experiments on the SoC design.

One of the outputs available from arm's design is the 'TARMAC' log file recording time stamps using cycle counts for the contents of the bus, giving data and address as 8 hex digits representing 32 bits, and the instruction textually. Table 2.7 reproduced from [63] clarifies what is printed in the TARMAC log file.

TABLE 2.7: TARMAC log contents. TARMAC files use cycle counts as a timestamp, and print the information contained here

IT	Instruction taken (condition code passed) with disassembly.
IS	Instruction skipped (condition code failed) with disassembly.
R	Register update
B	Bus access (instruction, data, or system bus).
M	Memory access (duplicating bus accesses, all bus accesses are memory accesses).
E	Exception information and events.

2.6 Benchmarks

There are a range of benchmarks that are consistently found in the literature, most adapted from MiBench [39]. MiBench is composed of freely available source code and includes benchmarks that cover six categories including: Automotive and Industrial Control, Network, Security, Consumer Devices, Office Automation, and Telecommunications. Each of these is relevant to IoT devices, and MiBench is specifically targeted at embedded devices. Hicks [46] has redesigned a selection of the provided benchmarks for IoT devices, still representative of all six of these categories.

Theoretically these categories allow researchers to analyse their designs for a particular market segment, although intermittent computing research remains fairly broad, and few authors target specific industries, instead evaluating the correctness of their approach with some processing load provided by a benchmark.

Another set of benchmarks appropriate for intermittent computing devices, but not often used in evaluation are the BEEBs benchmarks produced by Pallister et al. [83]. BEEBs, unlike the majority of Mibench benchmarks, does not assume the presence of a host operating system or filesystem. Both sets represent a broad range of embedded areas, however BEEBs

selects benchmarks with the intention of incorporating a range of energy consumption characteristics. Each of the BEEBs benchmarks could be incorporated into IoT applications.

Chapter 3

The Impact of Memory in Targeting Reactive Intermittent SoC Designs

3.1 Introduction

As discussed in Chapter 2 there are two main intermittent computing approaches that can be implemented on commercial-off-the-shelf (COTS) hardware: checkpointing, and task-based. Choosing between these is typically a trade-off between overhead and complexity.

Task-based designs can be very effective if carefully designed, with the code broken into optimally sized tasks, but this is a challenging undertaking. It requires the programmer to adapt their entire code-base with new principles governing the careful handling of task-shared variables. This is prohibitive for many designers who are considering incorporating battery-less operation into their system.

Checkpointing approaches can be equally taxing on the programmer if the checkpoints are added to the code manually, however some schemes have now shifted this to the compiler [122]. Reactive intermittent computing, a form of checkpointing scheme, can be designed in a way that adds no complexity to the program, instrumenting the whole scheme within interrupt routines that can be included in compilation. They can however have the largest overhead per checkpoint, and require additional hardware in some cases.

Aside from complexity vs overhead, there are several other factors that must also be considered such as cost, memory requirement and processing overhead. Most schemes rely purely on software modification, however this itself incurs additional costs at design-time, and additional memory may need to be included to hold the code from the additional libraries.

The perceived complexity of intermittent computing systems will naturally limit the adoption of this technology, and the potential for reactive schemes to abstract the technical workings to interrupt routines makes the research more accessible both for research investigations and commercial adoption. This is a primary motivating factor in choosing to pursue reactive approaches as a focus of this thesis, however it also noted that other authors have quantitatively validated the increased performance of reactive schemes [91].

Reactive intermittent computing approaches can briefly be summarised as execution happens, until the power fails. This power failure is picked up by a voltage threshold detecting the falling voltage, and at this point all volatile data (i.e. registers and RAM) is saved into an NVM. When the power returns, this volatile information can be restored and execution resumes as if it had just been paused. The key relationship in this system is that data is moved between volatile and non-volatile memory. This transfer is also the primary overhead of a reactive intermittent computing system. Section 3.3 investigates this relationship, what exactly is contained within VM, and how this changes over the course of execution.

Whilst the intermittent systems implemented on experimenter boards that are common in the literature are useful for demonstrating the efficacy of these schemes, they do not allow this information to be obtained easily. Instead, a reactive scheme in a register transfer level (RTL) simulation of a Cortex-m0 processor is implemented, allowing detailed observation of the bus during execution. This scheme is described in Section 3.2.

This is of further interest because one of the motivations of intermittent computing is small mass, size, and cost. To manufacture these devices at scale and cheaply, System-on-Chip (SoC) implementations are a natural progression. This RTL implementation forms part of a further exploration and discussion of what is needed to realise these kinds of systems as the field of intermittent computing matures from proof-of-concept to commercially deployed devices in Section 3.5.

Finally, Section 3.4 further uses the large amount of data obtained from 72 BEEBs benchmarks [83] and a range of adapted MiBench [39] benchmarks to explore more than just the relationships in memory, but how the application as a whole interacts with a reactive intermittent computing scheme, and what improvements can be made from a deeper understanding of the kinds of applications running on these devices.

3.2 Implementing an Intermittent Computing Approach in RTL

To obtain the data required for the investigations in this chapter, 72 BEEBs benchmarks and several MiBench benchmarks were compiled for the arm Cortex-M0+. The Cortex-M0+ has the smallest footprint and lowest power requirement of the Cortex-M series of processors, making it an ideal candidate for intermittent computing. The MSP430 prevalent in

intermittent computing literature is a 16-bit architecture, whereas the M0+ is 32 bit resulting in higher performance as a result of faster operations, typically more powerful development tools, additional features, and better power efficiency. There is not currently a commercially available Cortex-M0+ experimenter board with integrated NVM (ignoring flash), unlike the MSP430FR series that incorporates FRAM. These were instead run on an RTL level simulation of an arm Cortex-M0+ SoC which was built as part of a larger arm research project developing a SoC for low-power wireless applications. RTL simulation has the benefit of observing exact bus transactions. RTL simulations from arm provide a text-based output called 'TARMAC' as a log file containing the contents of the bus at each time-stamp during computation. This allows detailed analysis of the behaviour of the system during the operation of the software. From this a summary of the memory-accesses can be extracted and plotted to do several things, including seeing which addresses would need to be saved in a checkpoint.

The benchmarks were compiled with a modified version of iclib [101] which is an allocated state intermittent computing approach, adapted for the RTL system. Allocated state means that instead of saving the entire address space, as Hibernus does, it only saves the registers and allocated regions e.g. '.data','.bss' and stack. The region of stack to save is determined at runtime according to the stack pointer. Much like Hibernus [8], iclib contains all of the additional software in separate libraries, and instruments the changes within interrupt routines. This allows the benchmarks to be compiled with a few additional .c, .S and .h files and a modified linker script.

A script was built to compile individual .hex files for all of the benchmarks at once. The only changes made to the benchmarks themselves were the inclusion of "__SEV();" commands which causes an event to be signalled in the processor that can be picked up on a waveform, similar to a print statement, to debug and verify expected behaviour. The SoC used for testing was part of an on-going research effort of a design with limited GPIOs and no UARTs to architect printfs.

There are two ways to run the simulation, intermittently and continuously. When the benchmarks are run continuously, i.e. with a stable power supply for the duration of the computation, the memory access pattern can be observed without the additional accesses related to checkpoints being saved and restored. Intermittently allows us to see how the intermittent computing approach affects and handles the applications. Section 3.4 uses data from the intermittent execution as does subsection 3.3.1. Section 3.3 primarily uses data from the continuous execution.

It is also possible to run a gate-level netlist simulation to obtain the power consumption associated with the behaviour observed in the RTL. This is also performed as part of this research.

3.2.1 Intermittent RTL Design

Whilst most intermittent computing research has been conducted on experimenter boards, being able to analyse what is happening cycle-by-cycle gives the opportunity for deeper insight. Because arm have not yet developed intermittent computing hardware designs or incorporated NVM into the project SoC RTL design I was working with, it was necessary to simulate non-volatile and volatile behaviour using only volatile RAM. In addition iclib [101], requires an interrupt signal to trigger a state-save. This was generated by feeding commands through the SPI.

The SoC has an SPI controller that recognises set instructions passed through SPI with associated address and data. A sequence of instructions can be written in a .txt file and passed through SPI as a form of testbench for the RTL design. This method would allow the design to be simulated on an FPGA, and triggered in the same way. Although not strictly a testbench, it is referred to as such in this chapter to distinguish from the software that is running on the RTL (containing both benchmarks and complied intermittent computing interrupt routines).

Figure 3.1 shows the relationship between the testbench SPI commands and the RTL with software that enabled the simulation of an intermittent environment.

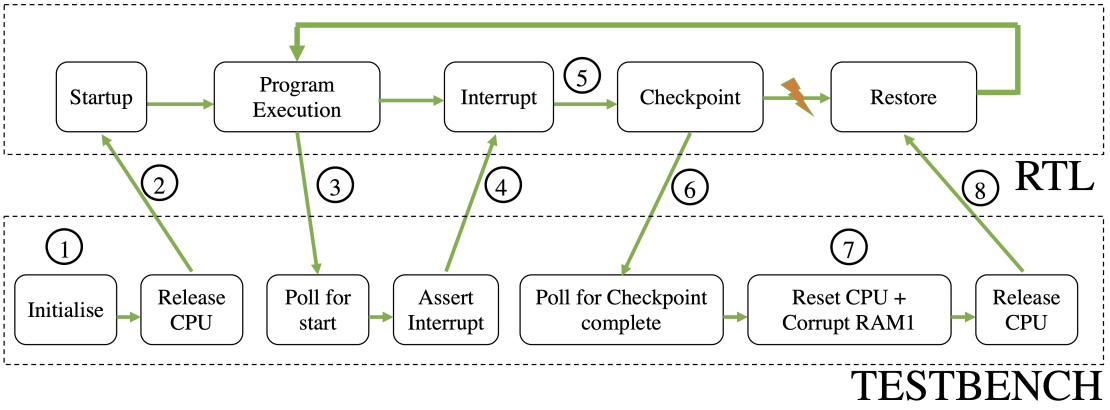


FIGURE 3.1: Register transfer level (RTL) implementation

The tasks listed in the dashed box labelled “RTL” are the applications compiled with the intermittent computing scheme running on the RTL simulation. The tasks listed within the dashed box labelled “testbench” are the additional commands contained within a .txt file that are fed one by one through SPI.

Figure 3.1 can be explained as:

1. CPU starts in a reset state whilst the SPI is initialised and the .hex file loaded into the memory through SPI.

2. Once this is complete the CPU reset is released and begins executing the program.

3. The testbench then polls a pin which is set to high in software when execution begins.
4. Several cycles later, the testbench initiates an interrupt as though a voltage drop has been sensed[1]
5. In the software, this triggers an interrupt routine which contains the state-save routine. This moves all of the registers, stack and allocated RAM into the NVM.[2]
6. Once the checkpoint routine is complete another pin is set, so that the testbench can recognise that the checkpoint is complete.
7. The CPU is reset by the testbench, clearing the registers and corrupting RAM1 with a write of 0xaa to all addresses to simulate power failure.
8. The CPU is released. The software start-up routine recognises the power failure, and initiates a restore, replacing the contents of RAM and the registers to continue program execution from the point of interruption.

Operating the RTL simulation with this testbench allows intermittent behaviour and the effect of falling voltage to be simulated without either of these needing to be simulated in the hardware. By repeating this testbench we're able to obtain intermittent execution data both from TARMAC log files, and using netlist simulation, also obtain power data, both for the total execution and checkpoint and restore. The following sections use this data to draw conclusions and gain insight for an intermittent SoC.

3.3 Volatile and Non-Volatile Memory in Intermittent Devices

In the introduction three areas of exploration were laid out:

- The relationship between volatile and non-volatile memory
- The contents of volatile memory
- How volatile memory varies over the course of execution

There are three main areas of interest within the memory. The initialised and uninitialised data memory (.data & .bss), the stack and the heap. These are the regions that are contained in the VM, and which must be saved during checkpointing. Each of these are explored using the results from the RTL simulation, and discussed in the following subsections.

[1] In reactive intermittent computing systems one pin of the MCU would typically be connected to a voltage detect circuit, which would trigger when the voltage drops below a certain threshold, here the pin is set high by the testbench.

[2] In this particular setup, this is simulated by another RAM block that is not reset.

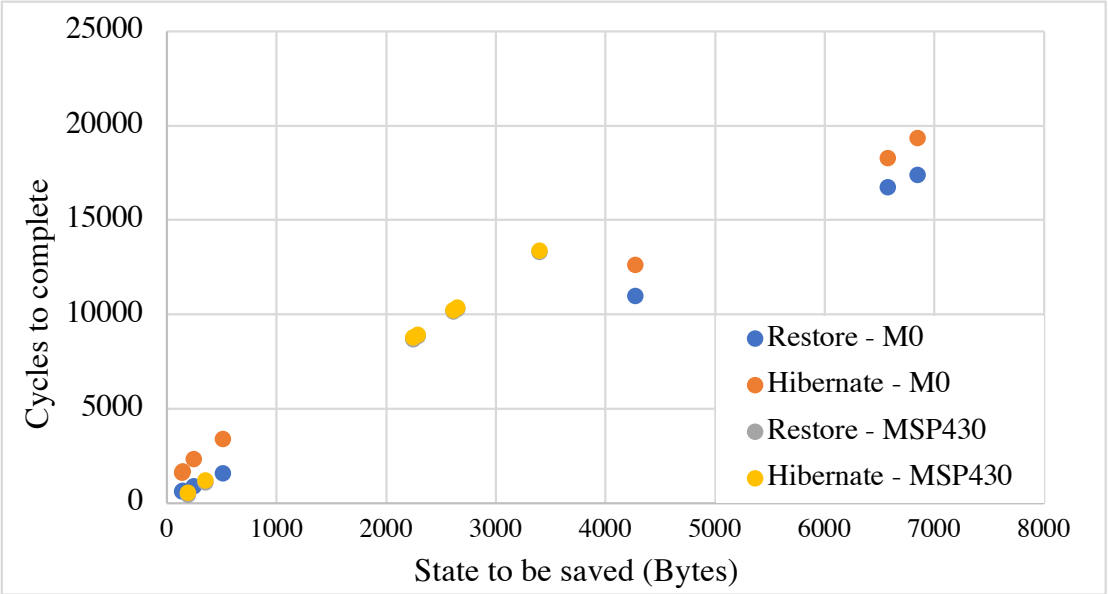
3.3.1 NVM vs VM in Design

Intermittent computing systems must take seriously the relationship between non-volatile and volatile memory use. In the most prevalent reactive schemes, every byte stored in VM must be moved across to NVM during each state-save. Reducing the number of bytes saved in VM will have a direct impact on improving the performance of checkpointing for schemes that save only the allocated regions of memory. To demonstrate this data is obtained from the RTL simulation, but two further simulations are also included: MSP430 Fused simulation and Cortex-M0+ Fused simulation. The RTL simulation of the Cortex-M0+ gives cycle-accurate information that should directly track to a real chip. However it also has no real world EH context, analog peripherals, capacitance etc... The Fused simulation on the other hand [102] is a an open source mixed-signal SystemC virtual prototype that simulates execution, power consumption, and power supply in a closed loop. It offers a range of debugging tools, and can execute binaries that can be deployed on real hardware. It also allows the modelling of external circuitry. Because there is more flexibility in the Fused model, it is setup to mirror the same characteristics implemented in the RTL for this experiment.

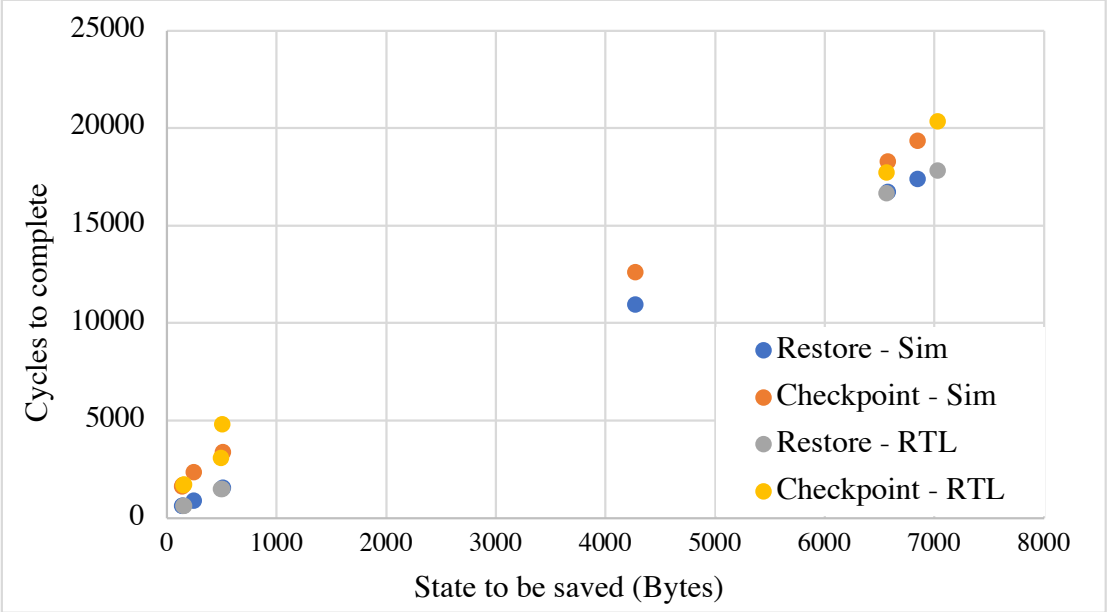
Figure 3.2 shows the relationship between the number of bytes to be saved (comprising of data, BSS and stack) and the cycles to complete a checkpoint. Registers are saved also, but the number of registers saved is fixed across all applications and does not vary. The applications included in order of state size: bc, counter, cem, nn-gru-cmsis, ar, aes, matmul, qrencode.

This demonstrates the relationship between state to be saved and cycles to complete a checkpoint or restore operation is directly proportional. It is beneficial therefore to ensure that the contents of VM are tightly controlled to reduce state save overheads. As an aside, this figure also demonstrates the close relationship between the Fused and RTL results, demonstrating for the first time the accuracy of the Fused simulation for the Cortex-M0 when compared to RTL.

Once the developer has optimised their code so that the VM footprint is as low as possible for the sake of checkpointing efficiency, they should also be careful with compiler optimisations. A compiler that optimises for execution speed or efficiency may improve the continuous execution performance, however to do so it may increase the VM allocation which would reduce the checkpointing performance. The checkpointing frequency will vary according to the energy harvesters characteristics, a point explored further in the following chapter, but this fact should be accounted for in design. For example, a source that creates frequent checkpoints due to a high degree of intermittency should prioritise the checkpointing performance, minimising the overheads, since this will have a more significant impact on the overall performance. Conversely, a source that only interrupts occasionally can instead prioritise continuous execution performance.



(A) M0 vs MSP430 Fused Simulation



(B) RTL vs Fused Simulation for CM0

FIGURE 3.2: Comparison of state transfer cycles for a number of benchmarks of differing size.

3.3.2 Trading Runtime Performance for Checkpointing Efficiency

If reducing the VM is such a priority, one suggestion could be to place all of the information into NVM, thereby eliminating the majority of the checkpointing overhead. This is implemented in QuickRecall [55], which utilises a unified FRAM memory. The RAM is replaced with FRAM and only the CPU registers must be saved. This greatly reduced the checkpoint overhead. When comparing the read/write costs of FRAM and other NVM technologies to SRAM however, it should be clear that unified memory approaches sacrifice runtime performance for checkpointing efficiency.

QuickRecall is the simplest implementation of a unified memory checkpointing scheme. The system transfers the program counter (PC), stack pointer (SP), status register (SR), and general purpose registers (GPRs) in response to a falling voltage. When the voltage recovers, the system restores these same registers. Whilst different programs use different GPRs, the system saves all of them during checkpointing. This smaller state save without the RAM is possible because the stack, .bss, .data and .text memory regions are all contained within the non-volatile FRAM as shown in Figure 3.3. This however means that each memory access during the entire execution occurs within the FRAM memory. FRAM has a significantly higher read/write energy and latency than SRAM. If the same unified memory approach were to be applied to another existing NVM technology, described further in Section 3.5 there would still be a significantly higher energy cost and latency. Other memories may also suffer from insufficient write endurance to sustain high access frequency over the long lifetimes anticipated for these devices.

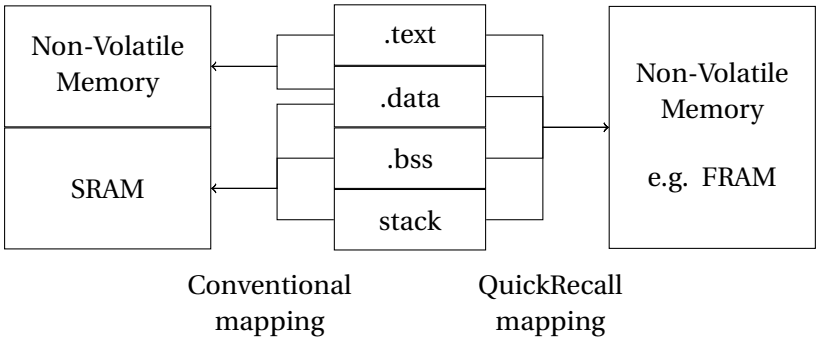


FIGURE 3.3: QuickRecall linker map compared to conventional system, adapted from [55]

For these reasons, the remainder of this research focuses on reactive schemes that keep some VM, so that the continuous performance is increased. The trade-off of increased checkpointing performance does not outweigh the overheads during execution as further illustrated by [91].

3.3.3 Static Contents of Volatile Memory

Returning to conventional mappings, when an application is compiled, data is distributed across the memory in an address space mapping comprising of the following sections: text or code segment (.txt), initialised data (.data), uninitialised data (.bss), stack and heap. The size and allocation of these sections can be obtained from the .map file generated by gcc. '.data' and '.bss' are the regions stored within the VM at runtime and as is shown in Figure 3.2 this has a direct impact on the cycle to complete a state save. The VM, therefore, is explored first. Figure 3.4 shows the size of these memory regions for a large range of BEEBs benchmarks compiled for the RTL implementation.

Huffbench through to miniz all have a significantly larger BSS region than the nearest benchmark. Further investigation reveals that this is due to an 8kB allocated memory region that is used for a software implementation of a heap. This heap will be further explored in subsection 3.3.4.2. Both data and BSS regions should be minimised to reduce the contents of state saves. Figure 3.5 shows examples of a plot of the read and write accesses against time and memory address during runtime obtained from the TARMAC file. Viewing the data in this way allows anomalies in the expected memory read/write patterns to be picked up.

Figure 3.5a gives a clear example of how the execution of a program can be observed through these plots. Bubblesort operates by swapping adjacent elements if they are in the wrong order. Gradually the number of variables needing to be read into memory and swapped declines, until the bubblesort is completed, at which point the variables can be written back into memory in the correct order.

In the case of Figure 3.5a the behaviour observed is what would be expected. However in Figure 3.5b there are a large range of memory addresses which are read and never written to. This might suggest that these are constant variables which could have been stored in NVM, instead defined as variables that have been allocated to the dynamic memory. Further investigation shows that in this example this is an array of data that could have been allocated to code memory but was defined as a variable array. If reading this data into the array from a sensor it is likely that this would reside in VM until it had been processed, so perhaps this was intentional from the benchmark designer. Figure 3.5c shows similar behaviour with a large set of constant data being stored in the VM, and being read but never written to. Viewing the data in this way allows mis-allocations like this to be picked up visually. For conventional embedded devices utilising VM unnecessarily may not have a negative impact on performance, but due to the way it increases overheads in checkpointing, it is relevant for intermittent computing platforms.

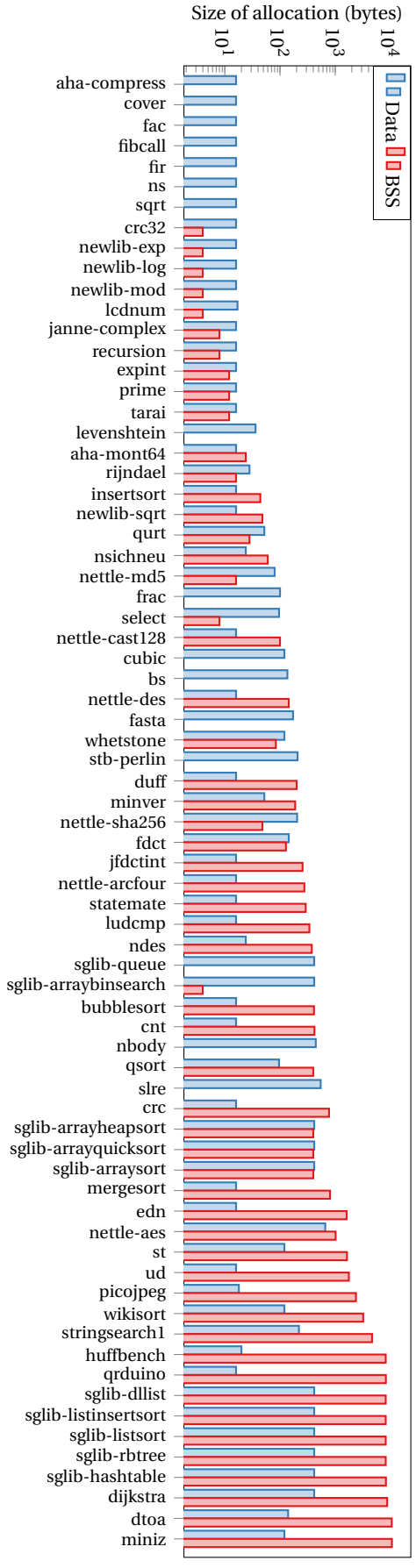
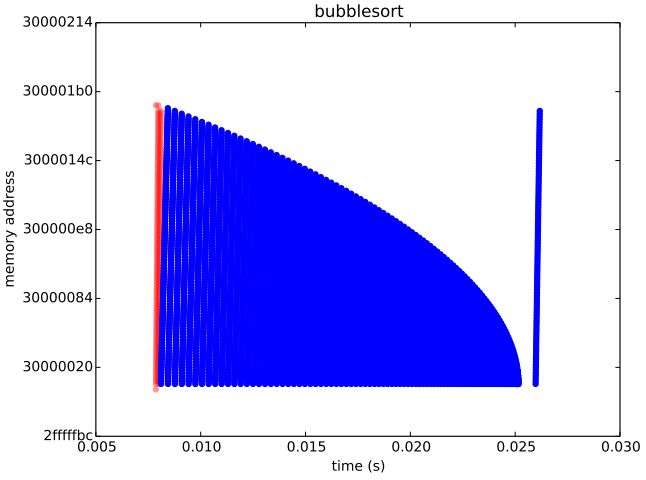
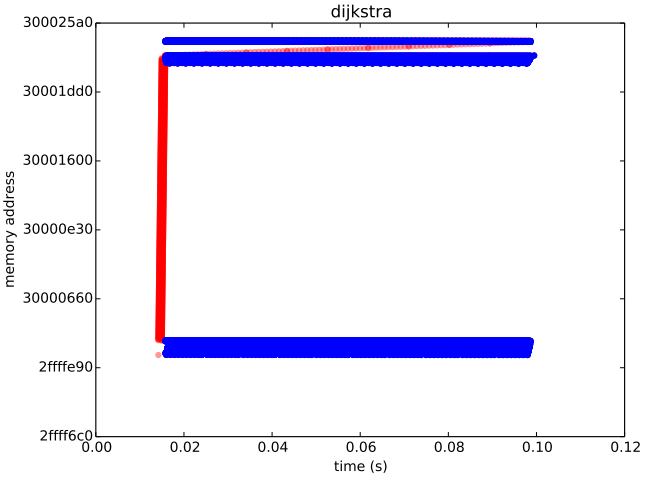


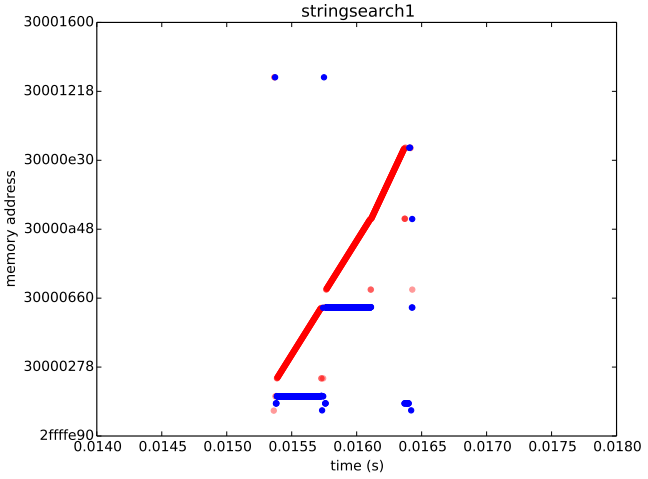
FIGURE 3.4: Data(left bars) and BSS (right bars) allocation across test programs, sorted according to total size.



(A)



(B)



(C)

FIGURE 3.5: A plot of memory reads and writes against time during execution of test programs

3.3.4 Volatile Memory Variation During Execution

The quantity of VM addresses in use is not fixed at compile time due to the variation in stack and heap utilisation. Not all embedded devices use a heap, but understanding the stack is crucial to determining how benchmarks affect VM allocation, and consequently checkpointing performance.

3.3.4.1 Stack

Whilst the data and BSS regions that must be saved during allocated state checkpointing are established at run-time, the stack is not. The stack grows and shrinks over the course of execution, resulting in larger and smaller checkpoints. The areas of interest for this investigation are how much the stack varies over the course of execution, and to what extent it affects the size of the checkpoint.

BEEBs benchmarks, with their wide-range of application styles also show great variation in the utilisation of the stack. There are three key observations from Figure 3.6.

1. Stack usage and maximum stack size are not correlated, meaning that a stack can be very active during runtime, but if properly released does not impact the size of the checkpoint greatly.
2. Some benchmarks, mostly those performing a sorting or searching function, cause the stack to grow to 1000s of bytes.
3. The percentage of the total checkpoint that comprises of the stack varies from 1-91%. This means that for some benchmarks the stack variation is particularly significant to checkpointing overhead.

Iclib [101] will save only the used portions of stack, however it does not adjust the voltage threshold. Currently there is no reactive approach that would dynamically account for this variation with a change to the voltage threshold and would instead allow insufficient energy for the memory transfer if the stack grows particularly large, or would have to conservatively allow for the largest possible stack size. Depending on the application, this value may not be predictable, as the quantity of data that must be sorted might be indeterminate.

This reveals that if the designer is running an application where there is a large amount of stack use, if properly released it will have little impact on the state-save overhead. However if the stack grows significantly then a scheme which handles this should be implemented. One such scheme is proposed in section 5.2, and a second suggestion is implementing a threshold that varies dynamically according to the current stack-size in order to closely track the changes in required energy when saving state, proposed as future work. This additional

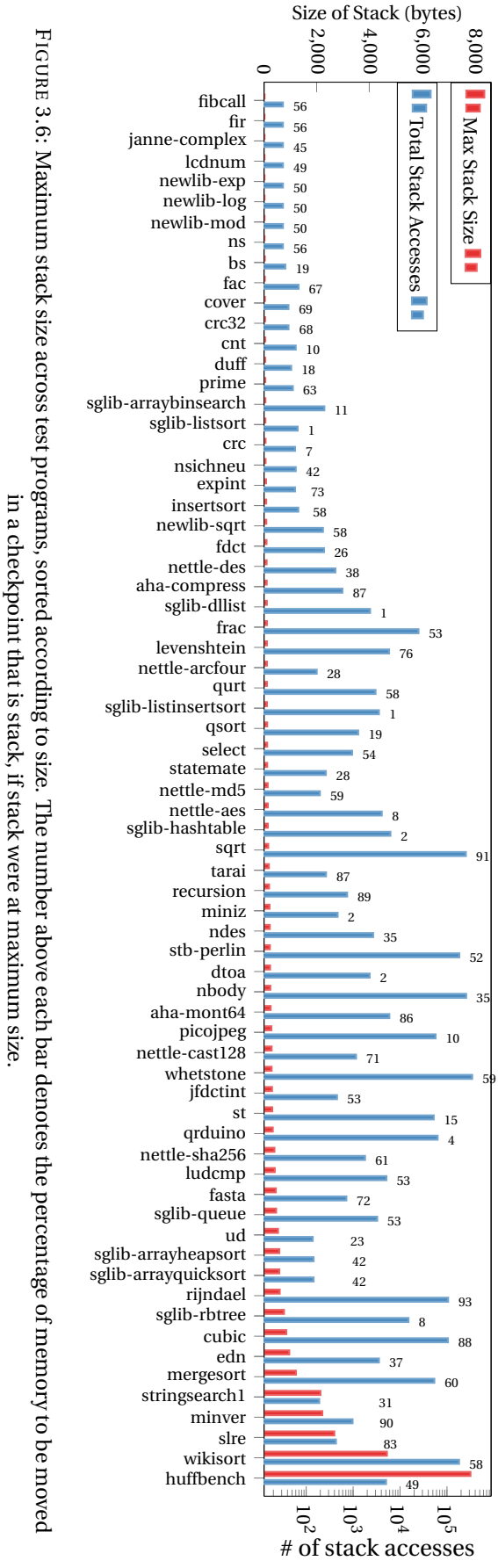


FIGURE 3.6: Maximum stack size across test programs, sorted according to size. The number above each bar denotes the percentage of memory to be moved in a checkpoint that is stack, if stack were at maximum size.

use case for dynamic thresholds motivates a scheme such as Hibernus++ [7] that has dynamic voltage thresholds implemented in later chapters.

3.3.4.2 Heap

The majority of intermittent computing schemes do not consider the heap in their design, discounting it as poor embedded system design, however with the increase in performance of 32 bit processors such as the Cortex-M0, many programmers are once again utilising heap. It provides improved flexibility and allows more complex languages and techniques to be employed. A method for saving only the allocated portions of heap is recommended by Bhatti et al. [17] but is not implemented in iclib or other reactive approaches.

Such a scheme greatly reduces the number of unnecessary state-save cycles, and would be necessary when implementing a heap into the system. As pointed out in subsection 3.3.3, in the case of the BEEBs benchmarks tested here, the heap is implemented in software and a portion of BSS is allocated to this. Existing schemes would recognise this 8kB region as necessary to include in each checkpoint since data, BSS, registers and stack are all saved, however the whole heap is not used, and so moving this data is unnecessary. This is a significant problem for the checkpointing performance when implementing these benchmarks intermittently.

3.3.5 Conclusion

Three simulations covering both the MSP430 and Cortex-M0+ demonstrate that the amount of VM used directly impacts the performance of reactive intermittent computing devices.

If the amount of data contained in VM can be reduced, then it is essential to checkpointing efficiency that this occurs. For example, variables that are constant should be declared ‘constant’ so that they are stored in the code memory in NVM rather than .bss in VM. At the very least it should be noted that techniques such as the software heap in the BEEBs benchmark suite, should not be used. During constant execution the additional VM allocation has little impact on the energy performance, however it significantly increases the checkpointing overhead in intermittent systems. Subsection 3.3.2 shows why a unified approach is not selected for this thesis. When the VM has been minimised, a more efficient transfer is necessary to further improving performance in this area. This motivates the Expedit scheme demonstrated and evaluated in Chapter 4.

3.4 Further Application Analysis

Up until now, application agnosticism has been included as a benefit of reactive intermittent computing systems [7]. The idea being that because the entire system state is

saved to NVM on every power-failure, the scheme can be implemented with any software. Unlike task-based systems where there need to be clear tasks defined, and careful handling of shared variables, reactive techniques allow the developer to compile their application with the additional libraries and interrupt routines without changing anything in the main application.

In this section whether or not the application can be ignored is explored using a large range of benchmarks, compiled and run on the intermittent SoC platform. The example of benchmarks that have a large stack growth has already been shown.

Subsection 3.4.1 first explores the kinds of applications implemented on intermittent schemes, and the evaluation criteria they use them to validate.

To complete the investigation of these applications, the intermittent RTL environment is modelled at the gate-level in a netlist simulation so that cycle accurate power information can be obtained. Using this it is demonstrated that a basic reactive intermittent computing scheme's performance can vary significantly from task-to-task.

3.4.1 Applications used in existing schemes

A key objective of the research in this thesis is to increase the performance of reactive intermittent computing schemes. Some opportunities for these gains lie in the overlooked characteristics of the software that is intended to run on these devices. Table 3.1 presents the range of applications and benchmarks used to evaluate research in the existing literature.

Scheme	**Software Evaluated**
Mementos	rsa64, crc, adc sampling
QuickRecall	rsa, crc, adc sampling
Hibernus	FFT
Hibernus++	FFT
DINO	AR, DS, MIDI interface
Ratchet	RSA, CRC, FFT, sha, picojpeg, stringsearch, dijkstra, basicmath
Chain	AR, CEM, RSA, CF
Alpaca	AR, CEM, RSA, CF, Blowfish, Bitcount
Flexi-check	AR, CEM, RSA, CF, print
ENZYME	MSP430 Benchmark Suite
Clank	MiBench2 IoT benchmark suite

TABLE 3.1: Applications evaluated in leading intermittent computing research

Ratchet, Clank and ENZYME are the only papers to use a significant number of benchmarks, many of the other schemes use four or fewer, and the reactive intermittent computing schemes notably use only a single benchmark for their evaluation citing the application agnosticism described earlier.

One interesting observation from the comparison of these approaches is that the reasoning and purpose of the evaluation varies from simply verifying correct behaviour and evaluating the scheme to demonstrating the performance benefits compared to other schemes, to exercising a range of computation, sensing and storage tasks.

The application choice doesn't seem to correlate strongly with the evaluation intentions, and the choice of schemes to compare to varies considerably with few clear reasons given to explain the choice.

To improve the evaluation of the efficacy and performance of novel intermittent computing schemes, a designated set of applications that correspond to the proposed use-cases for these types of devices and stress all of the computation, storage and sensing components that are present within the system should be developed. This is a suggestion for further work.

3.4.2 Reactive Overheads Compared with Task Completion

The overheads of an intermittent computing scheme such as Hibernus or Hibernus++ [8, 7] are consistent each time. This is because the entire address space is saved on each checkpoint. This makes it very simple to implement, but removes a significant area of potential saving. Allocated state approaches, which only save the regions of memory used for computation, such as suggested by Bhatti and Sliper [17, 101], reduce the overheads but the size varies from application to application. It will also vary as the execution proceeds due to changes in the size of the stack (and heap if implemented).

Table 3.2 gives the total energy used to checkpoint and restore mid-execution for a range of benchmarks. This data is obtained from the gate-level netlist simulation, that is a further extension of the RTL implementation described earlier. Because the netlist simulation gives power data for the entire SoC, most of which was unoptimised and not correctly full power-gated at the time of these experiments, only the power for the Cortex-M0+, RAMs and buses are taken, ignoring the rest of the chip.

TABLE 3.2: Power Analysis of several benchmarks from Cortex-m0+ netlist simulation

	Total Energy (µJ)			Cost compared to completing benchmark		
	Checkpoint	Restore	Continuous	Checkpoint	Restore	Total
aes	1.1	0.93	115.6	0.95%	0.81%	1.76%
ar	0.277	0.14	101.6	0.27%	0.13%	0.40%
bc	0.165	0.061	1.9	8.71%	3.25%	11.96%
cem	1.6	1.4	4.8	32.66%	29.57%	62.27%
counter	0.14	0.053	2.7	5.01%	1.95%	6.96%
nn-gru-cmsis	1.7	1.5	14.2	11.84%	10.49%	22.33%
qrencode	0.20	0.075	155.6	0.13%	0.05%	0.18%

cold-chain equipment monitoring (CEM) logs and periodically LZW-compresses temperature sensor data. For a benchmark like CEM which has a large input dictionary and

block size, but relatively low execution time, the size of the checkpoint is large compared to the completion time. This results in the combined overhead being equal to almost 2/3 of the energy required to complete the task on every checkpoint. CEM interrupted just twice during execution would increase total energy consumption by 125%, whereas qrencode would only increase 0.36% for the same number of interruptions.

Qrencode (QR encode), has a low total VM footprint, but a high execution cost. This results in the overheads making up a small percentage of the total completion time for the task. For software like this, many checkpoints still result in a reasonable overhead to complete the task, unlike if CEM were interrupted multiple times.

In reactive intermittent computing schemes, the checkpoint and restore overheads are the only consideration and have a large impact on the forward progress. For task-based approaches the re-execution cost can have an even more significant impact than the checkpointing overheads. More task boundaries would yield lower checkpoint size, but much more frequent checkpointing. Therefore a similar comparison would not yield useful observation in the same way.

To explore the knock-on impact of this variation in checkpointing overhead, the necessary capacitance ($\sum C$) can be calculated to successfully save a checkpoint using Equation 3.1 adapted from Equation 2.2 as shown in Table 3.3.

$$\sum C = \frac{2 * E_{\delta}}{(V_t^2 - V_{min}^2)} \tag{3.1}$$

TABLE 3.3: Capacitance needed for several benchmarks based on energy consumed during checkpointing

	Capacitance Needed (µF)	
	1V->0.9V drop	1.4V->0.8V drop
aes	11.6	1.7
ar	2.9	0.42
bc	1.7	0.25
cem	16.8	2.4
counter	1.4	0.21
nn-gru-cmsis	17.9	2.6
qrencode	2.1	0.30

Although 16 µF is a reasonable expectation for the decoupling capacitance of an experimenter board [8], that is a significant additional capacitance for an SoC. These energy figures were obtained using the simulated RAM, so if instead the technologies listed later in the chapter in section 3.5.2 are used, the numbers would be even greater as shown in Table 3.4. From the power analysis RAM0 (the simulated RAM as NVM) consumes approximately 60% of the total power during state saving. For each benchmark the power consumption of RAM0 is increased by 1.97x, 2.82x and 1.55x for FRAM, STT-MRAM and ReRAM respectively.

TABLE 3.4: Capacitance required to correctly checkpoint for a range of NVM technologies

	Total Energy to save state (µJ)				Capacitance for 1V->0.9V drop				Capacitance for 1.4V->0.8V drop			
	SRAM	FRAM	STT-MRAM	ReRAM	SRAM	FRAM	STT-MRAM	ReRAM	SRAM	FRAM	STT-MRAM	ReRAM
aes	1.10	1.79	2.39	1.49	11.59	18.86	25.17	15.70	1.67	2.71	3.62	2.26
ar	0.28	0.41	0.53	0.35	2.92	4.33	5.57	3.72	0.42	0.62	0.80	0.54
bc	0.16	0.24	0.31	0.21	1.73	2.57	3.30	2.21	0.25	0.37	0.48	0.32
cem	1.57	2.55	3.40	2.12	16.53	26.85	35.82	22.36	2.38	3.87	5.16	3.22
counter	0.14	0.22	0.30	0.18	1.44	2.33	3.11	1.94	0.21	0.34	0.45	0.28
nn-gru-cmsis	1.68	2.73	3.65	2.28	17.73	28.78	38.39	23.97	2.55	4.14	5.53	3.45
qrencode	0.20	0.32	0.43	0.27	2.07	3.37	4.50	2.81	0.30	0.49	0.65	0.40

For STT-MRAM, the capacitance would need to be up to 38 µF which would be prohibitive for a small SoC.

Although application agnosticism is okay for experimenter boards, and all of these programs will correctly execute given enough capacitance, as the field progresses to SoC designs (which have the small mass, size and cost required) the application must not be entirely neglected at the point of design. Applications that have a large quantity of volatile data will take significantly longer to checkpoint. This becomes even more pronounced when incorporating existing NVM technologies. This will take the capacitance beyond a reasonable level in most cases. Furthermore, some applications that are 2/3 through execution could be completed in the time/energy it would take to save a single checkpoint. This motivates an idea such as push-through, described in section 5.2.

3.5 System-on-Chip Intermittent Computing

The majority of intermittent computing strategies are implemented on COTS hardware which is often not as energy efficient, or cheap to produce as dedicated hardware or mass manufactured SoC designs. As the field of intermittent computing progresses towards commercialisation and mass deployment in the pursuit of a trillion IoT devices [107], SoC must be considered as an effective solution.

To realise these designs additional factors must be considered such as the reduced capacitance as mentioned in the previous chapter, and the type of NVM used. Although these are constraints, there also exist opportunities. Being able to design the SoC all the way down to the architecture of the processor allows specific optimisation that is not possible on COTS hardware.

3.5.1 RTL Implementation Error

The RTL implementation of iclib described and tested within this chapter is suitable for inclusion in future SoC designs, but porting the codebase from simulation to hardware simulation was not without setbacks. This particular error motivates one such architecture change as suggested above.

The error in the implementation and port of iclib, that is worth mentioning, relates to the Interrupt Program Status Register (IPSR). To begin, it is important to discuss exactly how the intermittent checkpoint and restore routines operate. Figure 3.7 shows the checkpoint routine and Figure 3.8 the restore routine. Both have relevant elements of the arm architecture described.

One of the key advantages of reactive schemes is that the checkpointing routine can be held within an interrupt routine, and it is within this interrupt routine that the final state of the

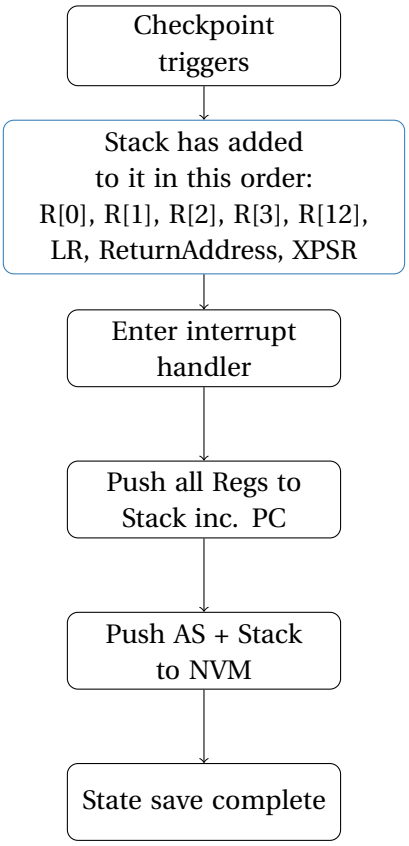


FIGURE 3.7: Checkpoint routine used in RTL implementation. Blue box represents arm architecture for interrupt handling

program counter and link register must be saved to NVM. This means that when the state is restored, it is not to the point of execution, but to the interrupt routine.

In Fused modelling, this didn't create any issues as the end of the routine is reached immediately, and the simulated architecture restores the program counter from the link register. In reality, this step is preceded in the ARMv6-m architecture by a check of the IPSR, which wasn't implemented in Fused, to ensure there are no faults. When resetting the MCU the IPSR is cleared, but, although the value is stored in NVM, IPSR is a protected register and cannot be written to, so the pop command is ignored. A fault is generated when the interrupt routine concludes [65].

To overcome this, assembly code is added to the restore routine to emulate the architectural interrupt exit without checking the IPSR, this returns the program to execution manually instead of allowing the program to reach the end of the interrupt routine and letting the architecture handle it.

This small observation opens up an area of further investigation in fault tolerance and architectural changes. Traditionally architectures have been created to detect faults and cease execution on their discovery. Intermittent computing clearly expects faults resulting from the sudden-loss of power, but if the specific faults can instead be tolerated knowing the

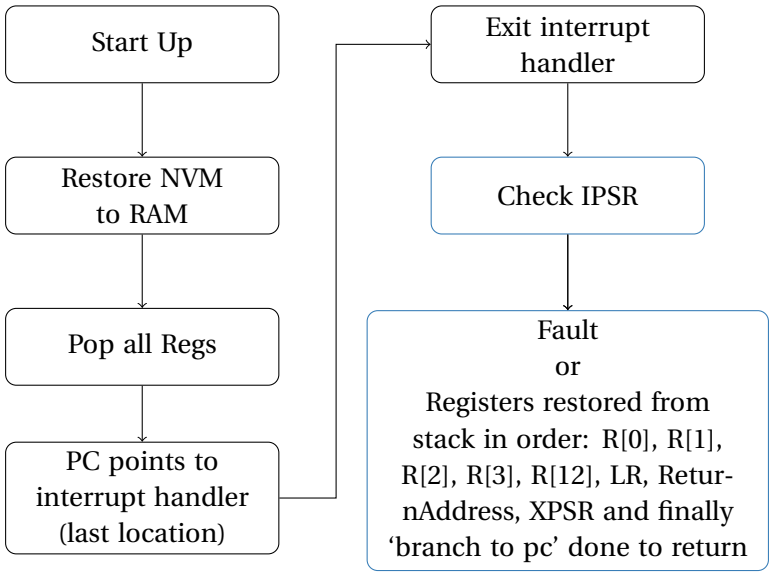


FIGURE 3.8: Restore routine used in RTL implementation. Blue box represents arm architecture for interrupt handling

source is the allowed intermittency, or further still, checkpoint and restore behaviour being incorporated into the architecture itself, then intermittent MCUs could be realised. These would differ from existing NVPs because the CPU and registers are all volatile, however the state-save and restore behaviour is handled by the architecture and specific faults tolerated. This level of custom application specific integrated circuits is beyond the scope of the work presented in this Thesis, but provides interesting future work.

3.5.2 Selecting the Correct Non-Volatile Memory

All embedded systems require both volatile and non-volatile memory to perform computation, and retain important data. The earliest intermittent computing approaches, such as Mementos [89], relied on flash memory for checkpointing. This had a significant impact on the efficiency of the memory transfer, and consequently Mementos was careful to limit the size of its checkpoints. The emergence and subsequent proliferation of faster, more efficient byte-addressable NVM technologies, such as FRAM have made intermittent computing much more efficient, however schemes such as Hibernus [8] limit the benefit by saving unused sections of the RAM. Other schemes, such as QuickRecall [55] opt for a unified memory approach, entirely removing SRAM in favour of minimal data transfer for checkpointing system state. VM accesses occur frequently in embedded computations, constituting up to 31% of memory accesses as shown in Figure 3.9. FRAM with an endurance of 10^{15} is able to cope with this increase, however other NVMs have a much lower endurance. In addition to this, writing to NVM can be orders of magnitude more expensive than SRAM both in terms of latency and energy consumption.

FRAM is currently the most commonly used technology in the literature, due to its availability on COTS experimenter platforms, as these devices move from proof-of-concept to fully designed SoCs memory technology will be an important choice. This section presents an overview of some of the NVM technology choices available, either commercially or in research, and the considerations relevant to intermittent computing. Of most relevance to the work of this thesis is the energy consumption, although the other comparisons motivate future research directions.

Table 3.5 compares key emerging NVM technologies against four categories with importance to intermittent computing, these aspects are discussed in more detail in the following subsections. The data given is a representative example, not intended to demonstrate an average or the state-of-the-art, as this is beyond the scope of the research. Data for STT-MRAM and ReRAM is taken from foundry data-sheets to indicate what performance is commercially available at the time of writing.

TABLE 3.5: Comparison of NVM memory technologies for IoT devices (* signifies simulation data only)

Technology	Write/Read Energy (per bit)	Write/Read Time (per bit)	COTS	Endurance (Write cycles)
NAND Flash[37]	470 pJ/46 pJ	200 µs/25.2 µs	Y	10^5
SRAM[4]	355 pJ/587 pJ	2.2 ns/2.1 ns	Y	Unlimited
FRAM[51]	1.4 nJ/1.4 nJ	120 ns/120 ns	Y	10^{15}
STT-MRAM[24]	2 nJ/34 pJ	250 ns/10 ns	Y	10^5
SOT-MRAM*[4]	334 pJ/247 pJ	1.4 ns/1.1 ns	N	$>10^{15}$
ReRAM[53]	1.1 nJ/525 fJ	10 µs/5 ns	Y	10^5
PCM*[62]	13.5 pJ/2 pJ	150 ns/48 ns	N	10^7

3.5.3 Energy Consumption

Research into intermittent computing systems aims to maximise the forward progress for a given supply. The energy access costs of the NVM used, either for processing or checkpointing, will have a significant impact on the performance of the device. All of the commercially available technologies have a greater cost than SRAM. SOT-MRAM on the other hand is approaching the cost of SRAM, and if it is able to satisfy the other requirements of intermittent systems will become an extremely promising solution. However if SOT-MRAM does achieve the projected energy consumption, being an NVM with SRAM like consumption, its disruption would far exceed ultra-constrained devices, but could reshape the entire field of embedded computing and even conventional computing.

3.5.4 Latency

The access times of NVM technologies vary by orders of magnitude, but there are scenarios where this will not affect the performance of the system. Table 3.6 shows the number of

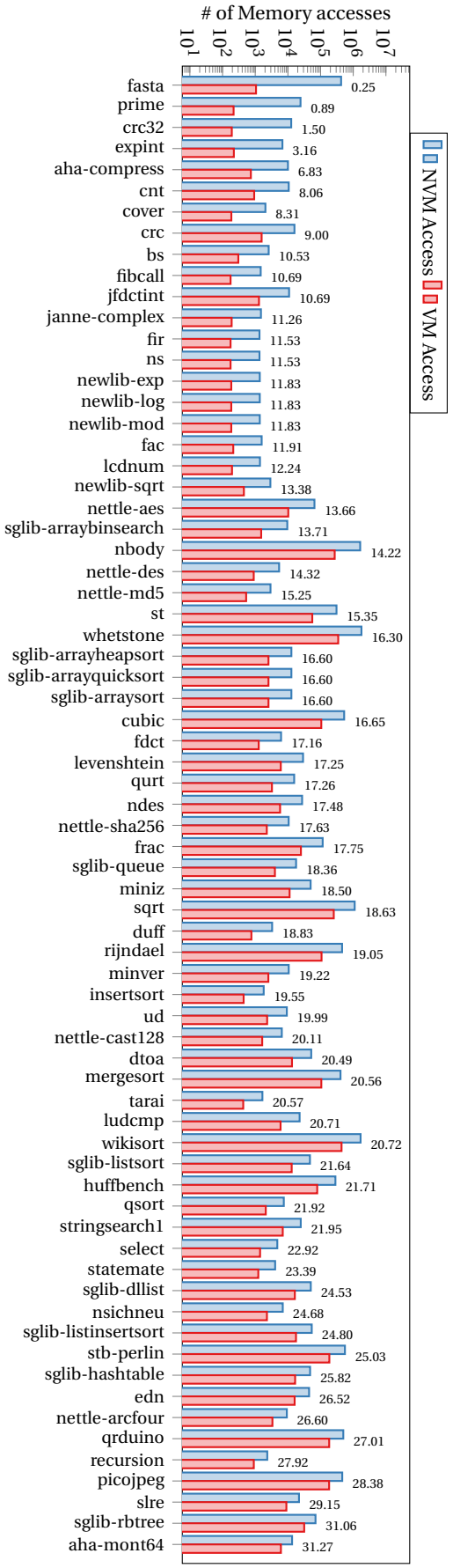


FIGURE 3.9: NVM (left bars) and VM (right bars) memory accesses across test programs, sorted according to VM accesses relative to NVM accesses. The number above each bar denotes the percentage of memory accesses that occur in VM.

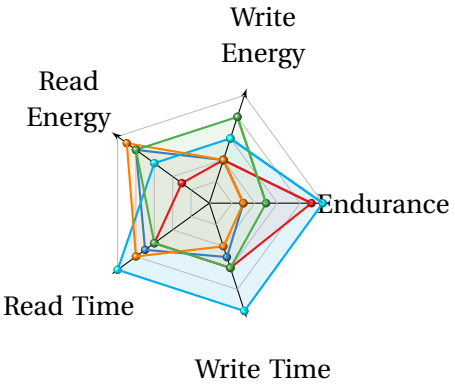


FIGURE 3.10: Radar plot of NVM characteristics. Line colors correspond to the colors in Table 3.5

cycles needed to access NVM at 24MHz, indicating how the latency of these technologies would negatively impact performance. For some of these technologies, non-blocking writes will mask the increased latency, however, when saving/restoring state, NVM latency will have a direct impact due to the sequential memory accesses.

TABLE 3.6: The number of clock cycles needed for NVM access at 24MHz CPU clock frequency.

Technology	Write/Read Time (per bit)	Clock cycles (Write)	Clock cycles (Read)
NAND Flash	200µs/25.2µs	4800	605
SRAM	2.2ns/2.1ns	<<1	<<1
FRAM	120ns/120ns	2.9	2.9
STT-MRAM	250ns/10ns	6	0.24
SOT-MRAM	1.4ns/1.1ns	0.03	0.03
ReRAM	10µs/5ns	240	0.12
PCM	150ns/48ns	3.6	1.15

3.5.5 Endurance

Intermittent schemes often use NVM for checkpointing in ways that would exhaust some NVM cells within a matter of months or even days. In Hibernus++ [7], for example, the authors demonstrate their scheme with a Seiko watch power trace, interrupting every 0.4s. With FRAM, this scheme could checkpoint successfully for 6.34 million years (ignoring all other factors), but STT-MRAM and ReRAM with 10^5 endurance would last only 4.63 days.

For devices to be truly long-running, NVM technologies with greater write endurance must be used, or intermittent computing schemes must consider checkpoints as a finite resource that should be more carefully allocated. Schemes such as Hibernus++, which save the entire RAM contents to NVM, must be replaced by schemes such as allocated/managed state [101] which greatly reduce the number of writes required.

As shown in Figure 3.9, a unified memory approach could increase NVM accesses during computation by up to 31%. FRAM with an endurance of 10^{15} is able to cope with this increase, however other NVMs have a much lower endurance. This further adds to the poor suitability of unified memory approaches for many NVM technologies as a result of endurance limitations. Many task-based intermittent computing approaches depend on unified memory models, so this could present a significant challenge, unless endurance can be improved.

Wear-levelling Currently intermittent computing systems take the simplest approach of saving and restoring to the same region of NVM on every checkpoint. For these long-running systems, it may be the NVM that wears out first and results in the breakdown of the device. If it is necessary to use an NVM that has a lower than desired endurance, it would be possible to implement a wear-levelling extension to the checkpointing routine. The idea being that the address space where the checkpoint is saved to is incremented each time it occurs. This would more evenly wear the NVM. It may be that the NVM comes with a wear-levelling controller, but this should be an important consideration when opting for the relevant technology.

3.5.6 Comparison

To conclude, the latency of all of these technologies is acceptable for the perfomance of these low end devices. Intermittent computing schemes require a good endurance, and this would rule out technologies like ReRAM in their current form. The key limitations of FRAM, which has been the most popular technology to date, would be its proprietary nature and the lack of scalability. FRAM will not scale below 130nm, whereas the STT-MRAM technology presented is 22nm. Whilst decreased energy consumption is desirable, and the hope is that these technologies continue to improve, my conclusion would be that STT-MRAM is the most suitable technology for intermittent computing at the time of writing. It is hoped that arm experimenter boards featuring this technology will become available in the near future for further research, though off-chip STT-MRAM options are currently available.

3.6 Summary and Discussion

A reactive intermittent computing approach has been designed and implemented for RTL implementation using a Cortex-M series SoC, which is prevalent in IoT devices. It has an exceptionally small silicon area, low power and minimal code footprint. However, the same techniques could be applied to other low-power SoCs. This would require software

adjustments specific to the architecture, but the implementation is not dependent on the arm architecture.

From this Cortex-M0+ RTL implementation a large amount of data has been gathered from 80 benchmarks, both for continuous and intermittent execution. Applications have been demonstrated to have VM allocation that varies significantly from application to application, as well as during execution, particularly due to the stack. This directly impacts the cost of saving state as shown in Figure 3.2. The cost of saving state is determined through gate-level simulation and compared with the continuous execution energy consumption. This demonstrated that some tasks have a very large overhead compared to completion, up to 66%, and the requirement for a large capacitance.

Furthermore, as these systems target SoC implementations, they will need to select the appropriate NVM technology, and the commercially available technologies are shown to be significantly more energy intensive than volatile RAM. All of this should begin to indicate that to have an optimal intermittent computing scheme, the application used and usage of memory cannot be counted as insignificant. Memory use varies significantly, not just program to program, but also mid-execution as the stack shrinks and grows. As a result application agnosticism is not recommended in the design of deployed in intermittent computing devices.

However, in research it is still possible to make improvements to intermittent computing without developing specific intermittent applications by instead concentrating on reducing the overheads of saving state and restoring. The need for this is demonstrated in section 3.4.2 where it is shown that if overheads are not reduced, the capacitance required is too great and the relative overhead compared with completion too high for some applications.

Reducing these overheads becomes the priority of the remaining research of this thesis, in expectation that improvements can be made over the state-of-the-art relevant to all applications. In the following chapters this is achieved by directly targeting the nature and frequency of state-saves. In this way, even if the overall schemes cannot be made application agnostic, there is scope for improvements in forward progress without diving into specific software development changes. Furthermore, having demonstrated the potential for intermittent computing to move to SoC implementation in this chapter, the proposed approaches are designed to be generic enough that they can improve upon and be implemented in both commercially available and SoC devices. This allows comparison with the existing research, and opportunity for future implementations.

Chapter 4

Enhanced Checkpointing for Intermittent High-Power Sources

4.1 Introduction

In Chapter 3 it was established that the power consumption varies according to the application running on the intermittent computing device, and the variation can be too significant to ignore. Where the 'power out', was discussed in the previous chapter, this chapter will begin by exploring the 'power in' by comparing the types of EH used in prior research, and exploring characteristics which can be generalised across harvesters and exploited for improved performance.

The conclusion in the previous chapter made a case for improvements that do not fixate on factors which vary according to the application that is running on it. Time and energy overheads during checkpointing are one of the primary concerns for reactive schemes, and these are relevant to all applications. This chapter concentrates on reducing this overhead. The impact of energy consumption during checkpointing is demonstrated and the solution of disabling system resources during checkpointing is presented. The most significant of these being disabling the CPU through the Expedit scheme, which when practically validated was shown to reduce the time and energy overheads of saving state by 83.6% and 85.6% respectively, giving up to a 84.4% improvement in completion time over a state-of-the-art approach, Hibernus++ [7].

4.2 Energy Sources Used in Existing Validations

There are several sources of harvested energy that can be exploited for IoT devices to run for long periods of time. Each of them have different characteristics that will affect the performance of the intermittent computing scheme that is powered by them. In addition to

this, different applications have been shown to have very different consumption characteristics. For example accessing an NVM location has an energy cost which is orders of magnitude higher than a processing clock cycle of the CPU, and therefore memory-access heavy applications will have a more significant run-time energy consumption. Figure 4.1 further demonstrates the variation in the average CPU power consumption across benchmarks (rounded to the nearest μW), with rijndael drawing 37% more power than the lowest consuming benchmarks.

With high-power output harvesters, this may not be problematic, since the power-in could exceed the power-consumed, however with low-power supplies that rely on charging a capacitor to reach a defined operating voltage, this will more severely limit the forward progress. Therefore it is important to understand the incoming power supply and outgoing power consumption to design schemes that not only demonstrate enhanced performance over the state of the art, but will continue to do so when deployed in a realistic scenario.

Some intermittent computing approaches directly target certain power conditions, such as ENZYME requiring low-power sources to be effective. However, most of the intermittent computing literature target their system to cope with any degree of intermittency and unpredictability with the focus of the evaluation being consistency across power-failure. Whilst EH is unpredictable and intermittent, there are characteristics of each source that remain consistent. For example, PV cells always provide current-driven, DC power, whereas kinetic harvesters instead give large AC bursts of power when there is a sudden excitation.

Whilst most papers compare systems with more than one application, few use a range of real and simulated harvesters, often demonstrating performance with a single source. Table 4.1 demonstrates the limited range of input sources used in leading intermittent computing papers. This is problematic because intermittency of the supply is the key challenge that the papers aim to overcome. Even the RF or RFID approaches are setup an equal distance, fully oriented towards the constant power source, essentially being equivalent to connecting a constant current source, instead of representing a real scenario.

Furthermore, some of these schemes rely on prediction of the future power supply for increased effectiveness and yet do not acknowledge that predicting future power supply is not as simple with real-world harvesters and environments, and do not validate with such sources.

The choice of EH sources for evaluation clearly must be improved and standardised for fair comparison, particularly where predictive schemes show dramatic improvement without acknowledging the use of unrealistically predictable sources. The use of real harvesters in comparison is challenging due to difficulty in replicating the experiment in new locations. Ideal sources such as square wave supplies of varying frequencies are easy to replicate, but less representative. One alternative is the use of simulated sources where the I-V characteristics of the source are recorded over a set time period and simulated by a power generator to closely mirror the real harvester's response. When evaluating the schemes

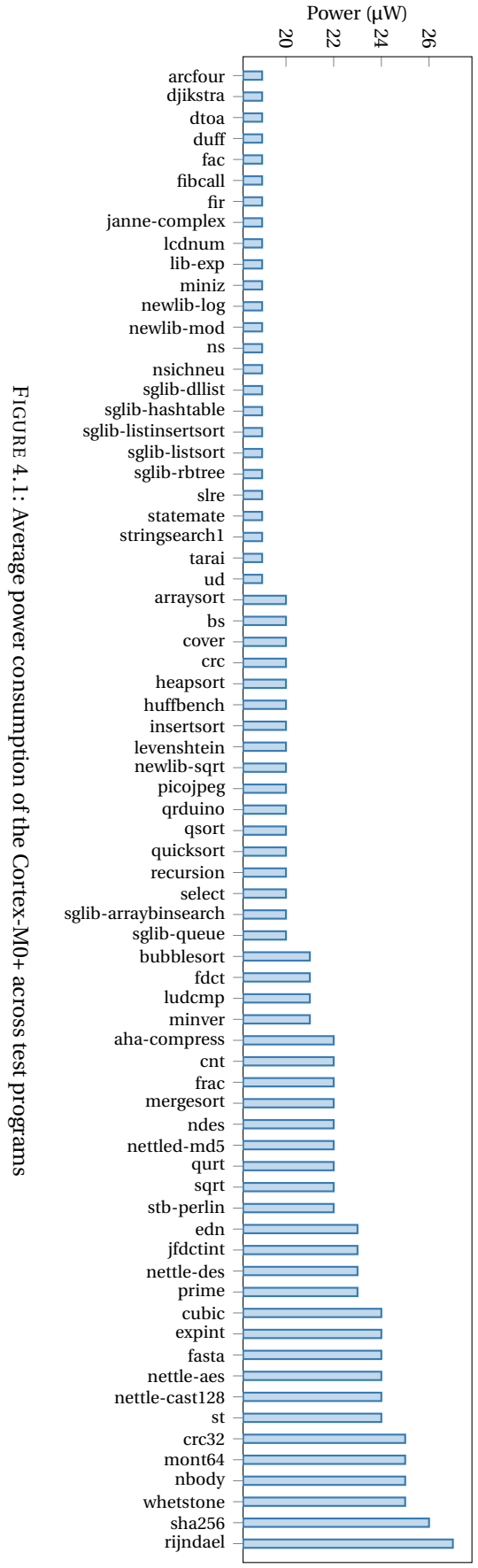


FIGURE 4.1: Average power consumption of the Cortex-M0+ across test programs

TABLE 4.1: Power-traces used for validation of intermittent computing schemes

System	In Simulation	Practical		
		Ideal	Simulated	Real
Mementos [89]	RF harvesting	Capacitor*	Ten "voltage traces"	
QuickRecall [55]		Square wave		
Hibernus [8]		Square wave, sinusoidal wave		
CTPL [2]		Sudden power loss		
DINO [67]				RFID Reader
Hibernus++ [7]	Rectified sinusoidal wave (1-25Hz)	Square wave (1-20Hz), constant current	Wind turbine, kinetic watch, micro PV	Wind turbine, micro PV
Chain [25]				RFID reader
Ratchet [122]	Random power failures (up to 1kHz)			
Clank [44]	No simulated or real sources used			
Alpaca [74]		Continuous		Harvested RF
ENZYME [84]			Four RF power traces	
Flexi-check [100]		14mW constant current,	Vibration harvester (up to 6mW)	

*Capacitor is charged to full before test, then left to decay, and re-charged fully after each failure.

presented in this Thesis a range of supplies are used to evaluate not only the correct operation of the scheme, but also the variation in performance.

Moving beyond a discussion of evaluation and progressing to insight for improved design, it is helpful to determine which sources can be expected to arise in intermittent IoT devices. Many of these harvesters are described in Chapter 2, and this chapter will look in more detail at shared characteristics that can be generalised and intermittent computing systems optimised for.

4.2.1 Low-Power and High-Power Sources

Sources such as PV cells with their high power density could provide abundant power for IoT devices, however where their size, mass and cost are restricted to meet design criteria, it may

not be possible to provide sufficient power for continuous operation of the device. This is especially true when peripherals with a high power consumption, such as transceivers and sensors, are used. This results in power-cycling as the small buffer of on board capacitance charges and discharges. Several schemes have aimed to improve this with the use of increased capacitance, but as discussed in the previous chapter, this may not be appropriate for SoC implementations. It also leads to other drawbacks such as increased cold-start time.

The harvested power input, $P_{harvest}$, can be classified as below for the purposes of this chapter.

$$P_{active} > P_{harvest} : \text{Low-power}$$
$$P_{harvest} > P_{active} : \text{High-power}$$

This allows the power state to be defined according to the characteristics of the device that it supplies. P_{active} will vary from device to device, and during execution, but this definition allows generalised assertions to be made that concentrate more on the relationship between the harvester and the intermittent computing device, rather than power figures.

“Low-power” sources, where the power required by the intermittent computing device exceeds the power produced by the harvester, depend on charging some form of energy storage, typically capacitance, to charge up and reach a useful operating voltage. A source that was low-power and highly intermittent would make useful progress very challenging, therefore energy harvesters used for intermittent devices are more likely to have slow-varying incoming current. This increased predictability can be leveraged in design. It should also be noted that any increase to the power consumption of the device will have a direct impact on the intermittency of this supply with the capacitance depleting more quickly in that case, and more frequent interruptions occurring as a result.

“High-power” sources, where the power produced by the harvester exceeds the power consumed by the device, such as wind harvesters or piezoelectric generators, provide abundant but intermittent harvested power. Abundant power is defined here as exceeding even power hungry peripherals. If a source is abundant and slow-varying then it would rarely cause intermittency in the device. Therefore sources that are high-power can also be assumed to be bursty in the context of intermittent computing research. This high degree of time-power variance makes predicting these sources challenging, and the incoming power supply can decline suddenly. However, the advantage of these sources is that when power is available it can be used aggressively.

When designing intermittent computing systems, high or low power supply characteristics are primarily determined by the kind of harvester used, however it is also important to understand the power consumption of the device and its peripherals to ensure that power output is great enough to support computation. A significant change in P_{active}, such as by adding a peripheral, could change the classification of the harvester. If the EH source

characteristics are intermittent as necessitated by a restriction in the size, mass or cost, it may be possible to predict the characteristics of the supply and utilising this can lead to improved performance.

4.3 Increased Energy Awareness

Increasing the energy awareness, both in design and at run-time, allows the system to better optimise for and adapt to the incoming supply profile. Not just to the incoming power in terms of watts, but also the variation over time.

4.3.1 The Benefit of Design Time Energy Awareness

The relative predictability of these EH sources motivates the exploration of greater energy awareness in design and research, and optimising for the nature of the incoming power depending on the harvested energy source. As with the previous chapter, the desire to directly optimise a specific design, or keeping a high-level research strategy that is applicable to a range of intermittent devices and designs must be held in tension.

Techniques such as maximum power point tracking and good power management must be utilised for an efficient end product that maximises the forward progress. Beyond that, the benefit of design-time energy awareness is that it doesn't require any additional hardware or energy overhead during execution. Any fixed power characteristics can be accurately pre-characterised, such as the consumption of some peripherals and the on-board decoupling capacitance and use these figures to establish voltage thresholds such as was done by Hibernus [8].

Hibernus [8] was the first work to use a hardware interrupt to prompt a state save immediately before power failure, giving a single checkpoint per power cycle. Power failure is defined as the supply voltage to the CPU declining below the minimum operating voltage of the processor V_{min} causing the volatile system state to be lost if not backed up. The author's approach was to set a voltage threshold that would trigger an interrupt as the supply dropped. This threshold ensured sufficient energy in the decoupling capacitance of the system (ΣC) to save state before the supply voltage dropped below the minimum operating voltage of the processor (V_{min}). This static threshold was calculated by ensuring the energy required was equal to the energy in the capacitance, as in Equation 4.1.

$$n_{\alpha}E_{\alpha} + n_{\beta}E_{\beta} = \frac{V_h^2 - V_{min}^2}{2} \times \sum C \tag{4.1}$$

Where V_h is the hibernate threshold voltage, n_α and n_β are the sizes of the RAM and registers (in bytes). E_α and E_β are the energy required to copy each RAM and register byte to NVM (J/byte).

4.3.2 The Benefit of Runtime Energy Awareness

This variation in power dynamics at runtime motivates an exploration into energy awareness during computation. The static threshold used by Hibernus, with its lack of run-time adjustment, could create instability if there is a change to system properties. For example, if there were changes in the dynamics of the power source, or there was increased system power consumption due to the inclusion of additional peripherals or component deterioration. Alternatively a conservative voltage threshold (V_h) could be set to allow changing characteristics at the expense of efficiency. Hibernus++ [7] overcame this limitation by introducing run-time adjustment of the thresholds (V_r and V_h) where V_r is the voltage at which system state is restored and execution resumes. When the system is first powered on, it runs a calibration routine that sets V_h, further adjusting if a checkpoint ever fails to complete. This is described in further detail in section 5.4.

This runtime energy awareness allows the system to adapt to changing characteristics in the power consumption during checkpointing. The amount of energy consumed for saving state and restoring does not vary from application to application in Hibernus++ because the entire address space is saved during checkpointing. This means that whether a lot, or a little of the VM is filled with stack, BSS and registers, the energy consumed to transfer it is constant, assuming the checkpoint occurs at the same voltage each time.

What can vary from device to device, and even with the addition of new peripherals, is the capacitance on board the device and the quiescent current consumption which consumes additional power during checkpointing. Hibernus++ not only allows the designer to do less work with pre-characterising everything, but it will also adjust for these changes. Furthermore, the capacitance of the system can be expected to decline over the life-time of the device as components degrade [75]. As stated before, the overhead of runtime energy awareness is that it must be measured, the most obvious way is with an ADC, but these have a high power consumption. There are alternative ways to include energy awareness which are discussed in section 4.5.4.

Having increased knowledge about the power-in and power-out allows the system to adapt to changes in the supply, inspiring ideas such as dynamically eliminating checkpoints as shown in Figure 4.2. When an interrupt is triggered due to falling voltage when the energy spent on forward progress is less than the energy required to save state, therefore it is more energy efficient to disable checkpointing so long as it is safe to do so, i.e. there have been no idempotency violations.

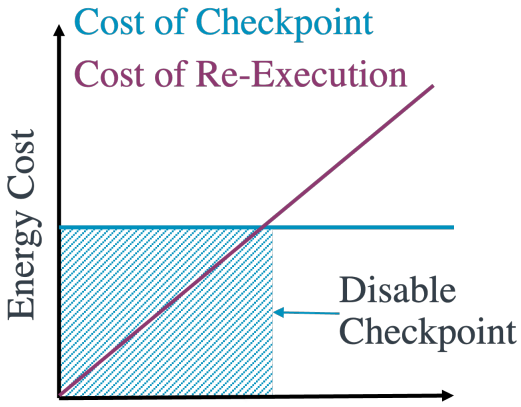


FIGURE 4.2: Dynamically disabling checkpointing, where the shaded region indicates disable checkpointing

Whereas design-time energy-awareness has no negative effect on the performance of the device, only on the burden placed on the designer, runtime energy awareness requires either additional computation or hardware power consumption. For example, a voltage detection circuit or an ADC is needed to monitor the energy stored in the capacitance. Therefore it is important that the most efficient methods of determining the energy state are used, and that this data is used only obtained when most significant to limit additional power consumption in the process. Dynamically disabling the checkpointing is an example of a system with a high degree of runtime energy awareness, making it a costly design. It also eliminates one benefit of reactive intermittent computing by reintroducing the risk of idempotency violations occurring. For these reasons this design is not pursued further.

4.3.3 Energy Consumption During Checkpointing

In a reactive intermittent computing scheme, program execution is suspended when the threshold is detected. This is to prevent an inconsistent state occurring where non-volatile values are updated after the checkpoint has been completed. With dynamic threshold adjustment, as in Hibernus++, reducing the energy consumed during checkpointing means that the threshold can be reduced, and triggered at a lower voltage. This means that a greater proportion of the active CPU time is spent on forward progress.

For “high-power” sources, the energy consumed during execution is irrelevant to the performance of the device because $P_{harvest} > P_{active}$, but after sudden power failure there is a small energy window until the buffer depletes. Examples where this might occur are when pressure is released from a piezoelectric generator or vibration from a car passing stops on a kinetic harvester. Maximising the percentage of this energy used for active computation rather than saving state will lead to increased forward progress.

Figure 4.3 demonstrates the impact of drawing additional current during checkpointing, Higher current consumption requires an increased voltage threshold resulting in decreased

active execution time. The time taken to take the checkpoint remains constant, in this example, but decreasing the time checkpointing takes would also reduce the energy consumption and allow the threshold to be reduced.

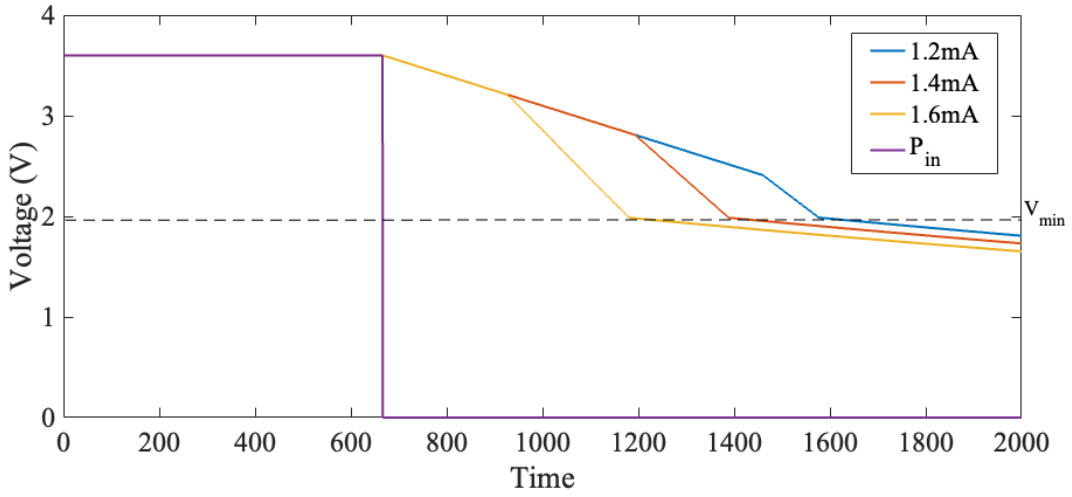


FIGURE 4.3: Representation of the difference in time available for active computation with three different current consumption's during checkpointing

Therefore for high-power sources, although power can be used aggressively during periods of availability, once the supply drops it is important to minimise the power consumption of the device. Fortunately in reactive intermittent computing schemes, once a checkpoint is initiated, no other computation or use of peripherals occurs during checkpointing. There is also no further computation or use of peripherals after the checkpoint is completed, unless the restore threshold is met. This is because unlike task-based systems which accept re-execution of progress after checkpoints, through the careful handling of non-volatile variables (to prevent idempotence violations), reactive schemes take a single checkpoint as close to power failure as possible. Once the volatile data is safely stored, it could corrupt the program state if execution continued without also being saved. For this reason it should be possible to eliminate unused system resources during this critical time window.

4.4 Disabling System Resources for Efficient Checkpointing

The ideal solution is to isolate all unnecessary elements of the system that are draining power either actively or through leakage during the checkpointing routine. All that is needed is to remove power from these devices when the checkpoint occurs, which can be done in one of two ways. For COTS devices, this could be with additional hardware such as the inclusion of a MOSFET to separate the peripherals from the supply when the checkpoint routine is occurring. For an SoC implementation it would require power gating to be implemented, typically using a header switch cell made of PMOS transistors to gate the voltage supply rail. These are further described below. The final subsection explores the

low-power modes of the MSP430FR5739, which isolate different parts of the system, affecting the power consumption and latency.

4.4.1 Isolating Peripherals

Table 4.2 gives the power consumption of some common input transducers. By comparing this with the average power consumed during checkpointing of the MSP430 measured at 3.90 mW, it is clear that the inclusion of any of these sensors would have a significant impact on the state save overhead if left active. This could occur if the checkpoint occurred during their use. Some of these sensors will have a low standby power consumption, however others may require a large initialisation and have a higher standby power to maintain this state once initialised. Either way it should be clear that power-gating peripherals to reduce the power consumption during saving state and restoring will increase the forward progress. This could be controlled by GPIO pins connected to a MOSFET that is triggered at the beginning of saving state, separating the peripherals from V_{cc}.

TABLE 4.2: Power consumption of different types of sensor reproduced from [61]

Sensor Type	Power Consumption
Gas sensor	500mW-800mW
Image sensor	150mW
Pressure sensor	10mW-15mW
Acceleration sensor	3mW
Temperature sensor	0.5mW-5mW

4.4.2 Power-Gating System-On-Chip Resources

The RTL implementation described in Chapter 3 was also simulated at the gate-level for cycle-accurate power consumption information to be used in this research. To focus attention on the work completed for this thesis, only the CPU, RAMs and bus power consumption were analysed despite a much larger SoC being simulated. In total the CPU, RAM and bus made up 17-38% of the total energy consumption of the whole chip during execution. The remainder was only in the initial prototyping phase, and so would be expected to reduce, but still, this demonstrates the opportunity to decrease the leakage power consumption during checkpointing for system elements that are definitely not in use. Clock-gating or power-gating is common in the design of SoCs, the recognition here is that the vast majority of the system's resources can be gated once the checkpoint is triggered, leading to a considerably lower current consumption during checkpointing.

4.4.3 Low Power Modes

The different low-power modes available for the MSP430 are listed in Table 4.3.

TABLE 4.3: Low Power Modes available for the MSP430FR5739

Condition	Current Consumption (μA)	Wake up Time (μs)	Other Info
LPM 0 (8MHz)	177	0.58	– CPU is disabled – ACLK active – MCLK disabled – SMCLK optionally active – Complete data retention
LPM 2	61	12	– CPU is disabled – ACLK active – MCLK disabled – SMCLK optionally active – DCO disabled – Complete data retention
LPM 3	6.4	78	– CPU is disabled – ACLK active – MCLK and SMCLK disabled – DCO disabled – Complete data retention
LPM 4	5.9	78	– CPU is disabled – ACLK, MCLK, SMCLK disabled – Complete data retention
LPM 3.5	1.5	310	– RTC operation – Internal regulator disabled – No data retention – I/O pad state retention – Wake-up input from RST, generalpurpose I/O, RTC events
LPM 4.5	0.32	310	– Internal regulator disabled – No data retention – I/O pad state retention – Wake-up input from RST and general purpose I/O

The power consumption of the MSP430 is significantly reduced in these low-power modes compared with active computation, and many of them have the ability to maintain complete data retention whilst disabling the CPU, auxiliary clock, main clock and sub-main clock. The following section targets placing the CPU into one of these LPMs even during checkpointing. Although the CPU typically plays an important role in the moving of data, there is an alternative solution. Whilst this chapter concentrates on the opportunity of focusing on improvements for high-power sources, the appeal of entering a LPM for reduced current consumption is clearly applicable to low-power sources. This idea is developed in the following chapter.

4.5 Disabling the CPU During Checkpointing

Whilst there are high-power consumption peripherals, the predominant power consumption in the average intermittent computing platform comes from the MCU, with

CPU overhead being a key part of this. Clearly the power performance of CPUs will continue to be improved, however this research demonstrates that it is possible to eliminate the use of the CPU altogether during the majority of a checkpointing routine, instead allowing it to enter a LPM where the energy overhead is significantly reduced.

DMA controllers allows data to be accessed and copied without intervention from the CPU. They are useful in embedded systems for controlling memory transfers between peripherals and main memory. They also can be utilised for speed-up, multi-tasking or reducing load on the processor in embedded systems, however for large block transfers it is possible for the processor to enter sleep, in order to save power whilst the DMA is operating.

Checkpointing in intermittent computing systems necessitates a large transfer of memory, and presents an opportunity for a DMA to be utilised in a novel way. Since execution is suspended in reactive intermittent computing schemes, to prevent NVM locations being altered after checkpointing, it means that the CPU can be placed into a low-power mode during a state-save without any drawbacks. In the pursuit of disabling unnecessary system resources, this scheme demonstrates that even the CPU can be disabled during checkpointing.

4.5.1 Expedit - Reducing State Save Overheads

The design of a reactive intermittent computing scheme that utilises DMA, given the name Expedit, is presented within this section. This scheme could be applied to most existing intermittent computing schemes, but is implemented upon Hibernus++ in this evaluation. Hibernus++ [7] was chosen as a state-of-the-art approach for comparison and design. Its primary overhead being the large time and energy overhead when checkpointing and restoring, offering both an area for improvement but also this is a good scheme to benchmark against.

Reactive transient systems necessitate large memory transfers from RAM into NVM and back. By transferring the registers and RAM into NVM using the DMA, time and energy overheads can be reduced because the DMA is more efficient than a CPU during transfer. The DMA is used in block-transfer mode. It is initialised with a source address in VM, and a destination address in NVM (or vice versa) and the size of transfer. The RAM can be copied in a single block, however the general purpose registers in the MSP430 are distributed, and are transferred in multiple blocks. For the MSP430, a CPU-controlled data transfer uses five cycles, whereas DMA uses two [111]. Additionally, DMA is more power efficient, whilst also allowing the CPU to enter sleep. This gives a lower total power consumption when DMA is in use. The on-chip area overhead is also small, DMAs with a gate count of 3-10k are commercially available [64].

The benefits of Expedit are applicable any time data is copied from volatile to non-volatile memory or vice-versa. Therefore it should be made clear that although Expedit is validated

on an MSP430FR5739 with Hibernus++, it could also be implemented on a SoC, or included as an extension to the majority of the leading intermittent computing papers, across both reactive and task-based approaches. By demonstrating the improvement on a leading scheme such as Hibernus++, both the efficacy of the scheme and improvement over existing research can be demonstrated. In validation, a range of input sources and benchmarks on real hardware is prioritised over combining the scheme with multiple intermittent computing approaches. In addition, the larger the block of data, the lower the relative impact of initialising the DMA. Therefore, reactive intermittent computing systems which copy the entire contents of RAM and registers, such as Hibernus++, gain the greatest benefit. Expedit is simulated and practically validated as an extension to this approach alone.

4.5.2 Expedit Design

In discussing the design of Expedit, only the features of Hibernus++ relevant to the moving of data are included here. Chapter 5 contains greater detail since the scheme, PowerNapping, presented there involves the re-design of other aspects such as the threshold choice. Algorithm 1 shows the original Hibernus++ checkpointing routine.

Algorithm 1 Hibernus++ Checkpoint Routine

Require: $V_{cc} < V_h$
  Copy all 16 core registers to FRAM;
  Set FRAM write pointer to FRAM-Start Location;
  for $i = 0; i < 16; i++$ **do**
    Increment FRAM write pointer;
  end for
  while RAMptr < RAMend **do**
    Copy RAM to FRAM location;
    Increment pointers;
  end while
  for $i = 0; i < 230; i++$ **do**
    Pointer made to Registers[i];
    NVM-Registers[i]=pointer;
  end for
  return State save success

The Hibernus++ code uses a number of 'for' loops, arrays and temporary variables to copy the registers and RAM. This is an inefficient approach. Algorithm 2 shows how much simpler the code for state save is with Expedit, and restore is similarly simplified. The size of the code written to the MSP430 can be expected to be reduced.

Hibernus++ was designed for and evaluated on an MSP430FR5739. This experimenter board is prevalent in the field due to its on-board FRAM memory. Fortunately it also has a DMA controller, and so can also be used in the evaluation of this work. The DMA controller module has three transfer modes: single transfer, block transfer and burst-block transfer.

Algorithm 2 Expedit Checkpoint Routine

Require: $V_{cc} < V_h$
Copy all 16 core registers to FRAM
Initialise DMA with source, destination and size (RAM, FRAM, size)
Enable DMA, CPU enters LPM
Initialise DMA (Registers,FRAM,size)
Enable DMA, CPU enters LPM
return State save success

These were explained in subsection 2.1.2, block transfer is selected as the most appropriate for this application.

If the design of an intermittent computing approach requires accurate and faultless handling of data transfer between non-volatile and volatile memory, designing an approach that passes this task onto another piece of hardware requires precise initialisation. Recognising which addresses the DMA is able to access (including which registers), how large a single transfer can be and ensuring that interrupt priority won't result in an error state during the triggered checkpoint are all important. The Expedit schemes simplicity is the key to its enhanced performance over more complex data-transfer approaches, so there were not many challenges, however these were where most errors in designing the approach occurred.

For example some of the general purpose registers are read-only meaning that during the restore routine a single transfer resulted in an error, and so the DMA must be initialised several times to skip out these registers. In this instance the DMA can be used in repeated block transfer mode where the DMA enable remains enabled instead of being cleared at the end of each block transfer.

4.5.3 Simulation

Expedit is expected to reduce the overhead of the state save and restore routines, both in terms of time due to the increased efficiency of transfer, but also power since the CPU is able to enter LPM. This will result in an even greater improvement to the energy consumption. To verify the behaviour, and predict performance, mathematical simulation was used. Energy consumption for active, sleep, hibernation and restore were obtained from both the MSP430FR5739 data sheet [1] and physical characterisation. MATLAB was used to run these values through the mathematical models with various test conditions and input sources as demonstrated below. In simulation, the overhead of initialising the DMA is considered negligible, because the cycles required to move data are orders of magnitude greater than initialising. Because the DMA uses two cycles per word, instead of five, hibernate and restore times are predicted to be reduced by 60%. The power consumption of the MSP430FR5739 was measured to be 10% less with the DMA active and CPU in sleep, so this is also included.

Expedit is expected to give the greatest improvement over Hibernus++ when time spent on hibernating and restore as a function of the total completion time is high, such as with frequently interrupted *high-power* sources. The simulated improvement for such a source is given in Figure 4.4.

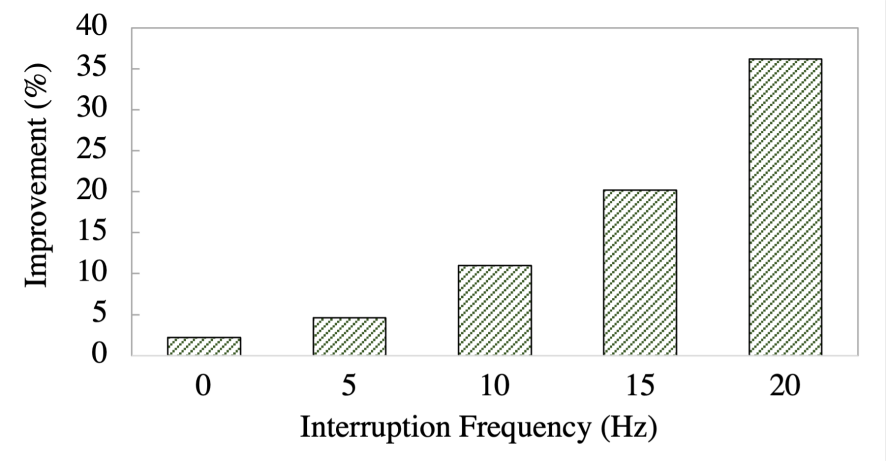


FIGURE 4.4: Predicted improvement in forward progress according to MATLAB simulation for an interrupted high-power source

As predicted, the improvement is shown to be greater at higher interruption frequencies.

For *low-power* supplies, the system typically does not drop below V_{min} after the checkpoint is taken, because the CPU enters a LPM which consumes less power than $P_{harvest}$ and so the voltage across the capacitor recovers. Therefore only T_h is relevant, and the improvement is expected to be less than for *high-power* supplies. Figure 4.5 shows the expected improvement of Expedit in simulation for low-power sources and a PV cell modelled as described in section 5.6.

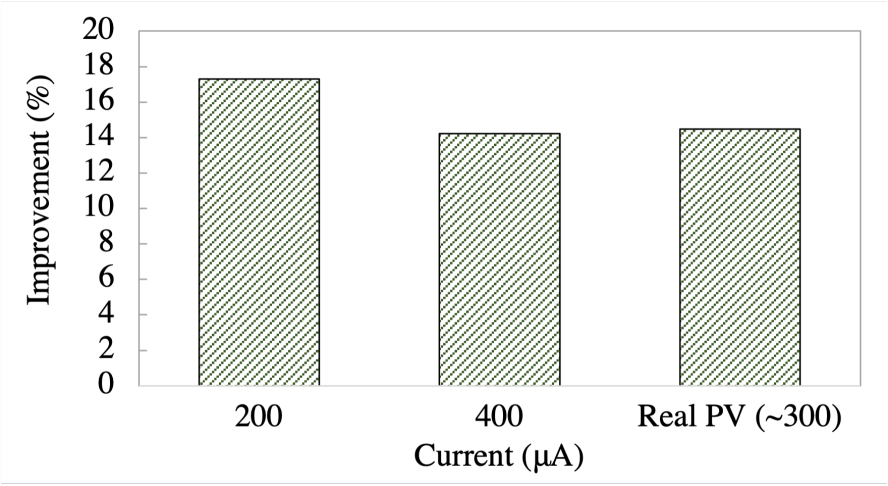


FIGURE 4.5: Predicted improvement in forward progress according to MATLAB simulation

Where the first two results are a constant current source, the modelled PV cell although estimated at 300 µA when at 3V, will deliver a different current at different voltages according

to the I-V characteristics. Where Expedit is able to trigger a checkpoint at lower voltages than Hibernus++, more time is spent operating at a lower voltage, which results in less improvement than the equivalent constant current source. This is a great example of where testing with a greater range of input sources can identify flaws in the predicted improvement over the state of the art due to other factors.

The mathematical simulation used here is insufficient to provide a fully accurate evaluation, and so practical evaluation using a physical experimenter board is performed as described below.

4.5.4 Practical Validation

The practical validation of the Expedit scheme is completed on an MSP430FR5739 experimenter board, with an external variable voltage threshold circuit, the details of which are presented in Hibernus++ [7]. This was selected as the current consumption is approximately 1 µA compared to 76 µA measured for the internal comparator circuit. Figure 4.6 shows the experimenter board. The accelerometer and LEDs displayed here were removed with a soldering iron to reduce the power consumption of the board. The debugging and programming interface is disconnected after the .hex file is uploaded and before testing by removing the jumpers.

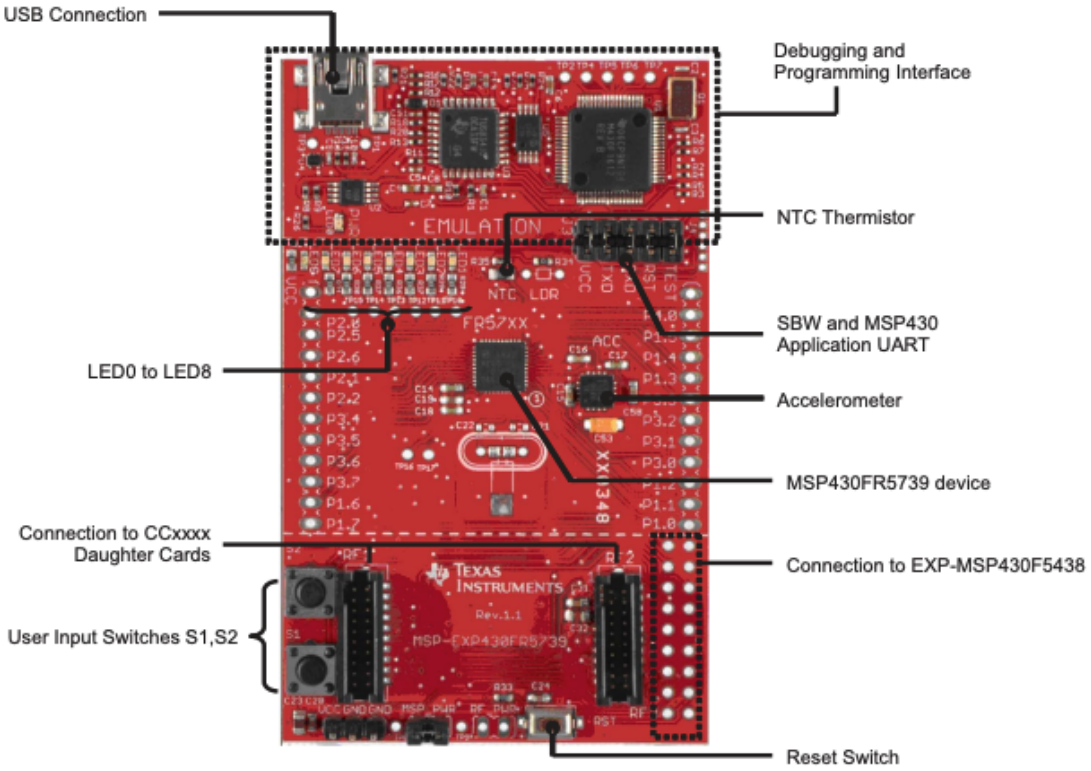


FIGURE 4.6: MSP-EXP430FR5739 Overview taken from [112]

4.5.4.1 Experimental Method

Once the circuit is assembled and the compiled .hex file containing the adapted Hibernus++ code and a fast-fourier transform (FFT) benchmark is flashed onto the MSP430, the board is connected to the external comparator and the digital channels of a picoscope (PC oscilloscope) as shown in Figure 4.7 and Figure 4.8.

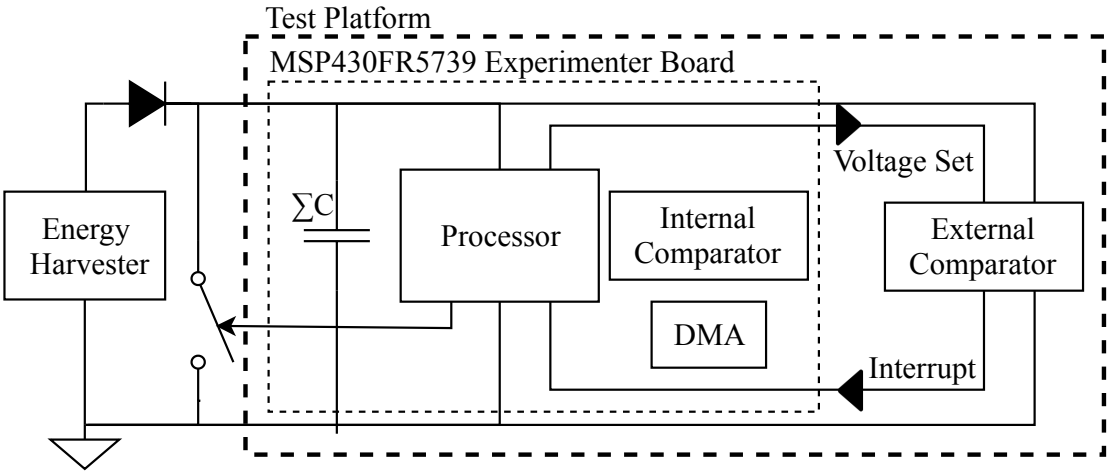


FIGURE 4.7: Schematic of the test platform

The energy harvester output is rectified by a Schottky diode. A low-dropout regulator (LDO) could be included to regulate the voltage within safe limits, however none of the sources used in testing exceeded the maximum operating voltage of the MSP430.

To obtain the results from the experiment different pins were set high at significant moments during the execution, and monitored by the picoscope e.g. execution start, checkpoint triggered, state save completed, etc... From these signals the times spent performing each of these tasks were established and compared with each other. Hibernus++ was also measured in the same way to produce the baseline for comparison. Figure 4.9 shows an example of the trace produced when performing a FFT benchmark on the picoscope from which data can be measured.

4.5.4.2 Results

The direct impact of the Expedit scheme is a reduction in the state-save and restore overhead, and so these are measured first and quantified in Table 4.4.

Table 4.4a shows that Expedit is faster than expected in simulation. This is thought to be due to the additional algorithm simplifications, such as not reading register locations from a look up table. Table 4.5 shows how this also affects the size of code written to the MSP430 when programming the device. FFT is also given as an example base application code size.

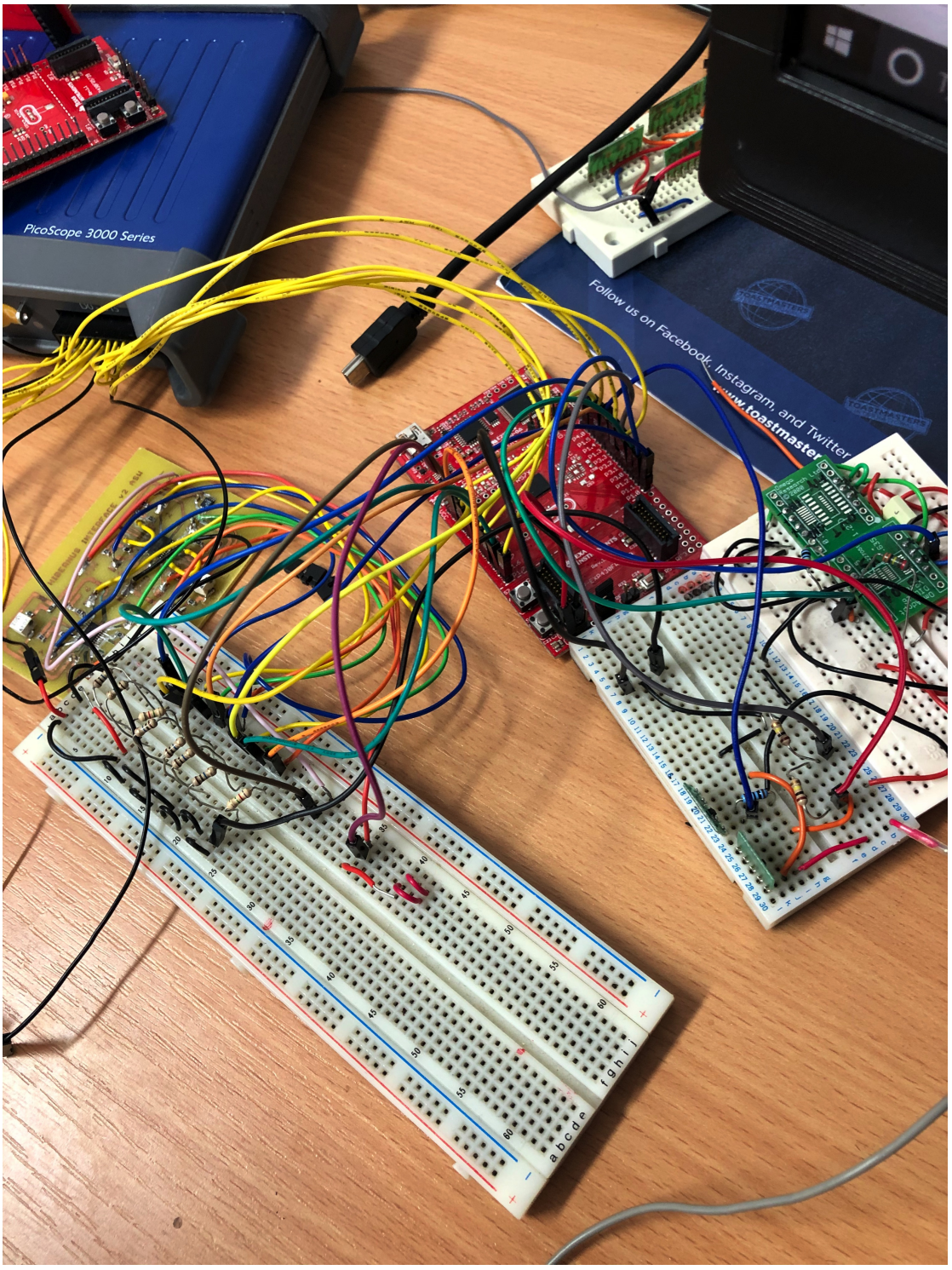


FIGURE 4.8: Experimental Test Setup

Table 4.4b shows the improvement in energy consumption. Expedit not only improves energy efficiency by completing the hibernate/restore more quickly, but the DMA also consumes less power. This leads to a hibernate energy cost reduction of 73.9% and restore, 85.6%. By reducing these overheads, more time and energy goes towards forward progress.

As can be seen, Expedit reduces the size of the code overhead by over 1000 bytes (11%). This

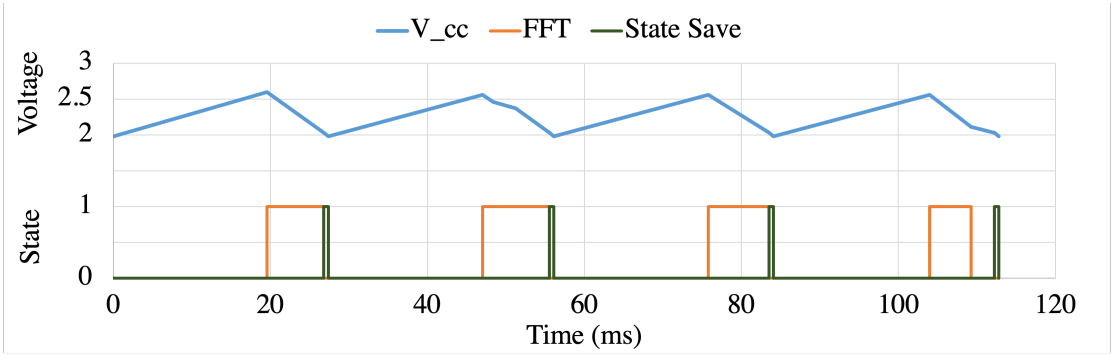


FIGURE 4.9: Voltage over time for FFT benchmark running Expedit, including digital signals for active computation and state saving

TABLE 4.4: Total overhead comparison of Expedit (Exp) against Hibernus++ (H++)

(A) Time to complete listed functions

	H++ (μs)	Exp (μs)	Improvement (%)
Restore	1874	306	83.4
Hibernate	1876	555	70.4

(B) Energy to complete listed functions

	H++ (µJ)	Exp (µJ)	Improvement (%)
Restore	7.30	1.05	85.6
Hibernate	7.31	1.91	73.9

TABLE 4.5: Comparison of code overhead for proposed schemes.

Additional Overhead	Code Bytes	Data Bytes
FFT alone	1852	1248
Hibernus++	+3486	+1508
Expedit	+3348	+586

may enable Hibernus++ to be implemented on more tightly constrained systems. Hibernus++ contained large arrays of address locations (increasing data bytes), and 'for' loops for moving data (increasing code bytes). These elements are removed by using DMA.

Wind harvesters and piezo-electric vibration harvesters are examples of high-power sources. In a reactive intermittent computing system with little energy storage, the processor is likely to hibernate and restore frequently. To represent a high-power source in this validation, an ideal DC voltage is interrupted with varying frequency to approximate the system response to these types of harvester. Figure 4.10 further demonstrates the impact of these reduced times by comparing the percentage overhead for a range of supply interruption frequencies. The higher the supply interruption frequency, the more state saves are required, and the greater the relative benefit of Expedit. Figure 4.10a demonstrates how the percentage of the active time spent on hibernate/restore is increasingly less than Hibernus++ and Figure 4.10b demonstrating the consequence on energy overhead.

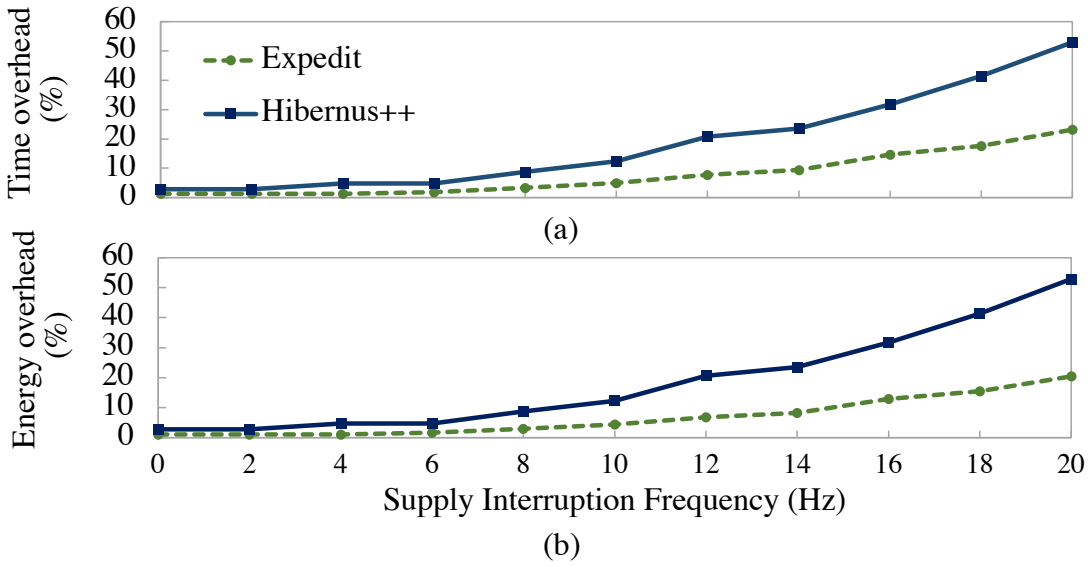


FIGURE 4.10: Overhead comparisons for Expedit and Hibernus++

The increase in active time can be illustrated by focusing on one cycle of a *high-power* AC input in Figure 4.11. By reducing the time and energy required, the hibernation can begin later, at a lower voltage threshold. This is established when first calibrating the system as explained in Section 5.4. Doing so results in a longer active time per power cycle, which is the cause of the increased forward progress demonstrated by Expedit.

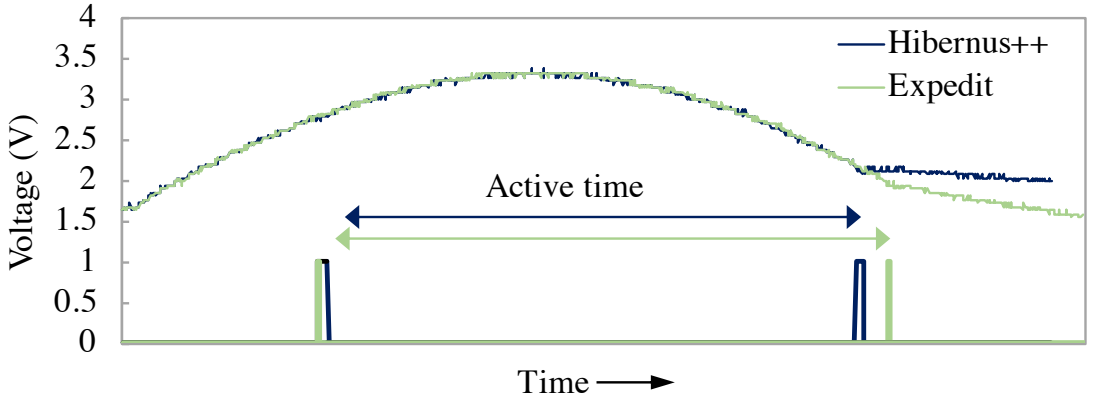


FIGURE 4.11: Expedit increase in active time with sinusoidal voltage input

The improved state save and restore overheads are good, however it is the impact on the forward progress of the application running on the device that demonstrates the significance of the improvement. Demonstrating an increase in the forward progress means that the intermittent computing device could either handle more complex applications for a given time/supply profile, or complete the same load in a shorter period. Either could be beneficial to system designers who are deploying these kinds of devices to meet their requirements.

An FFT completing three times is selected as a representative load, with three axes of accelerometer data provided as the input. The total time to complete these three FFTs was measured for a range of interruption frequencies (high-power source) for both Hibernus++ and the improved Expedit scheme. Figure 4.12 shows the improvement in completion time.

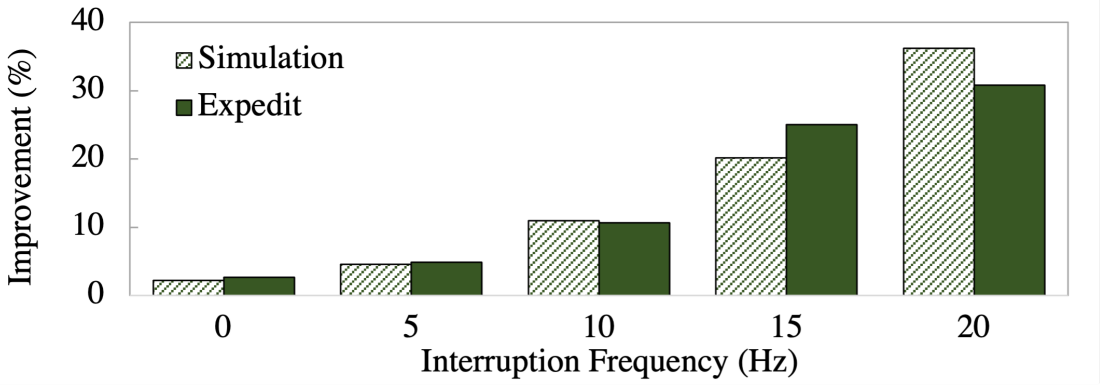


FIGURE 4.12: Improvement in FFT complete time for Expedit over Hibernus++ including simulation for a representative high-power source

This shows up to 36.6% improvement in the completion time for a high-power source. As predicted, the greater the percentage of time spent checkpointing, the greater the improvement. Figure 4.13 shows the improvement for a greater range of interruption frequencies up to 26Hz.

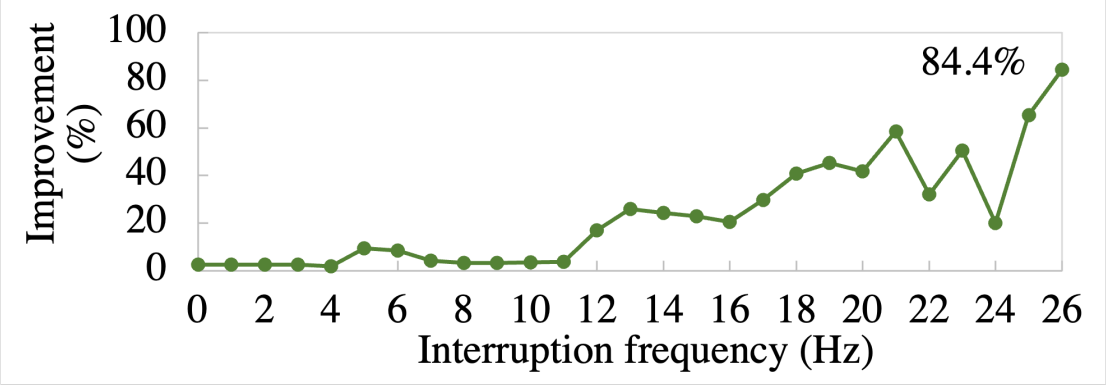


FIGURE 4.13: Improvement in FFT complete time for Expedit over Hibernus++ for 0-26Hz interrupted source

This figure also indicates the downside of the approach to measuring improvement as a function of the completion time. Although the improvement appears non-linear, what is happening is that the improvement is proportional to the number of interruptions that occur before completing the FFT, and the number of interruptions is not directly proportional to the frequency of interruption. Hibernus++ almost always has more interruptions than Expedit due to the larger overhead. Sometimes the FFT will complete just before the final interruption, and sometimes just after resulting in an additional power cycle. This occurs with both Hibernus++ and Expedit, so when the two are compared the results vary non-linearly. Figure 4.14 compares the improvement to the number of

interruptions rather than the interruption frequency for a few benchmarks to demonstrate that there is a proportional increase as expected.

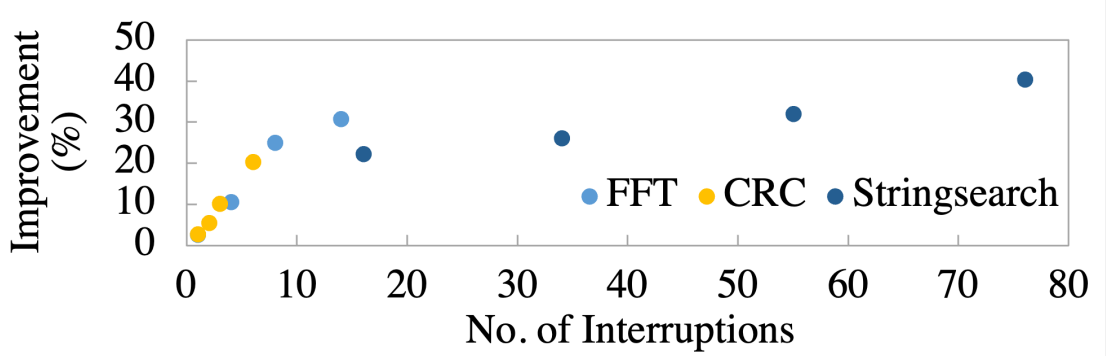


FIGURE 4.14: Improvement in benchmark complete time for Expedit over Hibernus++ compared with number of interruptions

Although this chapter has focused on the opportunities to improve intermittent computing for high-power sources, Expedit also gives improvement for low-power sources as simulated earlier in the chapter. Figure 4.15 shows the improvement in completion time compared with the simulation for low-power sources.

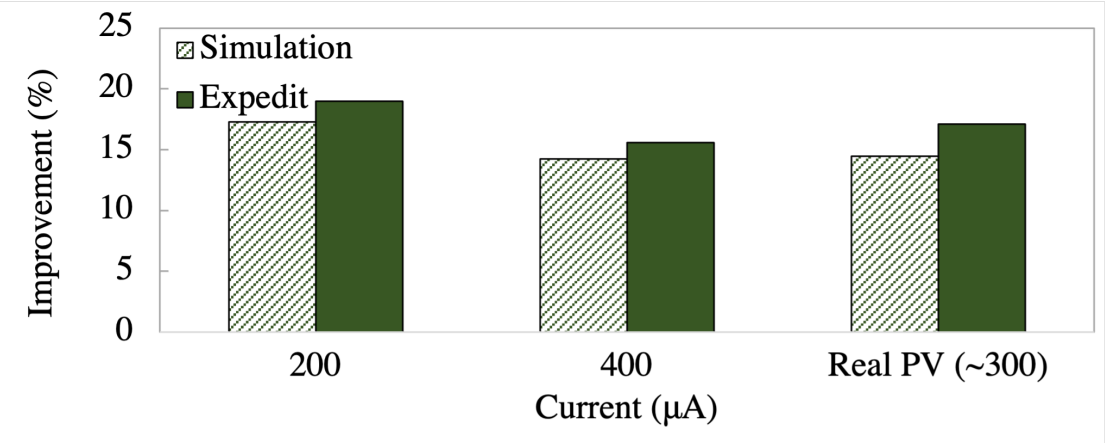


FIGURE 4.15: Improvement in FFT complete time for Expedit over Hibernus++ including Simulation for low-power sources

The reductions in the time and energy overheads by Expedit are applicable not just to high-power sources on Hibernus++, but all checkpointing strategies with all power sources where blocks of data are moved from VM -> NVM and back. Although the DMA means a slightly increased footprint for SoC, the performance benefits for intermittent computing have been demonstrated. The initialisation cost is the only runtime overhead, and this will reduce relatively as the number of bytes in the block to transfer increases.

4.6 Summary and Discussion

Characterising “power-in” as high or low power provides opportunities to design intermittent computing systems for increased performance with a range of harvesters, separated by characteristic. High-power places no constraints on the power consumption of the device when active, but when supply failure occurs, and a state save is required it is important to minimise the current consumption of the device.

For reactive intermittent computing systems, all elements of the system not required for checkpointing should be power-gated to reduce the current consumption. One element of the system that is needed for state saves is the CPU, however an alternative method of saving state using the DMA is presented that allows even the CPU to enter a LPM, reducing the time and energy overheads of checkpointing by up to 83.6% and 85.6% respectively. This gives up to an 84.4% improvement in completion time over the state-of-the-art. Expedit is able to reduce the overheads, and improve the forward progress so significantly because the reactive scheme is able to checkpoint so much later. This is possible because the energy consumed during saving state is reduced, and therefore the buffer depletes more slowly, meaning the checkpoint threshold can be reduced.

Whilst this scheme was designed to optimise for high-power sources, improvement of up to 19% has also been demonstrated for low-power sources. The following chapter aims to specifically target low-power supplies for increased performance.

Chapter 5

Intermittent Computing with Consistent, but Scarce Energy Sources

5.1 Introduction

Having designed a scheme to optimise for high-power, yet discontinuous power sources, this chapter focuses on the other characteristic indicated in section 4.2.1, designing for slowly-varying, but low-power, sources.

Energy harvesters that provide a slow-varying supply may not be considered "intermittent" at first inspection, however when they provide less power than is required for active processing, or peripheral use, the supply voltage will fluctuate as the buffered energy depletes, or charges. Harvesters providing power of this nature include PV cells and RF energy harvesters. Their limited surface area or antenna size respectively can limit the available output power below that required for active computation.

If the power generated cannot be increased, then the active power consumption can be reduced, however this may not be sufficient to prevent intermittent execution. In light of this, for reactive approaches, it is important that checkpointing performance is increased. For devices with a consistent but low-power input, any gains in checkpointing performance at the expense of runtime efficiency would still deplete the buffer and lead to increased intermittency. Therefore schemes should ideally increase checkpointing performance without significantly increasing the active power consumption.

Section 5.2 describes an approach to minimise the checkpointing overhead, and complete tasks before power failure. An alternative scheme, presented in section 5.3 further leverages the relatively predictable nature of these harvested sources to greatly reduce checkpointing frequency and improve the forward progress. This is experimentally validated in section 5.7 demonstrating up to 50% improvement over a state-of-the-art approach.

5.2 Push-Through

Typically a reactive intermittent computing scheme will initiate the same behaviour each time a checkpoint is triggered by the falling voltage. Once it is triggered, there are no additional decisions made. As stated previously, the key advantages of reactive over task-based schemes includes the lack of additional software development, the lack of energy wasted on re-execution, and that tasks/applications do not need to be correctly sized to prevent sisyphean tasks. The primary benefit of task-based schemes is their lower checkpointing overhead.

5.2.1 Task Completion

If reactive and task-based computing could be combined with a predicted energy expenditure then it should be possible to make a decision at runtime about whether saving state is the best approach, or whether instead pushing through to the end of the task could be attempted. There are two ways of defining a task in this instance. Either the entire benchmark, e.g. CEM which is one among many tasks such as sense, transmit, etc... or using a task-based approach to subdivide the application into tasks where the boundaries are defined.

Task-based approach typically struggle to optimally size tasks so that they don't take too long to complete. If combined with a reactive approach, progress can still be made. If the task is only part-way through on power-failure, a reactive checkpoint can be taken.

Figure 5.1 shows an example of a checkpointing decision for the push-through scheme. Where E_σ is the energy required to transfer system state and $E_{complete}$ the energy required to complete the current task.

The central decision for push-through is whether $E_{complete}$ is greater than or less than E_σ. For approaches where the whole RAM is saved, E_σ would be fixed, but $E_{complete}$ will vary according to progress through the task and the number of cycles yet to complete. As we've demonstrated earlier in this chapter, the overhead of a clock cycle can vary significantly depending on what is being executed: memory access, CPU operation, etc.... This opens the opportunity for two main solutions: pre-characterisation or prediction.

Pre-characterisation would require the use of look-up tables that predicted the energy remaining based on the number of cycles that had been performed. Prediction through machine learning would estimate the energy required. Both approaches add significant complexity.

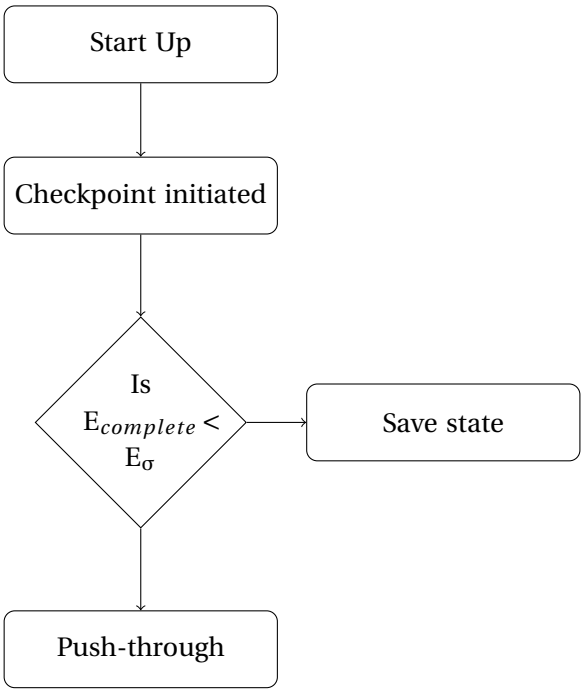


FIGURE 5.1: Decision flowchart for push-through mechanism

5.2.2 Checkpoint Optimisation by Reducing Stack

A further level of push-through that could be considered is where instead of pushing through to the completion of a task, the execution runs until a point where the stack significantly reduces. Figure 5.2 shows the stack variation over time of the mergesort benchmark. As you can see the stack shrinks significantly at ~0.95ns, this would greatly reduce the size of the checkpoint as none of these volatile values would need to be saved. If the energy required to reach this point is less than the energy required for checkpointing, then it is beneficial to forward progress to reach that point in the code.

5.2.3 Increased Complexity

Although section 3.4.2 showed a clear need for reducing the overheads of checkpointing in order to implement the design for an SoC with a reasonable capacitance, and the benefits of push-through are promising, the implementation is not straightforward. Section 4.3.2 explains that increasing the energy awareness at runtime introduces additional overheads. Push-through would require precise energy data for the energy remaining in the capacitance, the energy required to continue task to completion and the energy for the checkpoint following task completion. Whilst the first of these can be measured, the others must be calculated at runtime using either pre-characterisation or prediction as the point in the execution will be different at each interruption.

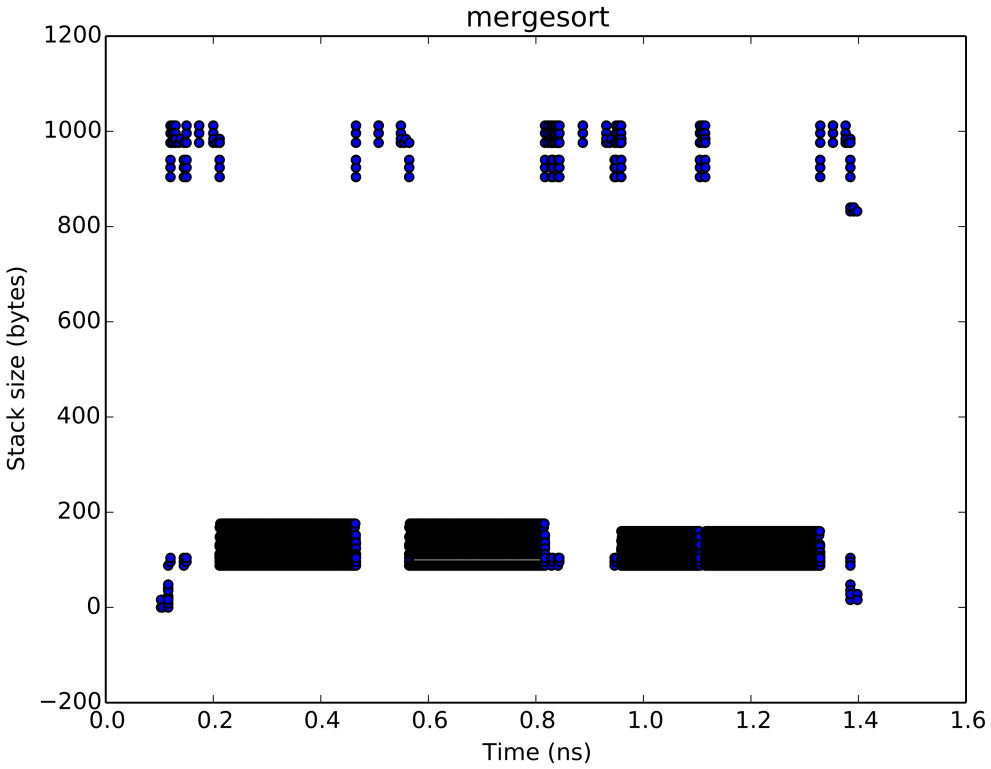


FIGURE 5.2: Stack variation over time for mergesort

For this reason, the concept of push-through is not developed further in this research, the additional energy awareness is expected to increase the active power consumption, which is not desirable for low-power sources.

An alternative method of progressing through power failures without checkpointing for low-power supplies is presented instead. Where push-through targets any source, this scheme, called PowerNapping, is designed only for low-power sources. However, it achieves a significant increase in the forward progress and significantly reduces the overheads of checkpointing in specific cases to meet the needs described in section 3.4.2.

5.3 PowerNapping

As established in the earlier chapters, reactive intermittent computing approaches provide a way of coping with intermittent power, without increasing program complexity, or risking memory inconsistency. Unified FRAM memory approaches are costly for low-power supplies, as established in 3.3.2, because the additional power consumption results in more frequent checkpointing. This means a separate RAM is preferable for these sources. However, this results in large checkpoint overheads, with checkpoints in the range of 1000s of bytes being saved on every supply interruption. This has a significant impact on the

forward progress of computation on these devices, particularly if the supply interrupts frequently.

Understanding the predictability of certain EH sources enables solutions to these large checkpointing overheads to be explored through this lens. Whilst supplies such as wind-harvesters or piezoelectric generators have a high temporal power variance, with the power they supply being highly intermittent, solar and thermoelectric harvesters instead typically have a slow-varying DC output. Whilst these slow-varying sources typically provide insufficient power for active computation at times, they more frequently supply power that exceeds the consumption of low-power modes. Table 5.1 demonstrates that for a $1cm^2$ PV cell, LPM4 would allow the capacitance to charge, however active computation would drain it. The data taken from [97] does not record the associated lux levels, but on a very cloudy day much less than 40µW would be expected.

TABLE 5.1: Power comparison for MSP430FR5739 [111] and $1cm^2$ PV cell power data taken from [97]

Incoming Source	**Power generated**
$1cm^2$ cell - sunny day	120µW
$1cm^2$ cell - cloudy day	< 40µW
CPU state	**Power consumption at 2.5V**
Active	3.90 mW
LPM 0 (8MHz)	442.5µW
LPM4	14.75µW

In subsection 4.2.1 two states of power were defined. Here an additional power measurement is included, P_{sleep} which indicates the power consumption of a device in low-power mode, such as LPM4 shown in Table 5.1, where no active processing is being completed. This results in updated definitions and a third state of harvested power input: Insufficient Power. Being in a state of *low-power* or *insufficient power* is separated by whether a buffer connected to the system would charge or discharge when in a LPM.

$$P_{harvest} < P_{sleep} : \text{Insufficient power} \tag{5.1}$$

$$P_{active} > P_{harvest} > P_{sleep} : \text{Low-power} \tag{5.2}$$

$$P_{harvest} > P_{active} : \text{High-power} \tag{5.3}$$

$P_{harvest}$ is the power supplied by the harvester, P_{active} is the active consumption of the device and P_{sleep} is the consumption of the device when in sleep mode.

The operating voltage is dependent on the energy stored in the system capacitance. Where the harvested power, $P_{harvest}$, exceeds the power consumption of the system (P_{active} or P_{sleep} depending on system state), the energy, and therefore operating voltage, rises; on the other hand, operating voltage will drop, eventually leading to the system responding by hibernating.

Hibernus++ sets the threshold for state save with a conservative approach, calibrating and setting voltage thresholds for the worst case power interruption: sudden incoming supply drop to 0 V. This leads to the system hibernating earlier than necessary if supply instead declines slowly. *Low-power* is sufficient to retain state in VM if in sleep mode [111]. If *low-power* is sustained, existing reactive systems still complete resource-intensive state saves which are unnecessary if the supply can instead be prevented from dropping below the minimum operating voltage of the processor, V_{min}, using a sleep state, consequently retaining volatile data.

Furthermore, for these devices to meet the criteria of small mass, size and cost, the energy harvesters they employ must also be small. These smaller harvesters can be expected to deliver *low-power* frequently, meaning that an increasing number of state saves occur unnecessarily, since VM contents are not lost.

It was suggested by Lukosevicius et al. that introducing a third threshold to trigger sleep without saving state [68] could allow the system supply voltage to recover without dropping below V_{min}. This simple approach was limited, with the threshold set statically as in Hibernus, and always enabled. If there is *insufficient power*, entering a sleep state would unnecessarily increase the time overhead of checkpointing.

Entering a sleep state is an effective tool for intermittent computing, however it must be done appropriately, efficiently and not interfere with saving state correctly. This section presents a scheme that achieves this with adaptive thresholds, supply characterisation, and safeguards to avoid missing a state-save, even on total power failure. Following this, the proposed scheme is validated and evaluated as an extension to Hibernus++. The benefit to forward progress is shown to be up to 46.8%.

5.4 PowerNapping Design

Designers of intermittent computing devices don't typically incorporate additional batteries or supercapacitors due to the increased size, mass and cost this leads to. The devices do have a decoupling capacitance for other purposes, and this can be leveraged as small, but non-negligible buffering of energy to allow time to save system state.

The MSP430FR5739 experimenter board[1] has a decoupling capacitance of 16µF. This means that when the incoming power is greater than the consumed power, the buffer will charge, and the voltage across the capacitor will increase. Where the incoming supply is relatively consistent, such as an indoor PV cell under constant lighting conditions, reducing the consumed power below the supply should prevent power failure. *Low-power* is sufficient to retain state in VM if in sleep mode [111].

[1] An MSP430FR5739 is chosen for this work due to its built in FRAM memory and prevalence in related intermittent computing works.

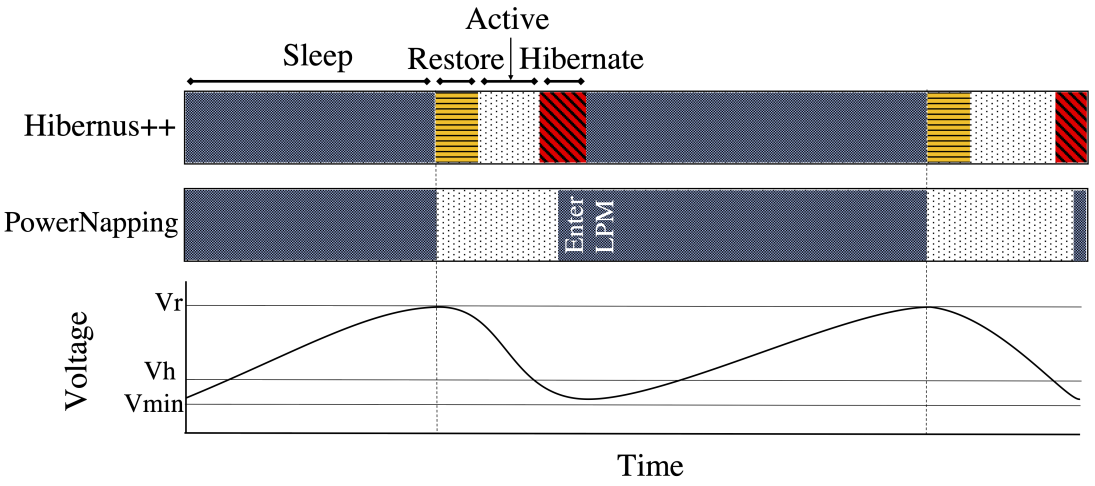


FIGURE 5.3: Illustration of system operation for a given supply with PowerNapping. Hibernus++ restores at a voltage threshold, V_r, and hibernate at V_h. PowerNapping, when enabled, instead of hibernating enters a sleep mode and retains data in VM, simply resuming at V_r.

If *low-power* is sustained, existing systems still complete resource-intensive hibernations which are unnecessary if the supply can be prevented from dropping below the minimum operating voltage of the processor, V_{min}, with sleep, and therefore retaining volatile data. This energy could otherwise be used to achieve forward progress. This is illustrated in Figure 5.3.

Furthermore, for these devices to meet the criteria of small mass, size and cost, the energy harvesters they employ must also be small. These smaller harvesters can be expected to deliver *low-power* more frequently, meaning that an increasing number of checkpoints are taken unnecessarily, since VM contents are not lost.

Entering sleep instead of checkpointing when there is a *low-power* harvested supply should therefore allow the supply voltage to recover without dropping below the threshold V_h, as defined in Equation 4.1. The incoming supply should not be expected to be sustained indefinitely, and a routine for checkpointing correctly should still be included to prevent data loss on power failure.

This enhanced approach, which is named PowerNapping, can be combined with existing intermittent computing approaches, both reactive and task-based. Since it prevents the entire checkpointing overhead, it follows that the schemes with the highest checkpointing overhead will benefit most significantly.

Hibernus++ [7] is a leading reactive intermittent computing approach. It is an improvement upon the original Hibernus [8] approach, that was demonstrated to be more effective than other similar schemes [91], despite the large checkpointing overhead. This is in addition to its energy awareness that is already incorporated. For these reasons, PowerNapping is added to Hibernus++ and evaluated against it.

The operation of the enhanced PowerNapping approach is detailed in Figure 5.4. By identifying which power state the system is in (Eq 5.1-5.3), and the current supply voltage, this system determines whether it is most effective to sleep, hibernate or restore.

High or low power supply is determined at run time during the start-up routine, as later explained, and insufficient power is detected by dropping below V_h during sleep. PowerNapping introduces additional states and decisions as shown within the dotted box, when compared to Hibernus++. This PowerNapping state adaptation could be applied to other schemes such as QuickRecall [55], but the Hibernus++ implementation is used for comparison throughout this chapter.

The voltage thresholds V_s, V_h and V_r in Figure 5.4 are required for reactive checkpointing, and are set at runtime according to the system properties with a calibration routine run when the system is first powered on:

1. An ADC is used to measure voltage (V_1). The supply is then isolated with a diode.
2. The system sleeps and wakes before taking a second reading (V_2).
3. The system hibernates and then takes a third reading (V_3).

The voltage threshold indicating there is sufficient energy for sleep/wake, V_s, is characterised by the voltage drop (V1-V2) and for hibernation, V_h, (V2-V3), giving the following thresholds:

$$V_h = V_{min} + (V_2 - V_3) \tag{5.4}$$

$$V_s = V_h + (V_1 - V_2) \tag{5.5}$$

Hibernus++ [7] dynamically sets the hibernate and restore thresholds as described earlier, but it also contained a method of determining whether the source was *high-power* or *low-power* when restoring. It was the first intermittent computing work to do this at runtime. This power-awareness is valuable information for coping more effectively with supply intermittency, however it is only utilised in the restore operation by the Hibernus++ scheme.

On system start-up, it tests the supply by taking two ADC readings, as shown in Figure 5.5. If the second is higher than the first, it identifies that there is *a high-power* harvested supply and the system restores state immediately, utilising the 'abundant' power to maximise forward progress. There is no downside to this, since, in a transient system, excess energy cannot be stored. If the second reading is lower, there is a *low-power* supply and the system will wait until there is sufficient charge on the capacitor before restoring. Trying to restore immediately would draw power, resulting in a voltage drop below the hibernate threshold.

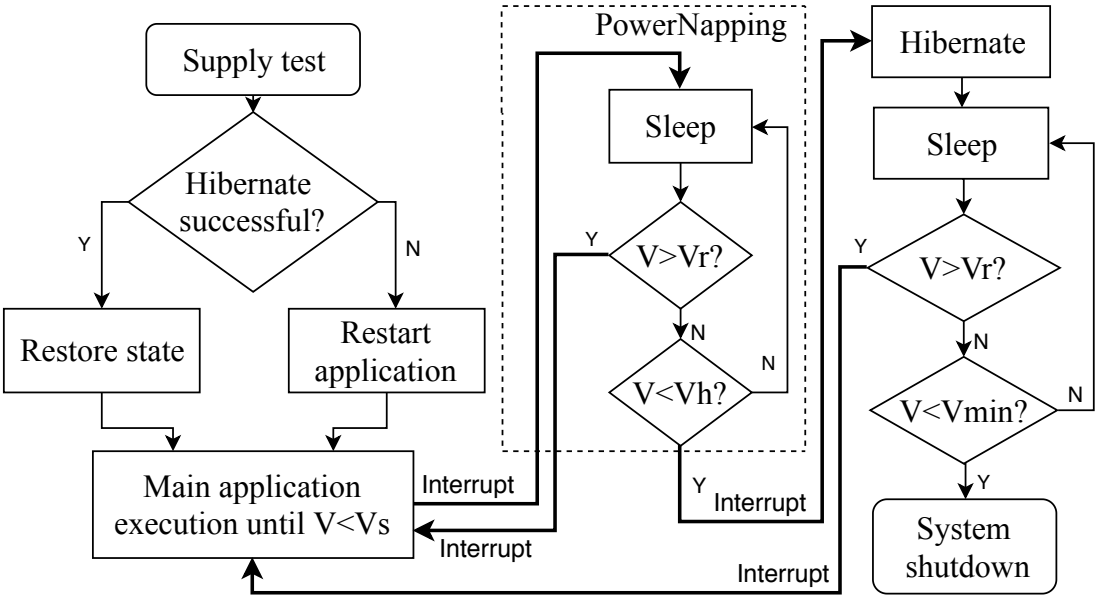


FIGURE 5.4: State diagram of PowerNapping approach

To avoid this, there is an incrementing voltage and timer interrupt. Whilst the voltage interrupts first, the charge is increasing steadily and it is better to continue charging the capacitance. When the timer triggers first, this means that the rate of charge has plateaued and the system restores. The voltage at which this occurs is set as the restore threshold, V_r, used to exit subsequent sleep modes. This will be greater than V_s and V_h.

Using this information about high and low-power supply state PowerNapping can be enabled or disabled at runtime. This is important because for high-power sources, it is typical to go from *high-power* directly to *insufficient power*, meaning that PowerNapping would just waste time on entering LPM and waking, which could instead be used for forward progress.

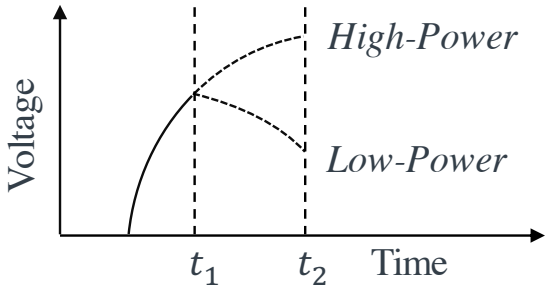


FIGURE 5.5: Illustration of high/low power is determined using an ADC during startup routine. At t_1 the ADC takes a reading and the CPU begins a test computation. At t_2 another reading is taken, and high/low-power can be established.

The system should always save state before power failure, and this is confirmed with a flag triggered at the end of checkpointing and cleared on restore. If this flag is not set, the checkpoint did not complete and the application must restart. V_h and V_s are also increased

to trigger hibernation earlier on subsequent power failures, whilst there remains greater energy stored in the capacitance.

When the supply voltage drops below the calibrated threshold, V_s, the system enters sleep, expecting supply recovery. As explained, this only occurs for low-power sources, however P_{active} is orders of magnitude higher than P_{sleep}, so a *low-power* (Eq 5.2) supply can occur frequently. The MSP430FR5739 used in Hibernus++ and throughout this Chapter has a number of low-power modes that retain data in registers and RAM with low current consumption (6µA in LPM4). [111]. The trade-off with these different modes is the trade-off between latency when switching to/from active mode and the current consumption. LPM4 is used in this evaluation, and it is herein referred to as sleep.

If the supply voltage recovers, the system again waits for sufficient charge on the capacitor. If the supply instead declines below V_h, indicating there is *insufficient power*, the system wakes, hibernates and returns to sleep. If *insufficient power* continues, the voltage will drop below V_{min} and the system will shut down, corrupting the VM. Provided its contents are safely stored in NVM, this will not matter.

PowerNapping has a safeguard against missing a state-save. Existing intermittent computing systems monitor the restore threshold when sleeping, but with PowerNapping the hibernate threshold is also monitored. This guarantees that when in sleep mode data will not be lost, even on sudden power-failure, whilst also waiting for the optimal restore voltage. The need to monitor a second threshold is the primary overhead of this scheme. After a state-save, the system can safely lose power without data loss, therefore only the restore threshold is monitored.

5.4.1 Efficient Threshold Detection

PowerNapping achieves a significant increase in the forward progress of computations on intermittent computing devices, however it requires the use of an additional voltage monitor to alert the system to falling or increasing voltage simultaneously. There are several approaches discussed in the literature such as ADC, internal comparator, or external voltage threshold. None of these approaches are discussed in great detail, or significant effort dedicated to improving them, aside from a recent paper published by Williams et al. [121] called failure sentinels. They propose a novel approach that utilises ring oscillator frequency to track voltage, however the improvement they state is over poor implementations found across the field of intermittent computing. Monitoring the energy is critical for reactive systems, and there is certainly scope for improvement in the performance of voltage monitoring using commercially available solutions, and even more so as these devices transition to SoC implementation.

In order to best utilise low-power sources, energy savings must be made across the whole system. This implementation uses an external comparator that consumes 1 µA and an

internal comparator that consumes 75 µA. PowerNapping requires both simultaneously. Two external comparators could be used in future implementations.

If re-designed for SoC implementation, more efficient comparators could be used, drastically reducing the main overhead of PowerNapping. Table 5.2 gives the power and voltage range of a number of voltage threshold monitoring circuits demonstrating that 23 nW threshold overheads are possible.

TABLE 5.2: Comparison of voltage threshold monitor circuits reproduced from [19]

Author	Caicedo [19]	Mattia [78]	Mattia [77]	Vlassis [115]	Wang [119]	Sengupta [98]
Voltage range (V)	0.97-3	0.6-1.8	0.6-1.2	1-3.6	1.9-2.1	2-2.5
Power	57nW	23nW	23nW	50µW	290µW	388µW

5.4.2 Static vs. Fixed

A key benefit of Hibernus++ and the proposed PowerNapping scheme is that the voltage thresholds are variable. This allows the system to adapt depending on the capacitance and consumption of the experimenter board. It would also mean that schemes with a variable checkpoint overhead, e.g. due to the stack growth investigated in Chapter 3, it could adapt the threshold dynamically.

On the other hand, fixed threshold voltage monitoring can be done with much simpler electronics i.e. fewer transistors, and therefore the size, cost and quiescent energy consumption can all be reduced. Therefore it is worth stating that a trade-off exists with the choice of voltage monitoring, particularly if two are required as in PowerNapping.

As the use of intermittent computing schemes extends to mass-produced or SoC implementations where the capacitance, energy consumption and NVM technology are all well understood and consistent board to board, then a static approach to threshold monitoring can result in a much lower power consumption. To implement the PowerNapping approach in this way would require three voltage monitors for sleep, hibernate and restore despite only two being monitored simultaneously.

Furthermore, for variable voltage thresholds, both the restore and hibernation threshold can be tracked with a single comparator. The system designer doesn't need to do any pre-characterisation of the on-board capacitance, including the decoupling capacitance, find the energy consumed during checkpointing and perform the calculation of Equation 2.1. They can continue to adapt to the program that they are running, and add peripherals and the calibration routine will handle the changes. This explains the choice of a variable approach in this scheme.

There exists significant scope for further research in the way that voltage threshold monitoring is done for reactive intermittent computing systems. Particularly as the

inclusion of additional hardware is one of the most commonly cited negatives of these schemes.

5.5 Mathematical Analysis

In the Hibernus++ implementation, checkpoints were taken every time the supply voltage dropped below a certain threshold. For *low-power* sources such as PV cells, the supply voltage often recovers immediately following the checkpoint routine. To maximise active time for useful computation, the aim is to reduce the time overhead of performing unnecessary state saves. Equation 5.6 shows the time available for making forward progress when a given supply profile leads to a total system on-time, T_{on}. This will be dependent on the number of supply interruptions. This can be broken into interruptions that are:

1. avoidable (n_{ia}) - *low-power*, where the supply voltage would not drop below V_{min} if sleep was entered by the CPU.
2. unavoidable (n_{iu}) - *insufficient power*, supply will certainly drop below V_{min} regardless of CPU state.

$$T_{\phi} = T_{on} - T_a - n_{ia}(T_h + T_{sw} + T_{\lambda}) - n_{iu}(T_h + T_r + T_{\lambda}) \tag{5.6}$$

Where T_{ϕ} is the forward progress for a baseline intermittent approach, T_a is the time overhead introduced by the intermittent compute algorithm, T_h is the time taken to hibernate, T_{sw} is the overhead of sleeping and waking, T_r is the time to restore system state and T_{λ} is the time spent sleeping or shut down before restoring.

The equivalent for PowerNapping is given by Equation 5.7:

$$\begin{aligned} T_{\phi pn} = T_{on} - T_{apn} - n_{ia}(T_{sw} + T_{\lambda} + T_{\lambda pn}) \\ - n_{iu}(T_h + T_r + T_{\lambda} + T_{sw}) \end{aligned} \tag{5.7}$$

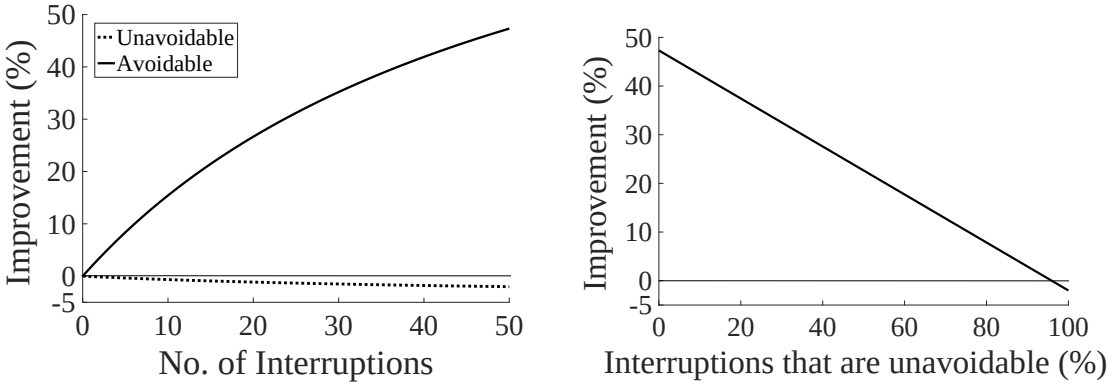
Where $T_{\phi pn}$ is the forward progress when implementing PowerNapping , T_{apn} is the time overhead of the intermittent compute algorithm when including PowerNapping and $T_{\lambda pn}$ is the additional time in sleep due to PowerNapping. For this system to improve upon the baseline intermittent approach, it requires $T_{\phi pn} > T_{\phi}$. For a given time, T_{on}, by removing equivalent terms the difference in forward progress between PowerNapping and the baseline can be seen:

$$\begin{aligned} T_{\phi pn} - T_{\phi} = (T_a + n_{ia}(T_h)) \\ -(T_{apn} + n_{ia}(T_{\lambda pn}) + n_{iu}(T_{sw})) \end{aligned} \tag{5.8}$$

The algorithm time difference, $T_a - T_{apn}$, is negligible. Using an MSP430FR5739 experimenter board T_h and T_r are found to be $\approx$ 1.87ms and from the data sheet T_{sw} is 78μs for LPM4. Equation 5.9 demonstrates how the improvement is dependent on the quantity of n_{iu} and n_{ia}.

$$T_{\phi pn} - T_{\phi} = n_{ia}(T_h - T_{\lambda pn}) - n_{iu}(T_{sw}) \tag{5.9}$$

This relationship is visualised in Figure 5.6, demonstrating that PowerNapping is most beneficial in situations where V_{min} is unlikely to be reached i.e. *low-power* sources, where $n_{ia} >> n_{iu}$. This benefit increases proportionally with the number of interruptions for a given time, T_{on}. When the majority of interruptions are unavoidable, as for intermittent *high-power* sources, PowerNapping is no longer beneficial and will be disabled. Figure 5.6b shows that for 50 interruptions PowerNapping increases forward progress until > 96% of interruptions are unavoidable.



(A) Improvement with increasing n_{ia} and n_{iu} independently

(B) For 50 interruptions, n, impact of increasing percentage that are unavoidable

FIGURE 5.6: Improvement of PowerNapping with changing n_{ia} and n_{iu} over 50 interruptions

If the checkpoint is avoidable, then all the energy that would have been wasted on a state save is retained. From the MSP430FR5739 data sheet, E_{α} = 4.2 nJ/byte and E_{β} = 2.7 nJ/byte, with a total RAM size of 1024 bytes and register size of 512 bytes. Compared with a complete state save, there should be an additional 5.7µJ of energy available for forward progress. $T_{\lambda pn}$ is minimised by having the PowerNapping threshold (V_s) as close to the hibernate threshold (V_h) as possible, achieved at runtime using the calibration routine described previously.

5.6 Simulation

The objective of PowerNapping is to increase forward progress for *low-power* supplies. To verify the behaviour, and predict performance, mathematical simulation was used. Energy consumption for active, sleep, hibernation and restore were obtained from both the MSP430FR5739 data sheet [111] and physical characterisation. MATLAB was used to run these values through the mathematical models with various test conditions and input sources as demonstrated below.

Figure 5.7 shows the results of a simulation of the system response to an ideal 200 µA source that drops to 0 µA at ~600 ms. This is the worst case test, as typically *low-power* EH sources have slower current variation over time. An example case where this could occur is an indoor PV cell, powered with a consistent incoming light, having the light suddenly switched off.

Each time the supply drops to the threshold V_h the Hibernus++ algorithm saves state; moving the contents of RAM, registers and CPU state to FRAM, using energy from the decoupling capacitance of the system. The checkpointing routine completes before reaching V_{min} (2 V in this case) at which point the system enters sleep and the supply recovers.

PowerNapping instead enters sleep immediately and therefore recovers more quickly. For this supply profile, PowerNapping is simulated to increase the forward progress by 24.4%, because less energy is wasted on saving state. When the supply does drop below V_h, due to *insufficient power* during sleep, the system is able to wake and hibernate as seen around 1400 ms.

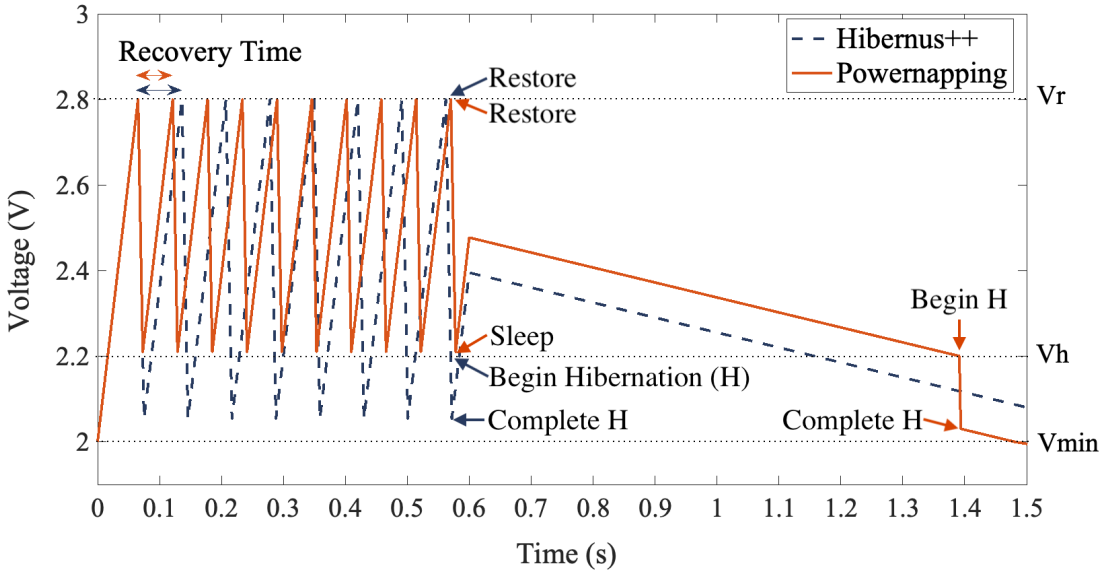


FIGURE 5.7: Comparison of Hibernus++ and PowerNapping for a constant current source interrupted at ~600ms

These two approaches were also compared using a single diode PV cell model [29] as the supply input as described in subsection 2.5.1. PV cells are low current DC sources, but do not supply a perfectly constant current. The current varies according to the I-V characteristics of the cell. A cell was selected with a maximum power point (MPP) voltage within the operating voltage range of the MSP430. The model uses the open-circuit voltage, short circuit current, MPP voltage and MPP current to generate an I-V curve, fed into the MSP430 MATLAB model. With the PV cell, PowerNapping increased forward progress by 18.3% with ~300 µA supply and 27.5% with ~100 µA supply.

The PowerNapping strategy should allow the supply voltage to recover without saving system state, but in the case where the supply does not recover, will save state successfully. From simulation, it is expected that the lower the input current, the greater the benefit of PowerNapping. Wasteful state saves occur more frequently for Hibernus++ in this case, since the lower supply current leads to faster discharging of the capacitance when making forward progress, increasing the number of avoidable interruptions.

PowerNapping has little overhead. The system characterises the source on start-up. For bursty *high-power* supplies, where the supply typically dies without recovery, PowerNapping is disabled and there is no additional overhead. When PowerNapping is utilised, the overhead is (1) an increase to the hibernation time due to sleeping first, and (2) the inclusion of a second voltage threshold monitoring circuit using the internal comparator. This overhead is included in simulation and the physical test setup.

It has been shown in simulation that decreasing the number of resource intensive hibernations leads to an increase in forward progress. This increase in forward progress leads to faster program execution, and a more efficient system. Improvements would therefore demonstrate an increased active time, and reduced completion time for benchmarks on a physical system, as demonstrated in Section 5.7.

5.7 Experimental Validation

5.7.1 Experimental Method

To verify that the decreased time and energy overhead of entering a sleep state instead of saving state results in improvement to the forward progress of the application, and that the more interruptions, the greater the improvement, a number of different tests were performed.

The adapted Hibernus++ code with PowerNapping included was programmed onto an MSP430FR5739 experimenter board. This was connected to an external threshold detection circuit, monitoring V_h or V_r respectively. The internal comparator is only used for V_s. The experimental setup is shown in Figure 4.7.

The experimental setup is the same for PowerNapping as it was for Expedit. The only change is the inclusion of a second voltage threshold using the internal comparator of the MSP430FR5739 experimenter board. FFT is used as a benchmark for comparing forward progress with a range of supply conditions. Additional benchmarks are also included to demonstrate the performance of this scheme across different compute loads. Because the entire contents of RAM and registers are saved during hibernation, time and energy overheads of hibernation and restore are not application dependent.

The internal comparator is only active when in sleep, so there is additional power consumption in this state. For a 2 V supply, the internal comparator consumes 75 µA whereas the external comparator consumes 1 µA. PowerNapping requires both simultaneously. Two external comparators could be used in future implementations as described in Subsection

5.7.2 Results

Table 5.3 compares the restore, hibernate and sleep of PowerNapping against Hibernus++. These remained constant across all benchmarks presented.

TABLE 5.3: Total overhead comparison of PowerNapping (PN) against Hibernus++ (H++)

(A) Time to complete listed functions

	H++ (µ*s*)	PN (µ*s*)
Restore	1874	1876
Hibernate	1876	1878
Sleep	N/A	166

(B) Energy to complete listed functions

	H++ (µJ)	PN (µJ)
Restore	7.30	7.31
Hibernate	7.31	7.32
Sleep	N/A	0.65

The expectation is that PowerNapping makes little change to the overhead, but implements a sleep state that is reached much more quickly than hibernation. This is shown with PowerNapping's sleep, which operates only when the harvester is generating a *low-power* supply, being 88.3% faster than hibernating.

PowerNapping is most effective for slow-varying *low-power* sources such as PV cell power outputs. Figure 5.8 compares Hibernus++ (H++) and PowerNapping (PN) with a 10x3cm PV cell under 500 lux indoor light, to demonstrate how state saves impact active time. This delivers ~300µA of current. PowerNapping allows the processor to remain active for longer, seen by the increased width of forward progress, because it reduces costly hibernations. The decoupling capacitance charges earlier, leading to more frequent periods of forward progress, because less energy is spent on state saves. This leads to improvement in the completion time of the FFT.

Figure 5.9 demonstrates this for a range of source conditions. For *low-power* supplies PowerNapping allows immediate supply recovery, increasing forward progress. The restore

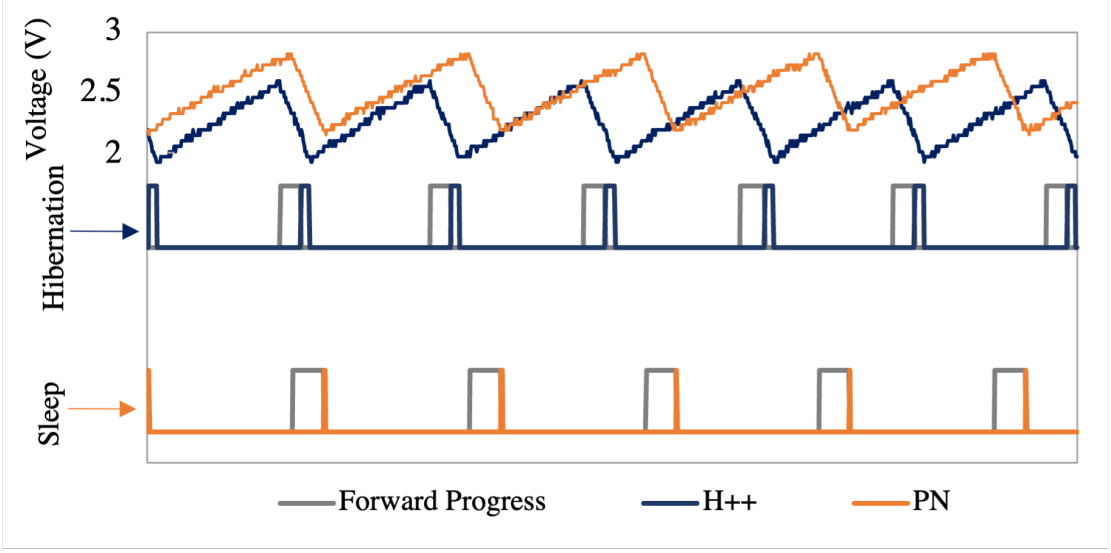


FIGURE 5.8: Comparison of PowerNapping and Hibernus++ with PV cell input energy source, showing an increase in forward progress for PowerNapping.

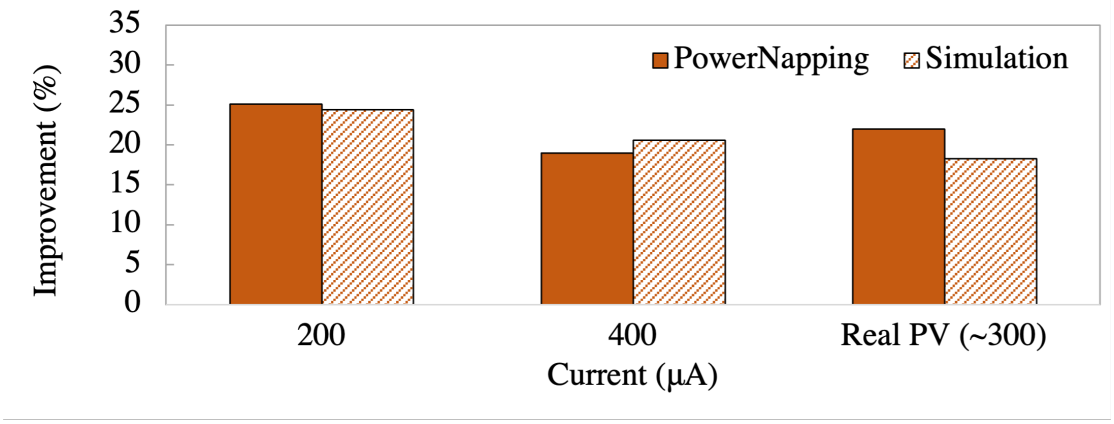


FIGURE 5.9: Improvement in FFT complete time for PowerNapping over Hibernus++ including simulation and practical validation with low-power sources

routine isn't utilised for *low-power* supplies, since the supply voltage doesn't drop below V_{min}, and volatile data is not lost. PowerNapping removes resource-intensive hibernations by instead entering sleep. This occurs much more quickly as seen in Table 5.3a, leading to this improvement in the overall completion time. For lower currents this improvement exceeds the expectation from simulation. In simulation, V_r is approximated as a fixed value, however as explained in Section 5.4, V_r varies according to the source characteristics. At lower currents, the capacitor charges more slowly, leading to V_r being triggered earlier and consequently more frequent hibernations. This favours PowerNapping with its lower overhead.

Figure 5.10 compares the total on-time, T_{on}, of the processor when completing three FFTs. Removing time when supply is not available helps clarify the comparison. PowerNapping is not intended to give an improvement for high-power supplies, due to being disabled, however, the results demonstrate that with high-frequency interruptions there is significant

improvement: up to 28.9% for PowerNapping. When run with a stringsearch benchmark, the same experiment resulted in 46.8% improvement.

This is different to the expectations from simulation, and is actually a bug relating to the test setup. At these high frequencies, the supply is interrupted faster than the ADC startup test can occur, but buffered by the decoupling capacitance, resulting in the system registering *low-power* and enabling PowerNapping, unlike in simulation (Section 5.6). The interruptions are close enough together that when in low-power mode, as in PowerNapping, the decoupling capacitance is sufficient to prevent the supply dropping below V_{min}. However, when saving state (as Hibernus++ does) the supply does drop below V_{min} and considerable energy is wasted on saving state, resulting in excellent improvement for PowerNapping.

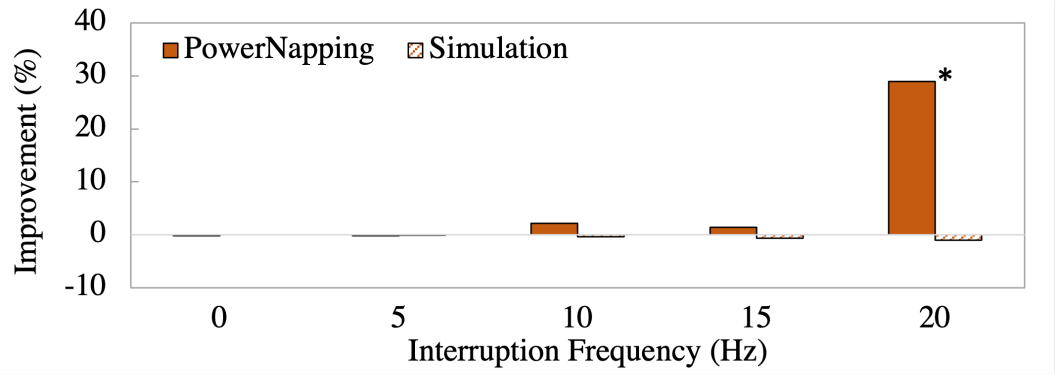


FIGURE 5.10: Improvement in FFT complete time for PowerNapping over Hibernus++including simulation and practical validation with high-power sources. The * signifies a result where PowerNapping is enabled and by entering sleep the system avoids dropping below V_{min} across some interruptions.

The slight improvement of PowerNapping at 10Hz is again, as with Expedit, due to the number of interruptions occurring not being directly proportional to the frequency of interruption. Hibernus++ occasionally experiences an additional cycle or two over PowerNapping, and therefore has a longer completion time. Figure 5.11 compares the improvement of PowerNapping to the number of interruptions rather than the interruption frequency, showing a proportional increase as expected. This was with low-power sources.

5.7.3 Combination with Other Schemes

Since PowerNapping targets *low-power* sources and Expedit *high-power* sources, the combination of the two approaches has also been evaluated with the full range of test conditions present in both chapters. PowerNapping only improves the checkpointing routine, whereas Expedit improves both restore and checkpointing leading to the combined approach having a consistently higher performance than either scheme separately.

Figure 5.13 shows an even more significant improvement: up to 28.9% for PowerNapping alone, 50% when combined with the faster restore time of Expedit.

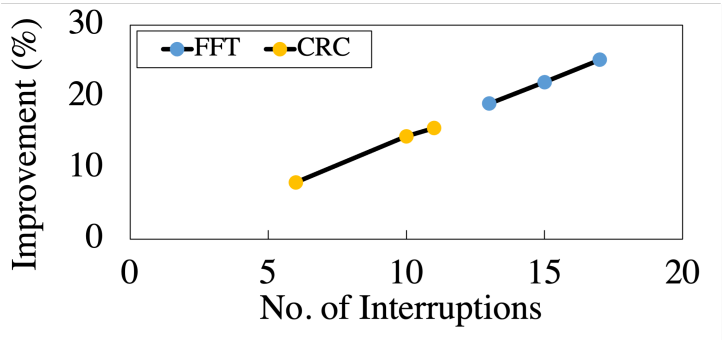


FIGURE 5.11: Improvement of PowerNapping over Hibernus++ with respect to interruptions for FFT and CRC

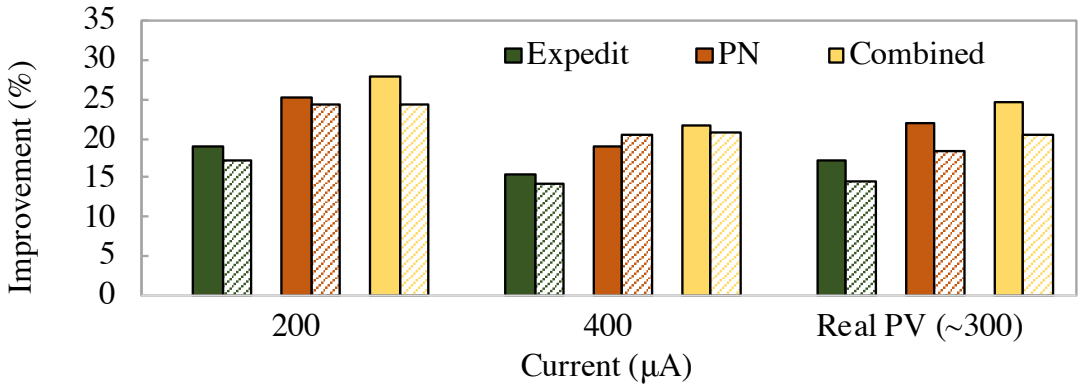


FIGURE 5.12: Improvement in FFT complete time for the two schemes over Hibernus++ for simulation and practical validation with *low-power* sources. Solid colour represents practical validation, and cross-hatch, simulation results.

Although PowerNapping was not designed for high-power sources, and should be disabled for them, Figure 5.10 indicates that the low-current consumption of sleep allowed the decoupling capacitance to sustain the voltage above V_{min} until power returned, resulting in an improvement in forward progress. If PowerNapping was combined with increased energy storage then this same effect would occur at lower frequencies, allowing PowerNapping to further increase the forward progress for a greater range of high-power sources. If the interruption frequency could be determined at design time, increased capacitance may allow PowerNapping to enable the voltage to be sustained across power failures. Furthermore, Zhan et al. [125] suggest that altering the size of the capacitance can lead to increased forward progress because power failures can be avoided. These results demonstrate that PowerNapping can also result in the avoidance of power failures due to the lowered power consumption. In combination with the approach from Zhan et al., PowerNapping may be able to further improve the forward progress for frequently interrupting power sources. The drawbacks of increased capacitance remain, and so the application requirements must be considered.

It is therefore recommended that these two schemes are implemented in combination for low-power sources and consideration given for high-power sources. For low-power sources,

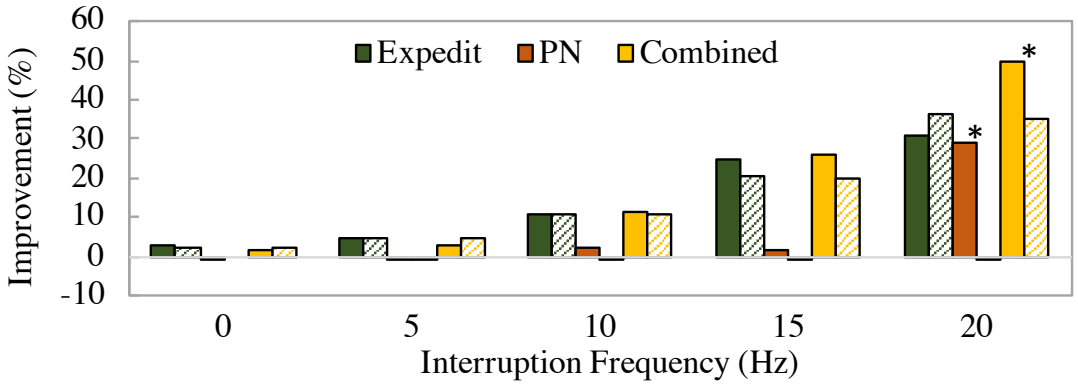


FIGURE 5.13: Improvement in FFT complete time for the two schemes over Hibernus++ for simulation and practical validation with *high-power* sources. The solid colour represents practical validation, and the cross-hatch is simulation results. The * signifies a result where PowerNapping is enabled and by entering sleep the system avoids dropping below V_{min} across some interruptions.

PowerNapping improves the performance by avoiding costly checkpoints altogether, and Expedit improves both the checkpoint and restore overheads as demonstrated in Figure 5.12 which shows an improvement of more than 25% with a combined approach. Conversely, when using high-power sources only Expedit was designed to improved the performance however as noted in the paragraph above in certain circumstances, PowerNapping can allow the voltage to be sustained above V_{min}. Ordinarily PowerNapping is disabled by the software at runtime as described in section 5.4, so can be included in any system without increasing energy overheads. For this reason, the combined approach may be beneficial in all circumstances. Expedit alone may be selected if the code memory is restricted, as PowerNapping increases this slightly, or if a second voltage threshold circuit cannot be included due to cost or hardware restrictions.

As discussed in Chapter 3, application agnosticism is not sustainable in the future design of intermittent computing schemes. The software used can have a significant impact on their effectiveness. The implementation of PowerNapping in this analysis is an extension of Hibernus++ which is a copy-all reactive approach, moving the entire contents of RAM. For this reason, it was expected that the software would not have an impact on the efficiency of this scheme. However, it is important to validate this by implementing additional benchmarks. The work in Chapter 3 was completed with 64Kb of RAM available, however the MSP430FR5739 experimenter board used in this experiment only had 16kb available, resulting in a smaller selection of benchmarks tested here. This is why many authors now opt for the MSP430FR5969 with its larger RAM. The scheme tested here also includes Expedit, and is compared to Hibernus++.

Figure 5.14 shows that PowerNapping in combination with Expedit provides improvement for a range of IoT benchmarks, giving up to 35.2%, 50.0% and 46.8% improvement respectively. With the addition of both Expedit and PowerNapping, Hibernus++ is improved for both high and low-power sources. As shown in Figure 5.6a, greater benefits are gained

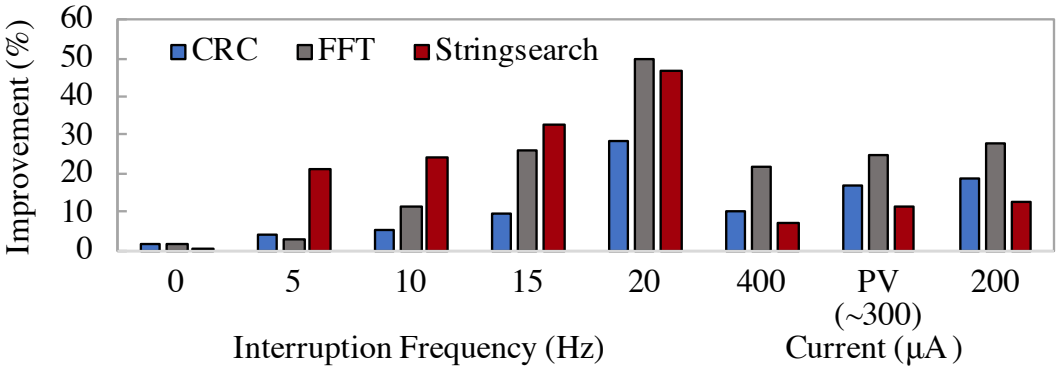


FIGURE 5.14: Improvement of Expedit and PowerNapping combined for FFT, CRC and Stringsearch with high and low-power sources

when more interruptions occur during a compute cycle, so results vary across benchmarks. Stringsearch takes 1045ms to complete, much longer than CRC at 103.8ms, and therefore more interruptions occur, across all frequencies, hence a greater improvement is observed for *high-power* sources. Conversely for *low-power* sources, longer base completion time means that hibernations make up a smaller percentage of total completion time, leading to lower percentage improvement being observed.

5.8 Summary and Discussion

Push-through was proposed as a solution to benchmarks with a high checkpointing overhead to completion time ratio, such as CEM described in section 3.4.2, however due to the runtime energy awareness costs, an alternative scheme, PowerNapping, is proposed. PowerNapping, instead of trying to push a task through to completion, or make a costly state save, enters a low-power mode to wait for the supply to recover without saving state.

PowerNapping allows time and energy previously wasted on state saves to be used to increase the forward progress of the system. Improvements are targeted at *low-power* supplies, with up to 28.8% improvement in forward progress demonstrated over the state-of-the-art for these sources.

At high interruption frequencies, with *high-power* sources, PowerNapping can achieve up to 46.8% improvement. The benefits of PowerNapping have been shown experimentally with both ideal and real EH sources across multiple benchmarks. On the whole, results align with those expected from the preliminary simulation and mathematical analysis, with PowerNapping being more effective at higher interruption frequencies than expected. This opens the opportunity for PowerNapping to be combined with an increased energy buffer in future work, to increase the performance for high-power sources, not just the low-power sources it was designed for as discussed in section 5.7.3.

Tested with Hibernus++ [7] in this work, PowerNapping demonstrates a reduction in overheads that would be beneficial to other reactive transient works such as QuickRecall [55], or task-based approaches where large checkpoints occur. This means that although the results here demonstrate the efficacy of the scheme, and a significant improvement over the state-of-the-art for reactive checkpointing, the contribution is significant for the broader field of intermittent computing. PowerNapping implemented on a SoC is expected to demonstrate an even greater improvement since the enhanced power-gating could further reduce the sleep power consumption, and the power draw of the voltage threshold monitoring circuits are much lower as shown in Table 5.2.

Increased performance isn't limited to more complex applications being able to be implemented, or progress being made more quickly, but also that the same application can be implemented on a more tightly constrained system. PowerNapping allows devices with supplies that are significantly lower power than P_{active} but above P_{sleep} to operate and acheive forward progress without a significant, and unsustainable, number of state saves being required.

Chapter 6

Conclusions

6.1 Conclusions

To realise the goal of a trillion IoT devices, battery-less devices with their lower size, mass and cost are an important area of research and development. Intermittent computing allows a range of EH sources to be used without additional storage. Reactive approaches can be instrumented within interrupt routines, allowing the complexity to be abstracted from the developer, further enabling the uptake of intermittent computing devices.

As these systems target SoC implementations for cheaper mass manufacture, further design considerations are necessary such as selecting the appropriate NVM technology, and the impact of a reduced area for capacitance. It was demonstrated through the use of RTL simulation that memory use varies significantly between applications and during execution. The following chapters present schemes that do reduce time and energy overheads, whilst also being applicable to all applications, by directly targeting the nature and frequency of state-saves. In this way even if the overall schemes cannot be made application agnostic, there is scope for improvements in forward progress without diving into specific software development changes.

With the two schemes, PowerNapping and Expedit, it has been demonstrated that the forward progress of reactive intermittent computing systems can be improved.

PowerNapping allows time and energy previously wasted on hibernation to be used to increase the forward progress of the system, with sleep reducing the time and energy overheads of checkpointing by up to 88.3%. Improvements are aimed at *low-power* supplies, giving up to 28.8% improvement in forward progress over the state-of-the-art. Though not originally intended, it is discovered that at high interruption frequencies, with *high-power* sources, PowerNapping can achieve up to 46.8% improvement since the sleep state allows the input to be smoothed with just the decoupling capacitance. Expedit reduced the time

and energy overheads of saving state by up to 83.6% and 85.6% respectively. This gives up to a 84.4% improvement in completion time over the state-of-the-art.

The benefits of these two schemes have been shown experimentally with both ideal and real EH sources across multiple benchmarks. On the whole, results align with those expected from the simulation, with the schemes being more effective at higher interruption frequencies, or with lower current inputs, both of which cause more checkpoints to occur.

In combination they bring improvement with a full range of high and low power supplies of up to 50.0%. Tested with Hibernus++ [9] in this work, these schemes demonstrate a reduction in overheads that would be beneficial to other reactive intermittent computing works such as QuickRecall [55], or task-based approaches where large checkpoints occur.

6.2 Future Work

The research presented in this thesis increased the efficiency of reactive intermittent computing systems. However, there is room for more research in this area until widespread adoption is achieved.

NVM-specific intermittent computing design Further research into the use of a range of NVMs, and experiments for intermittent computing designs that have been optimised for them could yield improved forward progress for those NVM technologies specifically. An example being that some NVMs have asymmetric read/write costs. If writing is more expensive than reading an additional compression of the checkpoint data before saving state could be more efficient, and result in a lower energy overhead, since reads are cheaper than writes. This would need practical validation of the improvement to the checkpointing energy overhead, and the energy cost of the compression. Determining additional approaches could begin with testing of a range of commercially available experimenter boards, and discrete NVM components with several intermittent computing approaches, and the completion of a survey. Similarly, for SoC implementations, this could involve further research with RTL simulation and corresponding NVM technologies accurately simulated and compared. Furthermore, the characteristics of NVM technologies could be contrasted with the features of existing intermittent computing schemes. This would either result in different schemes recommended according the to NVM technology used, or novel improvements for certain NVM technologies. This future work would build upon section 3.3.1, which gave an introduction to the choices available for SoC designers of intermittent systems, and showed that existing works almost exclusively use FRAM experimenter boards and other commercially available technologies.

Architecture adaptations for fault tolerant MCUs The focus of NVPs research is intermittency at the register-level, however intermittency could continue to be handled at the system-level, as in this research, but instead adapt the processor architecture to accept certain faults, simplifying the intermittent software development. Section 3.5.1 explained the architectural IPSR fault neglected from the FUSED environment that was resolved with a manual un-stacking from the interrupt. CPU architectures are not designed with intermittent computing in mind, and this created challenges during this research. Architectures such as Armv6-M found in the Cortex-M0 consider the faults created by a power-loss intolerable, so research into the faults generated by intermittency, and subsequently redesigning the architecture to accept them could streamline the state-save and restore process, avoiding the error found in section 3.5.1 and other potential faults. Furthermore, whilst leading reactive intermittent computing approaches instrument the transfer between non-volatile and volatile memory in software interrupt routines, future work could handle these within the architecture. This would remove the need to compile additional libraries for intermittent operation of software. The intermittent computing software is written with detailed understanding of the architecture, and so this work would involve checking through each stage of the checkpoint and restore routines and asking the question, could this be done more efficiently within the architecture or is this necessary due to an architectural feature that could be removed, i.e. a fault state.

Optimised or redesigned DMA Instead of using the DMA available on the test platform, research into a DMA optimised or redesigned specifically for the regular behaviour of saving state and restoring could further reduce overheads. Chapter 4 introduced Expedit, using such a commercial DMA to significantly improve the efficiency of an intermittent computing approach. Expedit uses a standard DMA that requires initialisation to handle a range of address locations, data lengths, and operation types, such as block-burst. Since the DMA is used identically each time, flexibility is not required. A DMA could be redesigned to handle this consistent operation more efficiently by simplifying the hardware to only take a single set of input conditions. This would be more efficient than using a commercial DMA, and significantly more efficient than handling the operation within the CPU. It may also be possible to reduce the software overhead required to operate the DMA as the initialisation could be simplified.

Combined task-based and reactive intermittent computing approach Section 5.2 introduced a design that would combine task-based and reactive intermittent computing features into one design. Although the merits of reactive checkpointing have inspired the work within this thesis to focus on this approach specifically, future combined approaches could prove to be even more effective. The benefits of the small overheads of the task-based approaches and the simplicity and lack of re-execution cost of reactive checkpointing schemes within a single design could be more effective than any single approach.

Comparing the successes and failures of each scheme, and looking for opportunities to combine the approaches when a success and failure overlap should be the initial research direction. From this, novel approaches could be developed.

Physical deployments Finally, Afanasov et al. [3] have provided one of the few real-world long-term deployments of intermittent computing systems, demonstrating some exciting results. This thesis discussed the need for a greater range of real sources and applications to truly demonstrate the benefits and performance of intermittent computing schemes. Physical deployments are one way that this could be achieved in future, and it is hoped that more research will appear in this area, and that the uptake of intermittent computing will help fuel the path to one trillion devices.

Appendix A

Published Papers

Improving the Forward Progress of Transient Systems

Tim Daulby, Anand Savanth, *Member, IEEE,* Geoff V. Merrett, *Senior Member, IEEE*, and Alex S. Weddell, *Member, IEEE*

***Abstract*—Emerging applications for Internet of Things devices demand smaller mass, size and cost whilst increasing capability and reliability. Energy harvesting can provide power to these ultra-constrained devices, but introduces unreliability, unpredictability and intermittency. Schemes for wireless sensors without batteries or supercapacitors overcome intermittency through saving system state into non-volatile memory before the supply drops below the minimum operating voltage, termed transient or intermittent computing. However, this introduces significant time and energy overheads. This paper presents two schemes that significantly reduce these overheads: entering a sleep mode to avoid saving state and utilising direct memory access (DMA) when state saves are required. Time and energy previously wasted on state saves can instead be used to perform useful computation, termed "forward progress". We practically validate the proposed approaches across a range of energy sources and IoT benchmarks and demonstrate up to 46.8% and 40.3% increase in forward progress and up to 91.1% and 85.6% reduction in overheads for each scheme respectively.**

***Index Terms*—Embedded systems, energy harvesting, low-power design, transient computing, intermittent computing**

I. Introduction

THE Internet of Things (IoT) is a fast expanding and developing field, ranging from connected homes to concepts of smart cities with a unity of physical world and cloud [1]. A key area of growth is ultra low power sensor devices [2]. To maximise the potential deployment opportunities for these devices, they must be autonomously powered and long-running. Some deployments necessitate a small cost, size and mass; examples include biomedical implants [3], data-rich radio frequency identification and structural monitoring. Powering these types of devices is a significant challenge. Batteries alone incur high maintenance overheads with frequent replacement or charging, or must be much larger, which is impractical. This has prompted developments in energy harvesting (EH) [4].

There are a range of approaches for incorporating EH into IoT devices. Energy neutral computing [5] matches the system demand to the incoming power over a given time period by buffering energy. However, these systems depend on rechargeable batteries that suffer from charge-cycling problems [6] or supercapacitors that increase the size, mass and cost. Transient (or intermittent) computing (Section II) is a new paradigm, powering devices from harvested energy without batteries or supercapacitors. They operate directly from the supply; processing, storing and sending data only when power is available.

A leading transient computing approach is the use of checkpointing and copying a snapshot of the system state to a separate non-volatile memory (NVM), first demonstrated for transient systems by Mementos [7]. At a checkpoint, data is copied from volatile memory, including registers, to a NVM before power failure, incurring an energy (E_{ss}) and time overhead. This data is then copied back when restoring system state when power returns. The smaller these overheads, the more energy and time can be put towards forward progress, that is computation beneficial to the progress of the active applications. To maximise the energy available for forward progress, E_{fp}, all other overheads must be reduced. Equation 1 shows this for a harvested energy budget, E_{harv}.

$$E_{fp} = E_{harv} - (n_{ss} \cdot E_{ss} + E_w + E_R) \tag{1}$$

Where energy is spent on: forward progress, E_{fp}; a number, n_{ss}, of snapshots (copying data to NVM), E_{ss}; wasted time due to re-execution[1], E_w; and restoring system state, E_R. This establishes a number of trade-offs. Some works, such as Clank [8], have frequent small checkpoints giving a high n_{ss} but low E_{ss}. Other works, such as Hibernus [9], aim for a single state-save per power cycle, greatly reducing n_{ss} and eliminating E_w, but increasing the size of the checkpoint, and therefore the energy to save it, E_{ss}. This single save is achieved by monitoring the supply voltage, checkpointing only when it drops below a set threshold close to the minimum operating point of the processor. This is known as a *reactive checkpointing* approach because it responds to a change in supply voltage, and the single state-save per power-cycle is known as *hibernation*.

EH supplies are typically unreliable, unpredictable and dependent on the environment they are deployed in, and transient systems have been designed to overcome this [10]. However, some EH sources do have predictable traits. Solar and thermoelectric harvesters typically have a slow-varying DC output, particularly outdoor solar. Wind and vibration

Manuscript received September 19, 2019. Accepted May 17 2020. This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/P010164/1.
T. Daulby, G. Merrett and A. Weddell are with the School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, U.K. (e-mail: td2g14@soton.ac.uk)
A. Savanth is with both the University of Southampton and Arm Ltd., Cambridge CB1 9NJ, U.K.
Experimental data used in this article can be found at: https://doi.org/10.5258/SOTON/D1394
Colour versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

[1]Re-execution is repeating processing that was lost due to power failure occuring whilst data was held only in volatile memory.

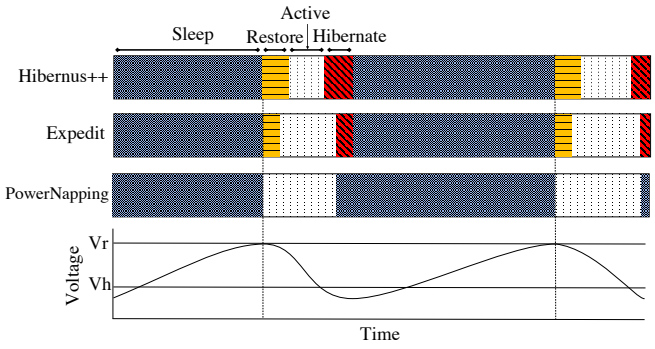


Fig. 1. Illustration of system operation for a given supply with the proposed schemes. Hibernus++ and Expedit restore at a voltage threshold, V_r, and hibernate at V_h. PowerNapping, when enabled, instead of hibernating enters a sleep mode and retains data in volatile memory, simply resuming at V_r.

harvesters may have a more intermittent AC output. Existing reactive transient systems do not account for these traits, and checkpoint when the supply voltage is low, despite the fact that in many cases the voltage would recover naturally, making the checkpoint redundant, if computation was simply suspended in favour of sleep.

This work makes two novel contributions to increase the forward progress of existing transient systems:

1) **PowerNapping.** Works alongside existing reactive checkpointing routines to enter a sleep mode **before** checkpointing. For consistent low-power harvested supplies, e.g. indoor solar, hibernations are avoided, reducing n_{ss} and therefore increasing E_{fp}. Additional safeguards are included to avoid missing a state-save on total power failure (Sec III).
2) **Expedit.** Uses a direct memory access (DMA) controller to handle state-saves and restores, reducing the time and energy overheads of each routine (Sec IV). Decreased E_{ss} and E_R leads to increased E_{fp}.

Figure 1 gives an illustration of the way active time is increased through these two contributions for a voltage trace that decreases when the processor is active, or hibernating/restoring and increases when in sleep. PowerNapping enables supply recovery without resource-intensive state saves, by entering sleep, leading to up to 28.8% improvement in forward progress. Expedit increases the efficiency of hibernation and restore, reducing time and energy overheads, leading to up to 40.3% improvement in forward progress. Both contributions are implemented on an MSP430FR5739 experimenter board with built-in FRAM (ferro-electric RAM) (Sec V)

II. Background and Related Work

The range of transient computing approaches in the literature can typically be classified as non-volatile processors (NVPs), task-based programming models or checkpointing strategies. The ideal case for these systems is retaining all relevant information in memory on every power loss, therefore being able to resume immediately, with no overheads.

NVPs introduce non-volatility at the processor level by developing state-retentive hardware. This involves flip-flop level NVM and parallel backup strategies [11]. NVPs range from one state-save per power interruption, to back-up every cycle [12]. However, these systems are still in the experimental stage and are not yet commercially available. They also have a larger on chip footprint [13], increasing costs. These systems additionally face idempotency problems and in-rush current peaks due to the large parallel back-up [14].

Task-based programming models such as Mayfly [15], Chain [16], Flexicheck [17] and Alpaca [18] overcome intermittency by ensuring atomic computing tasks are completed each power cycle. There is a large burden on the programmer to break the code into tasks and complex code structures.

In contrast, checkpointing-based systems require little additional complexity at compile time, and can be implemented using commercially available hardware. These systems run intermittently, only when harvested power is available, achieving forward progress by saving the system state to NVM before supply failure. When the supply recovers, the system restores its state from NVM, allowing computation to resume from the point it was interrupted. One of the earliest works, Mementos [7], used static checkpoints, configured at compile time on function returns, every loop, with a timer or manually, to test the supply voltage and save state if below a static threshold voltage. This resulted in a number of checkpoints per power cycle, and data consistency violations[2].

Hibernus [9] instead uses a hardware interrupt to prompt a state save immediately before power failure, giving a single checkpoint per power cycle. A hibernate voltage threshold is set, to ensure sufficient energy in the decoupling capacitance of the system (ΣC) to save state before the supply drops below the minimum operating voltage of the processor (V_{min}). The minimum threshold to save state successfully can be determined by setting the energy required equal to the energy in the capacitance, as given below.

$$n_\alpha E_\alpha + n_\beta E_\beta = \frac{V_h^2 - V_{min}^2}{2} \times \sum C \tag{2}$$

Where V_h is the threshold voltage, n_α and n_β are the sizes of the RAM and registers (in bytes). E_α and E_β are the energy required to copy each RAM and register byte to NVM (J/byte).

Energy is not lost on roll-backs, i.e. re-executing code run since the last checkpoint, because the system stops execution, having pre-empted power failure, and waits until the supply voltage exceeds a restore threshold (V_r) or safely dies. This also removes the risk of data consistency violations. Hibernus requires characterisation of the system before compile time. This lack of runtime adjustment could create instability if there is a change to system properties, or a 'safe' hibernate threshold (V_h) must be set to allow changing characteristics at the expense of efficiency. For example changes in the dynamics of the power source, or increased system power consumption due to additional peripherals.

Hibernus++ [19], introduced runtime adjustment of the thresholds (V_r and V_h). When the system is first powered on, it runs a calibration routine that sets V_h, further adjusting if

[2]Data consistency violations can occur when re-initialised volatile values no longer correspond with persistent variables in NVM, and a state is reached that is inconsistent with continuous operation.

a checkpoint ever fails to complete. Hibernus++ anticipates power failure more accurately, but still takes a conservative approach, calibrating and setting voltage thresholds for the worst case power interruption: sudden incoming supply drop to 0 V. This leads to the system hibernating earlier than necessary if supply instead declines slowly. There are further situations where Hibernus++ does not perform well, for example, when the device has a constant low-power source; where the system does not die after checkpointing, but instead begins to recover. If this incoming power is sustained, the data in volatile memory has not been lost, and therefore the effort of taking a checkpoint has been wasted. The MSP430FR5739 used in Hibernus++ and throughout this paper has a number of low-power modes that retain data in registers and RAM with low current consumption ($6\mu A$ in low power mode 4). [20].

It was suggested that this wasteful checkpointing could be solved by introducing a third threshold where the system enters a sleep state without saving state [21]. This could allow the system supply voltage to recover without dropping below V_{min}. This threshold was set statically, as in Hibernus. Additionally, for intermittent high-power sources, where hibernations occur frequently, this technique further reduces system performance. Execution is traded for sleep at a higher voltage, despite hibernation being inevitable, losing forward progress with wasted time and energy.

Hibernus++ remains a key work in this field, with only a few authors re-visiting reactive approaches [22][23]. The key benefit of these reactive systems is that they place little-to-no burden on the programmer, instead allowing standard embedded programs to be compiled and run. They also require little hardware adaptation, unlike NVPs, whilst also avoiding data consistency violations, unlike other checkpointing approaches. The key drawback of reactive systems is their large time and energy overheads compared with other schemes, so this paper demonstrates a clear improvement over the state-of-the-art.

III. PowerNapping

Transient systems are designed to overcome intermittency in the supply, however the nature of this intermittency can vary greatly. We classify three states of harvested power input, P_{Harv}, for the purposes of this paper.

$$P_{Harv} < P_{Slp} : \text{Insufficient power} \quad (3)$$

$$P_{Act} > P_{Harv} > P_{Slp} : \text{Low-power} \quad (4)$$

$$P_{Harv} > P_{Act} : \text{High-power} \quad (5)$$

Where P_{Act} is the active consumption of the device and P_{Slp} is the consumption of the device when in sleep mode. The operating voltage is entirely dependent on the energy stored in the system capacitance. Where the harvested power, P_{Harv}, exceeds the power consumption of the system (P_{Act} or P_{Slp} depending on system state), the energy, and therefore operating voltage, rises; on the other hand, operating voltage will drop, eventually leading to the system responding by sleeping/hibernating. *Low-power* is sufficient to retain state in volatile memory if in sleep mode [20]. If *low-power* is sustained, existing systems still complete resource-intensive hibernations which are unnecessary if the supply can be prevented from dropping below the minimum operating voltage of the processor, V_{min}, with sleep, and therefore retaining volatile data. This energy could otherwise be used to achieve forward progress. Furthermore, for these devices to meet the criteria of small mass, size and cost, the energy harvesters they employ must also be small. These smaller harvesters can be expected to deliver *low-power* more frequently, meaning that an increasing number of checkpoints are taken unnecessarily, since volatile memory contents are not lost. For example a real $1cm^2$ PV cell ranges from $120\mu W$ on a sunny day to $\leq 40\mu W$ when cloudy [24].

We propose pre-emptively triggering sleep instead of hibernation when there is a *low-power* harvested supply. This allows the supply voltage to recover without dropping below the threshold V_h, as defined in Equation 2. The operation of the enhanced PowerNapping approach is detailed in Figure 2. By identifying which power state the system is in (Eq 3-5), and the current supply voltage, this system determines whether it is most effective to sleep, hibernate or restore. High or low power supply is determined at run time by the calibration routine, as explained later, and insufficient power is detected by dropping below V_h during sleep. PowerNapping introduces additional states and decisions as shown within the dotted box, when compared to Hibernus++. This PowerNapping state adaptation could be applied to other schemes such as QuickRecall [25], but Hibernus++ is used throughout this work.

The voltage thresholds V_s, V_h and V_r in Figure 2 are required for reactive checkpointing, and are set at runtime according to the system properties with a calibration routine run when the system is first powered on:

1) An ADC is used to measure voltage (V_1). The supply is then isolated with a diode.
2) The system sleeps and wakes before taking a second reading (V_2).
3) The system hibernates and then takes a third reading (V_3).

The voltage threshold indicating there is sufficient energy for sleep/wake, V_s, is characterised by the voltage drop (V1-V2) and for hibernation, V_h, (V2-V3), giving the following thresholds:

$$V_h = V_{min} + (V_2 - V_3) \quad (6)$$

$$V_s = V_h + (V_1 - V_2) \quad (7)$$

On system start-up, it tests the supply by taking two ADC readings, as shown in Figure 3. If the second is higher than the first, it identifies that there is *a high-power* harvested supply and the system restores state immediately, utilising the 'abundant' power to maximise forward progress. There is no downside to this, since, in a transient system, excess energy cannot be stored. If the second reading is lower, there is a *low-power* supply and the system will wait until there is sufficient charge on the capacitor before restoring. Trying to restore immediately would draw power, resulting in a voltage drop below the hibernate threshold. To avoid this, there is an incrementing voltage and timer interrupt. Whilst the voltage

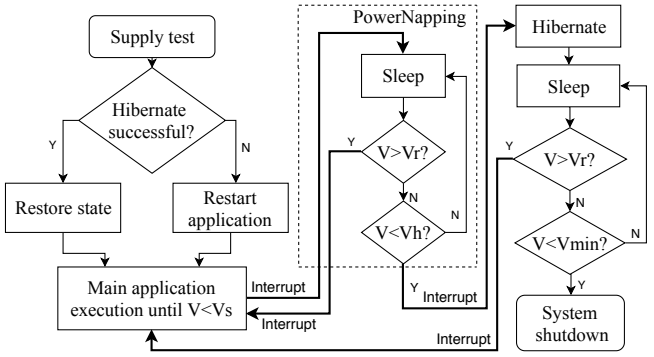


Fig. 2. State diagram of PowerNapping approach

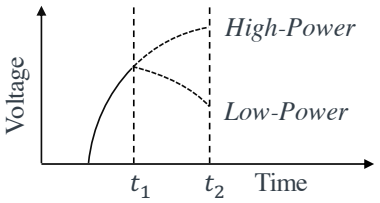


Fig. 3. At t_1 the ADC takes a reading and the CPU begins a test computation. At t_2 another reading is taken, and high/low-power can be established.

interrupts first, the charge is increasing steadily and it is better to continue charging the capacitance. When the timer triggers first, this means that the rate of charge has plateaued and the system restores. The voltage at which this occurs is set as the restore threshold, V_r, used to exit subsequent sleep modes. This will be greater than V_s and V_h.

The system should always save state before power failure, and this is confirmed with a flag triggered at the end of hibernation and cleared on restore. If this flag is not set, the checkpoint did not complete and the application must restart. V_h and V_s are also increased to trigger hibernation earlier on subsequent supply failures, whilst there remains greater energy stored in the capacitance.

When the supply voltage drops below the calibrated threshold, V_s, the system enters sleep, expecting supply recovery. As explained, this only occurs for low-power sources, however P_{Act} is orders of magnitude higher than P_{Slp}, so a *low-power* (Eq 4) supply can occur frequently.

If the supply voltage recovers, the system again waits for sufficient charge on the capacitor. If the supply instead decreases below V_h, i.e. under the *insufficient power* condition, the system wakes, hibernates and returns to sleep. If *insufficient power* continues, the voltage will drop below V_{min} and the system will shut down losing all data in volatile memory.

PowerNapping has a safeguard against missing a state-save. Existing transient systems monitor the restore threshold when sleeping, but with PowerNapping the hibernate threshold is also monitored. This guarantees that when in sleep mode data will not be lost, even on sudden power-failure, whilst also waiting for the optimal restore voltage. The need to monitor a second threshold is the primary overhead of this scheme. After a hibernation, the system can safely lose power without data loss, therefore only the restore threshold is monitored.

A. Mathematical Analysis

In Hibernus++, hibernation was completed every time the supply voltage dropped below a certain threshold. For *low-power* sources such as photovoltaic cells, the supply voltage often recovers after this. To maximise active time we want to reduce the time overhead of unnecessary state saves. Equation 8 shows the time available for making forward progress for a given supply profile that leads to a total system on-time, T_{on}. This will be dependent on the number of supply interruptions. We can break this into interruptions that are:

1) avoidable (n_{ia}) - *low-power*, where the supply voltage wouldn't drop below V_{min} if sleep mode was entered.
2) unavoidable (n_{iu}) - *insufficient power*, supply will certainly drop below V_{min}.

$$T_\phi = T_{on} - T_a - n_{ia}(T_h + T_{sw} + T_\lambda) - n_{iu}(T_h + T_r + T_\lambda) \quad (8)$$

Where T_ϕ is the Hibernus++ forward progress, T_a is the time overhead introduced by the Hibernus++ algorithm, T_h is the time taken to hibernate, T_{sw} is the overhead of sleeping and waking, T_r is the time to restore system state and T_λ is the time spent sleeping or shut down before restoring.

The equivalent for PowerNapping is given by:

$$\begin{aligned} T_{\phi pn} = T_{on} - T_{apn} - n_{ia}(T_{sw} + T_\lambda + T_{\lambda pn}) \\ - n_{iu}(T_h + T_r + T_\lambda + T_{sw}) \end{aligned} \quad (9)$$

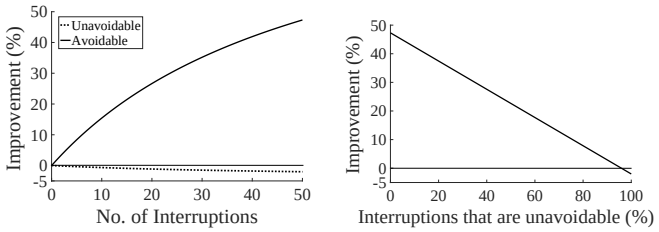
Where $T_{\phi pn}$ is the PowerNapping forward progress, T_{apn} is the time overhead of including the PowerNapping algorithm and $T_{\lambda pn}$ is the additional time in sleep due to PowerNapping. For this system to improve upon Hibernus++, it requires $T_{\phi pn} > T_\phi$. For a given time, T_{on}, by removing equivalent terms we can see the difference in forward progress between PowerNapping and Hibernus++:

$$\begin{aligned} T_{\phi pn} - T_\phi = (T_a + n_{ia}(T_h)) \\ -(T_{apn} + n_{ia}(T_{\lambda pn}) + n_{iu}(T_{sw})) \end{aligned} \quad (10)$$

The algorithm time difference, $T_a - T_{apn}$, is negligible. Using an MSP430FR5739 experimenter board, we found that T_h and $T_r \approx$ 1.87ms and from the data sheet T_{sw} is 78μs for LPM4. An MSP430FR5739 is chosen for this work due to its built in FRAM memory and prevalence in related transient-compute works. Equation 11 demonstrates how the improvement is dependent on the quantity of n_{iu} and n_{ia}.

$$T_{\phi pn} - T_\phi = n_{ia}(T_h - T_{\lambda pn}) - n_{iu}(T_{sw}) \quad (11)$$

This relationship is visualised in Figure 4, demonstrating that PowerNapping is most beneficial in situations where V_{min} is unlikely to be reached i.e. *low-power* sources, where n_{ia} >> n_{iu}. This benefit increases proportionally with the number of interruptions for a given time, T_{on}. When the majority of interruptions are unavoidable, as for intermittent *high-power* sources, PowerNapping is no longer beneficial and will be disabled. Figure 4b shows that for 50 interruptions PowerNapping is useful until >96% of interruptions are unavoidable.



(a) Improvement with increasing n_{ia} and n_{iu} independently

(b) For 50 interruptions, n, impact of increasing percentage that are unavoidable

Fig. 4. Improvement of PowerNapping with changing n_{ia} and n_{iu} over 50 interruptions

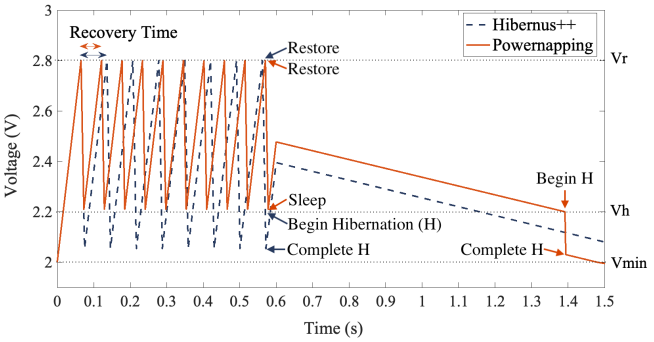


Fig. 5. Comparison of Hibernus++ and PowerNapping for a constant current source interrupted at ~600ms

If the checkpoint is avoidable, then all the energy that would have been wasted on a state save is retained. From the MSP430FR5739 data sheet, E_α = 4.2 nJ/byte and E_β = 2.7 nJ/byte, with a total RAM size of 1024 bytes and register size of 512 bytes. Compared with a complete state save, there should be 5.7μJ more energy available for forward progress. $T_{\lambda pn}$ is minimised by having the PowerNapping threshold (V_s) as close to the hibernate threshold (V_h) as possible, achieved at runtime using the calibration routine described previously.

B. Simulation

The objective of PowerNapping is to increase forward progress for *low-power* supplies. To verify the behaviour, and predict performance, mathematical simulation was used. Energy consumption for active, sleep, hibernation and restore were obtained from both the MSP430FR5739 data sheet [20] and physical characterisation. MATLAB was used to run these values through the mathematical models with various test conditions and input sources as demonstrated below.

Figure 5 shows the results of a simulation of the system response to an ideal 200 μA source that drops to 0 μA at ~600 ms. This is the worst case test, as typically *low-power* EH sources have slower current variation over time. An example case where this could occur is an indoor PV cell, powered with a consistent incoming light, having the light suddenly switched off. Each time the supply drops to the threshold V_h the Hibernus++ algorithm hibernates; moving the contents of RAM, registers and CPU state to FRAM, using energy from the decoupling capacitance of the system. The hibernation routine completes before reaching V_{min} (2 V in this case) at which point the system enters sleep and the supply recovers. PowerNapping instead enters sleep immediately and therefore recovers more quickly. For this supply profile, PowerNapping increases the forward progress by 24.4%, because less energy is wasted on hibernating. When the supply does drop below V_h, due to *insufficient power* during sleep, the system is able to wake and hibernate as seen around 1400 ms.

These two approaches were also compared using a double diode PV cell model [26] as the supply input. PV cells are low current DC sources, but do not supply a constant current. The current varies according to the IV characteristics of the cell. A cell was selected with a maximum power point (MPP) voltage within the operating voltage range of the MSP430. The model uses the open-circuit voltage, short circuit current, MPP voltage and MPP current to generate an IV curve, fed into the MSP430 MATLAB model. With the PV cell, PowerNapping increased forward progress by 18.3% with ~300 μA supply and 27.5% with ~100 μA supply.

The PowerNapping strategy should allow the supply voltage to recover without saving system state, but in the case where the supply does not recover, will hibernate successfully. From simulation, it is expected that the lower the input current, the greater the benefit of PowerNapping. Wasteful hibernations occur more frequently for Hibernus++ in this case, since the lower supply current leads to faster discharging of the capacitance whilst making forward progress.

PowerNapping has little overhead. The system characterises the source on start-up. For bursty *high-power* supplies, where the supply typically dies without recovery, PowerNapping is disabled and there is no additional overhead. When PowerNapping is utilised, the overhead is (1) an increase to the hibernation time due to sleeping first, and (2) the inclusion of a second voltage threshold monitoring circuit using the internal comparator. This overhead is included in simulation and the physical test setup.

It has been shown that decreasing the number of resource intensive hibernations leads to an increase in forward progress. This increase in forward progress leads to faster program execution, and a more efficient system. Improvements would therefore demonstrate an increased active time, and reduced completion time for benchmarks on a physical system, as demonstrated in Section V.

IV. Expedit - Reducing State Save Overheads

DMA controllers allows data to be accessed and copied without intervention from the CPU. They are useful in embedded systems for controlling memory transfers between peripherals and main memory. Their function is typically for speed-up, multi-tasking or reducing load on the processor, however for large transfers it is possible for the processor to enter sleep, in order to save power. Moving the contents of registers and RAM to NVM via DMA has not been done in any transient computing papers to date. Equation 8 shows that time spent on hibernate and restore has a direct impact on forward progress. By making hibernate and restore routines

quicker, there will be more time spent on useful computation, T_ϕ, for a given time, T_{on}.

Reactive transient systems necessitate large memory transfers from main memory into NVM. By transferring the registers and RAM into FRAM with the DMA, time and energy overheads can be reduced. The DMA is used in block-transfer mode. It is initialised with a source address in VM, and a destination address in NVM (or vice versa) and the size of transfer. The RAM can be copied in a single block, however the important registers in the MSP430 are distributed, and are transferred in multiple blocks. For the MSP430, a CPU-controlled data transfer uses 5 cycles, whereas DMA uses 2 [20]. Additionally, DMA is more power efficient, whilst also allowing the CPU to enter sleep. This gives a lower total power consumption when DMA is in use. The on-chip area overhead is also small, DMAs with a gate count of 3-10k are commercially available [27].

The benefits of Expedit are applicable any time data is copied from volatile to non-volatile memory or vice-versa. This applies to the majority of the leading transient papers, across both reactive and task-based approaches. The larger the block of data, the lower the relative impact of initialising the DMA. Reactive transient systems which copy the entire contents of RAM and registers, such as Hibernus++, therefore gain the greatest benefit. Expedit is simulated and practically validated as an extension to this system, and PowerNapping.

The Hibernus++ code uses a number of 'for' loops, arrays and temporary variables to copy the registers and RAM. The code for hibernate and restore with Expedit is simpler and so we can expect the size of the code written to the MSP430 to be significantly reduced. PowerNapping introduces additional interrupts and configuration of the internal comparator, so we can expect an increase to the code size for this scheme.

A. Simulation

In simulation, the overhead of initialising the DMA is considered negligible, because the cycles required to move data are orders of magnitude greater than initialising. Because the DMA uses 2 cycles per word, instead of 5, hibernate and restore times are reduced by 60%. The energy consumption of the MSP430FR5739 was measured to be 10% less with the DMA active and CPU in sleep, so this is also included.

For *low-power* supplies, the system does not drop below V_{min}. Therefore only T_h is relevant, and the improvement is expected to be less than for *high-power* supplies. Figure 8 shows the expected improvement of Expedit in simulation, alongside the practical validation.

Expedit is expected to perform better against Hibernus++ when both T_h and T_r are significant, such as with frequently interrupted *high-power* sources. The simulated improvement is given in Figure 11, alongside the practical validation.

V. Practical Validation

The Hibernus++ code was adapted and programmed onto an MSP430FR5739 experimenter board. This was connected to an external threshold detection circuit, monitoring V_H or V_R respectively. The internal comparator is only used for V_S.

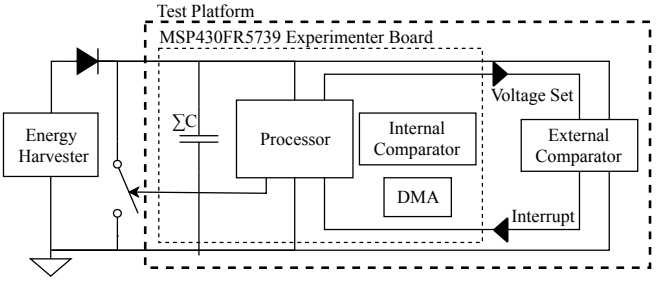


Fig. 6. Schematic of the test platform

TABLE I
Total overhead comparison of PowerNapping (PN) and Expedit (Exp) against Hibernus++ (H++)

	H++ (μs)	Exp (μs)	PN (μs)	Exp+PN (μs)
Restore	1874	306	1876	306
Hibernate	1876	555	1878	555
Sleep	N/A	N/A	166	166

(a) Time to complete listed functions

	H++ (μJ)	Exp (μJ)	PN (μJ)	Exp+PN (μJ)
Restore	7.30	1.05	7.31	1.05
Hibernate	7.31	1.91	7.32	1.91
Sleep	N/A	N/A	0.65	0.65

(b) Energy to complete listed functions

The experimental setup is shown in Figure 6. The test setup for Hibernus++ and the proposed approach is kept the same for all experiments and an FFT analysis is used as a test bench for comparing forward progress with a range of supply conditions. The FFT is completed three times emulating a realistic load: processing data from a 3-axis accelerometer. Additional benchmarks are included to demonstrate the performance of this scheme across different compute loads.

The proposed scheme is a reactive transient system that saves the entire contents of RAM and registers during hibernation. For this reason, time and energy overheads of hibernation and restore are not application dependent. Percentage improvement to completion time seen in these results is also independent of application, since the entire state is saved regardless of what that state is. For this reason other benchmarks are not included for clarity of results.

The energy harvester output is rectified by a Schottky diode. A low-dropout regulator (LDO) could be included to regulate the voltage within safe limits, however none of the sources used in testing exceeded the maximum operating voltage of the MSP430. The internal comparator is only active when in sleep, so there is additional power consumption in this state. For a 2 V supply, the internal comparator consumes 75 μA whereas the external comparator consumes 1 μA. PowerNapping requires both simultaneously. Two external comparators could be used in future implementations. If re-designed for on-chip implementation, more efficient comparators could be used, further reducing the overhead of PowerNapping. Table I compares the restore, hibernate and sleep for each scheme. These remained constant across all benchmarks presented.

The expectation is that Expedit reduces the hibernate and

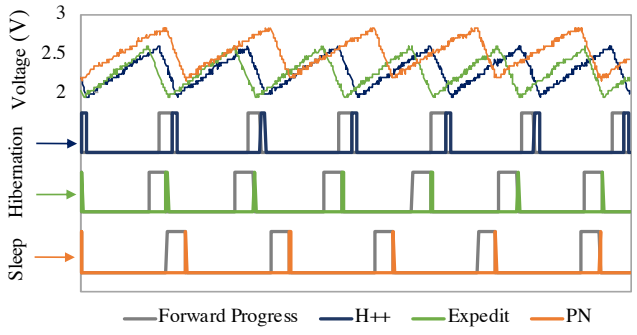


Fig. 7. Comparison of 3 schemes with PV cell input energy source, showing an increase in forward progress for PowerNapping and Expedit.

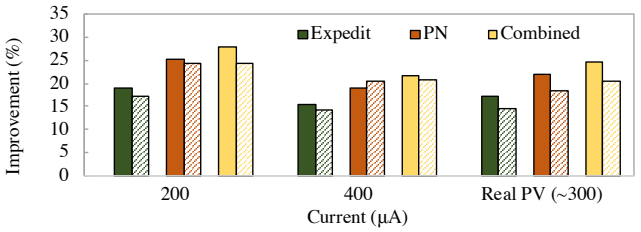


Fig. 8. Improvement in FFT complete time for the two schemes over Hibernus++ for simulation and practical validation with *low-power* sources. Solid colour represents practical validation, and cross-hatch, simulation results.

restore times, while PowerNapping has little overhead, but implements a sleep state that is reached much more quickly than hibernation. This is shown with Expedit improving hibernation time by 70.4% and restore, 83.6%. PowerNapping, which operates only when the harvester is generating a *low-power* supply, is 91.1% faster than hibernating.

A. Low-power Sources

Energy harvesters such as photovoltaic (PV) cells and thermo-electric generators (TEGs) typically supply power with lower current, but a more stable supply profile than other harvesters. PV cells also have the highest power density of mainstream energy harvesters [24]. PowerNapping is most effective for these *low-power* sources, but Expedit's reduced hibernate time is also beneficial.

Figure 7 compares Hibernus++ (H++), Expedit and PowerNapping (PN) with a 10x3cm PV cell under 500 lux indoor light, to demonstrate how hibernations impact active time. This delivers ~300 μA of current. Both Expedit and PowerNapping allow the processor to remain active for longer, seen by the increased width of forward progress, because they reduce costly hibernations. The decoupling capacitance charges earlier, leading to more frequent periods of forward progress, because less energy is spent on hibernations. This leads to improvement in the completion time of the FFT.

Figure 8 demonstrates this for a range of source conditions. For *low-power* supplies PowerNapping allows immediate supply recovery, increasing forward progress. The restore routine isn't utilised for *low-power* supplies, since the processor doesn't drop below V_{min}, and volatile data is not lost. PowerNapping removes resource-intensive hibernations by instead entering sleep. This occurs much more quickly as seen in Table I(a), leading to this improvement in the completion time. For lower currents this improvement exceeds our expectation from simulation. In simulation, V_r is approximated as a fixed value, however as explained in Section III, V_r varies according to the source characteristics. At lower currents, the capacitor charges more slowly, leading to a lower V_r and consequently more frequent hibernations. This favours PowerNapping and Expedit with their lower overheads.

TABLE II
COMPARISON OF CODE OVERHEAD FOR PROPOSED SCHEMES.

Additional Overhead	Code Bytes	Data Bytes
FFT alone	1852	1248
Hibernus++	+3486	+1508
Expedit	+3348	+586
PowerNapping	+4038	+1584
Combined	+4136	+1124

B. High-power Sources

Table I(b) shows the improvement in energy consumption. Expedit not only improves energy efficiency by completing the hibernate/restore more quickly, but the DMA also consumes less power. This leads to a hibernate energy cost reduction of 73.9% and restore, 85.6%. By reducing these overheads, more time and energy goes towards forward progress.

Table I(a) shows that Expedit is faster than expected in simulation. This is thought to be due to the additional algorithm simplifications, such as not reading register locations from a look up table. Table II shows how this also affects the size of code written to the MSP430 when programming the device. FFT is also given as an example base application code size.

As can be seen, Expedit reduces the size of the code overhead by over 1000 bytes (11%). This may enable Hibernus++ to be implemented on more tightly constrained systems. Hibernus++ contained large arrays of address locations (increasing data bytes), and 'for' loops for moving data (increasing code bytes). These elements are removed by using DMA with optimised address location initialisations.

Sources such as wind harvesters and piezo-electric vibration harvesters generate power with high current, but are typically bursty sources, with intermittent power. In a reactive transient system with little energy storage, the processor is likely to hibernate and restore frequently.

In this validation, an ideal DC voltage is interrupted with varying frequency to approximate the system response to these types of harvester. Figure 9 further demonstrates the impact of these reduced times by comparing the percentage overhead for a range of supply interruption frequencies. The higher the supply interruption frequency, the more state saves are required, and the greater the benefit of Expedit. Figure 9a demonstrates how the percentage of the active time spent on hibernate/restore is increasingly less than Hibernus++ and Figure 9b also, with percentage of energy.

The increase in active time can more clearly be seen for one

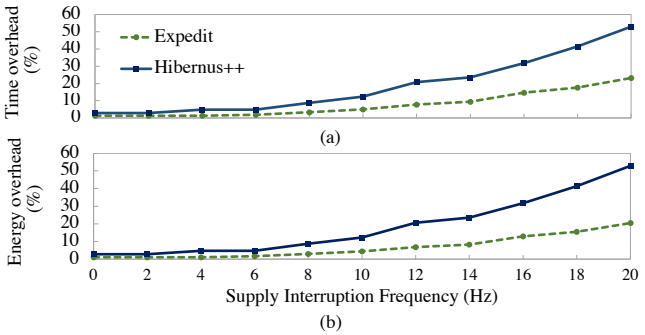


Fig. 9. Overhead Comparisons for Expedit and Hibernus++

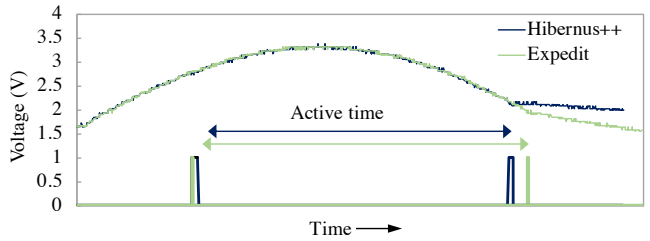


Fig. 10. Expedit Increase in Active Time with sinusoidal voltage input

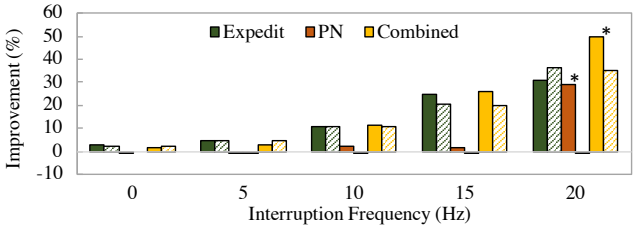


Fig. 11. Improvement in FFT complete time for the two schemes over Hibernus++ for simulation and practical validation with *high-power* sources. The solid colour represents practical validation, and the cross-hatch is simulation results. The * signifies a result where PowerNapping is enabled and by entering sleep the system avoids dropping below V_{min} across some interruptions.

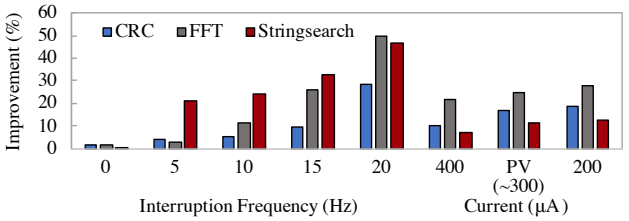


Fig. 12. Improvement of Expedit and PowerNapping combined for FFT, CRC and Stringsearch with high and low-power sources

cycle of a *high-power* AC input, as in Figure 10. By reducing the time and energy required, the hibernation can begin later, at a lower voltage threshold. This is established when first calibrating the system as explained in Section III-A.

Reduced time and energy overheads, and consequently increased active time have both been shown. The total time to complete 3 FFTs was also measured for a range of interruption frequencies. This further highlights the benefits of this scheme since faster task completion means that the processor is able to handle more complex tasks for a given supply profile.

Figure 11 compares the total on-time, T_{on}, of the processor when completing 3 FFTs. Removing time when supply is not available helps clarify the comparison. This figure shows that Expedit is able to complete the FFTs up to 30.8% faster than Hibernus++. As expected, PowerNapping does not give an improvement for high-power supplies. However, the results demonstrate that the associated overheads are minimal, except in the case of high-frequency interruptions which instead show significant improvement: up to 50.8% for PowerNapping alone, 51.7% when combined with the faster restore time of Expedit. At these high frequencies, the supply is interrupted faster than the ADC startup test can occur, defaulting to the system assuming *low-power* and enabling PowerNapping, unlike in simulation (Section III-B).

C. Additional Benchmarks

Figure 12 shows these schemes provide improvement for a range of IoT benchmarks, giving up to 35.2%, 50.0% and 46.8% improvement respectively. With the addition of both Expedit and PowerNapping, Hibernus++ is improved for both high and low-power sources. As shown in Figure 4a, greater benefits are gained when more intteruptions occur during a compute cycle, so results vary across benchmarks. Stringsearch takes 1045ms to complete, much longer than CRC at 103.8ms, and therefore more interruptions occur, across all frequencies, hence a greater improvement is observed for *high-power* sources. Conversely for *low-power* sources, longer base completion time means that hibernations make up a smaller percentage of total completion time, leading to lower percentage improvement being observed.

VI. Conclusions

With the two schemes, PowerNapping and Expedit, it has been demonstrated that the forward progress of reactive transient systems can be improved. PowerNapping allows time and energy previously wasted on hibernation to be used to increase the forward progress of the system. Improvements are particularly significant with *low-power* supplies, with up to 28.8% improvement in forward progress over the state-of-the-art. At high interruption frequencies, with *high-power* sources, PowerNapping can achieve up to 46.8% improvement. Expedit reduced the time and energy overheads of saving state by up to 83.6% and 85.6% respectively. This gives up to a 40.3% improvement in completion time over the state-of-the-art. The benefits of these two schemes have been shown experimentally with both ideal and real EH sources across multiple benchmarks. On the whole, results align with those expected from the simulation, with the schemes being more effective at higher interruption frequencies, or with lower current inputs. In combination they bring improvement with a full range of high and low power supplies of up to 50.0%. Tested with Hibernus++ [28] in this work, these schemes demonstrate a reduction in overheads that would be beneficial to other reactive transient works such as QuickRecall [25], or task-based approaches where large checkpoints occur.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Networks*, vol. 54, pp. 2787–2805, Oct. 2010.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, pp. 1645–1660, Sept. 2013.

[3] P. D. Mitcheson, "Energy harvesting for human wearable and implantable bio-sensors," in *2010 Int. Conf. of the IEEE Eng. in Med. and Biol*, pp. 3432–3436, IEEE, 2010.

[4] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, "Powering the internet of things," in *2014 Int. symp. on Low power electron. and des.*, ISLPED '14, (New York, NY, USA), pp. 375–380, ACM, June 2014.

[5] M. Rossi, L. Rizzon, M. Fait, R. Passerone, and D. Brunelli, "Energy neutral wireless sensing for server farms monitoring," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 4, pp. 324–334, Sept. 2014.

[6] A. Somov and R. Giaffreda, "Powering iot devices: Technologies and opportunities," *IEEE IoT Newsletter*, Nov. 2015.

[7] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on RFID-scale devices," *ACM SIGPLAN Notices*, vol. 46, pp. 159–170, May 2012.

[8] M. Hicks, "Clank: Architectural support for intermittent computation," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, pp. 228–240, June 2017.

[9] D. Balsamo, A. S. Weddell, and G. V. Merrett, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *IEEE Embedded Syst. Lett.*, vol. 7, no. 1, pp. 15–18, 2015.

[10] A. Rodriguez Arreola, D. Balsamo, and A. K. Das, "Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation," *Proc. Int. Workshop Energy Harvesting Energy Neutral Sens. Syst. (ENSsys)*, pp. 3–8, Nov. 2015. ISBN: 9781450338370.

[11] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, and Y. Xie, "Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power," *IEEE Micro*, vol. 36, pp. 72–83, May 2016.

[12] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan, "Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications," *IEEE Micro*, vol. 35, pp. 32–40, Sept. 2015.

[13] Y. Wang, Y. Liu, S. Li, D. Zhang, and B. Zhao, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *Proc. ESSCIRC*, pp. 149–152, Sept. 2012.

[14] Y. Liu, F. Suy, Z. Wangy, and H. Yang, "Design exploration of inrush current aware controller for nonvolatile processor," in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, (Hong Kong, Hong Kong), pp. 1–6, IEEE, Aug. 2015.

[15] J. Hester, K. Storer, and J. Sorber, "Timely Execution on Intermittently Powered Batteryless Sensors," in *Proc. ACM Sensys*, (Delft, Netherlands), pp. 1–13, ACM Press, 2017.

[16] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *Proc. OOPSLA*, (Amsterdam, Netherlands), pp. 514–530, ACM Press, Nov. 2016.

[17] P. Singla, S. S. Singh, and S. R. Sarangi, "Flexicheck: An adaptive checkpointing architecture for energy harvesting devices," in *Proc. Des. Autom. Test Eur. Conf. Exhibit.(DATE)*, pp. 546–551, IEEE, Mar. 2019.

[18] K. Maeng, A. Colin, and B. Lucia, "Alpaca: intermittent execution without checkpoints," *Proc. ACM Progam. Lang.*, pp. 96:1–96:30, Oct. 2017.

[19] D. Balsamo, A. S. Weddell, and A. Das, "Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, pp. 1968–1980, Mar. 2016.

[20] Texas Instruments, *MSP430FR573xx Mixed-Signal Microcontrollers*. http://www.ti.com/lit/ug/slau272d/slau272d.pdf, Dec. 2017.

[21] G. Lukosevicius, A. R. Arreola, and A. S. Weddell, "Using Sleep States to Maximize the Active Time of Transient Computing Systems," in *Proc. Int. Workshop Energy Harvesting Energy Neutral Sens. Syst. (ENSsys)*, (Delft, Netherlands), pp. 31–36, ACM Press, Nov. 2017.

[22] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, "Efficient intermittent computing with differential checkpointing," in *Proc. Int. Conf. Languages, Compilers, and Tools for Embedded Systems*, pp. 70–81, ACM, June 2019.

[23] K. Maeng and B. Lucia, "Supporting peripherals in intermittent systems with just-in-time checkpoints," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design and Implementation (PLDI)*, (New York, NY, USA), pp. 1101–1116, June 2019.

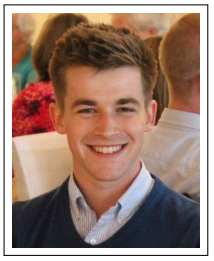
[24] A. Savanth, A. Weddell, J. Myers, D. Flynn, and B. Al-Hashimi, "Photovoltaic cells for micro-scale wireless sensor nodes: Measurement and modeling to assist system design," in *Proc. Int. Workshop Energy Harvesting Energy Neutral Sens. Syst. (ENSsys)*, (New York, NY, USA), pp. 15–20, Nov. 2015.

[25] H. Jayakumar, A. Raha, and V. Raghunathan, "QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers," *Proc. IEEE Int. Conf. VLSI Design*, pp. 330–335, Jan. 2014.

[26] M. C. Di Piazza and G. Vitale, *Photovoltaic sources: modeling and emulation*. Springer Science & Business Media, 2012.

[27] "PrimeCell uDMA Controller (PL230) Technical Reference Manual," 2007.

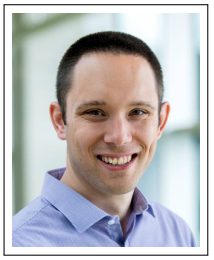
[28] D. Balsamo, A. Das, and A. S. Weddell, "Graceful Performance Modulation for Power-Neutral Transient Computing Systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, pp. 738–749, May 2016.



Tim Daulby received the B.Eng. degree (1st class honors) in electronic engineering from the University of Southampton, U.K., in 2017. He is currently pursuing a Ph.D. in electronic engineering with the ECS Group, University of Southampton. His research is focused on transient computing checkpointing strategies and hardware, software co-design for efficient batteryless embedded systems.

Anand Savanth received the master's degree in microelectronics from the University of Liverpool in 2010 where he received the Sir Robin Saxby Award. He is currently pursuing the Ph.D. degree with the ECS Group, University of Southampton, with a focus on custom and analog-assisted circuits for IoT platforms and energy harvesting applications. He has been with the Devices, Circuits and Systems Group, ARM Research, developing ultra-efficient circuits for Arm embedded processors and holds over 10 granted patents.

Alex S. Weddell (GSM'06-M'10) received the M.Eng. degree (1st class honors) and Ph.D. in electronic engineering from the University of Southampton, U.K., in 2005 and 2010. His main research focus is in the areas of energy harvesting and energy management for future Internet of Things devices. He has over 15 years' experience in energy harvesting systems, and has published over 65 papers in the area. He is now a Lecturer at the University of Southampton, involved with three projects funded by EPSRC, EU Horizon 2020 and Clean Sky 2.



Geoff V. Merrett (GSM'06-M'09-SM'19) is Professor of Electronic and Software Systems at the University of Southampton, U.K., where he previously received the B.Eng. (Hons) and Ph.D. degrees in 2004 and 2008, respectively. He is co-director of the Arm-ECS Research Centre, an award winning collaboration between the university and Arm Research, Cambridge. His current research interests are in energy management of mobile/embedded systems and self-powered devices, and he has published over 200 journal and conference articles on these topics.

Comparing NVM Technologies through the Lens of Intermittent Computation

Tim Daulby
Univesity of Southampton
Southampton, UK
td2g14@soton.ac.uk

Anand Savanth
Arm Research
Cambridge, UK
anand.savanth@arm.com

Alex S Weddell
Univesity of Southampton
Southampton, UK
asw@ecs.soton.ac.uk

Geoff V Merrett
Univesity of Southampton
Southampton, UK
gvm@ecs.soton.ac.uk

ABSTRACT

Intermittent computing (IC) promises long lifetimes for IoT edge devices. Running directly from energy harvesting sources enables these devices to be deployed and left, potentially for decades. As the field of IC progresses from proof-of-concept to deployable devices, the research focus must shift from processor-centric schemes to consideration of the whole system. The non-volatile memory (NVM) technology, as well as the way it is used, will have a significant effect. Properties such as latency, read/write energy, and endurance can vary by orders of magnitude, and this may affect the viability of many schemes presented in the literature. This paper presents a review of the characteristics of both commercially-available and future NVM technologies, and recommends design considerations for IC systems which incorporate these.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Intermittent Computing, Embedded Systems, IoT edge devices

ACM Reference Format:
Tim Daulby, Anand Savanth, Alex S Weddell, and Geoff V Merrett. 2020. Comparing NVM Technologies through the Lens of Intermittent Computation. In *The 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (ENSsys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3417308.3430268

1 INTRODUCTION

IoT edge devices are becoming increasingly pervasive, with many industries realising the value of a better connected physical world and cloud through live sensor data. However, the maintenance cost of battery powered systems, as well as their increased size, mass and cost, can be prohibitive. To overcome this, a new paradigm of battery-less devices is emerging, running intermittently on harvested energy. Intermittent computing (IC) works by saving state to non-volatile memory (NVM) before power failure. This state save can be periodic, at certain task boundaries, or triggered by a voltage interrupt. Most existing IC schemes have been implemented on commercial off-the-shelf (COTS) microcontrollers, e.g. MSP430FR5739, or in simulation. For systems to be deployed, greater consideration must be given to the impact of the whole system, including the available NVM technologies. Limited read/write speed, the high energy cost of accesses and poor endurance can all impact the system performance.

Table 1: Comparison of NVM memory technologies for IoT Devices (* signifies simulation data only)

Technology	Write/Read Energy (per bit)	Write/Read Time (per bit)	COTS	Endurance (Write cycles)
NAND Flash[4]	470 pJ/46 pJ	200 µs/25.2 µs	Y	10^5
SRAM[1]	355 pJ/587 pJ	2.2 ns/2.1 ns	Y	Unlimited
FRAM[5]	1.4 nJ/1.4 nJ	120 ns/120 ns	Y	10^{15}
STT-MRAM[3]	2 nJ/34 pJ	250 ns/10 ns	Y	10^5
SOT-MRAM*[1]	334 pJ/247 pJ	1.4 ns/1.1 ns	N	$>10^{15}$
ReRAM[6]	1.1 nJ/525 fJ	10 µs/5 ns	Y	10^5
PCM*[8]	13.5 pJ/2 pJ	150 ns/48 ns	N	10^7

2 NVM TECHNOLOGY PROPERTIES

The emergence and subsequent proliferation of faster, more efficient byte-addressable NVM technologies has made IC viable, with some authors completely replacing RAM with NVM in a unified memory approach (i.e. saving all data usually separated to ROM and RAM in the same non-volatile memory space, greatly reducing checkpoint size)[7]. Table 1 and Figure 1 compare leading NVM technologies, with SRAM and Flash included for comparison. Whilst there are many other considerations for system designers, we consider these key criteria for IC systems.

Energy consumption: Perhaps the most obvious consideration for NVM in ultra-constrained IC devices. The energy consumption of embedded CPU cycles can be orders of magnitude lower than the NVM access cost. This motivates the use of IC schemes that minimise the number of accesses, such as managed-state checkpointing [9]. In addition, many

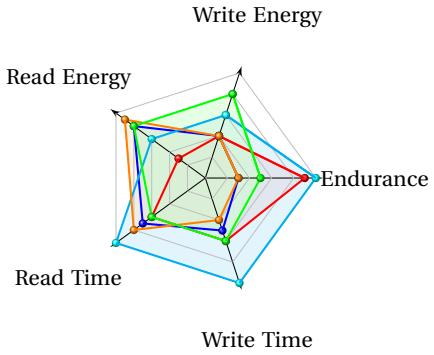


Figure 1: Radar plot of NVM characteristics. Line colors correspond to the colors in Table 1.

Table 2: The number of clock cycles needed for NVM access at 24MHz CPU clock frequency.

Technology	Write/Read Time (per bit)	Clock cycles (Write)	Clock cycles (Read)
NAND Flash	200µs/25.2µs	4800	605
SRAM	2.2ns/2.1ns	<<1	<<1
FRAM	120ns/120ns	2.9	2.9
STT-MRAM	250ns/10ns	6	0.24
SOT-MRAM	1.4ns/1.1ns	0.03	0.03
ReRAM	10µs/5ns	240	0.12
PCM	150ns/48ns	3.6	1.15

NVMs are asymmetric, i.e. writes are more expensive. This could motivate schemes that prefer re-execution over checkpointing, if clock cycles and reads are significantly cheaper.

Latency: The access times of NVM technologies vary by orders of magnitude, but there are scenarios where this will not affect the performance of the system. Table 2 shows the number of cycles needed to access NVM at 24MHz, indicating how the latency of these technologies would negatively impact performance. Enabling non-blocking writes would mask some of this latency but, when saving/restoring state, NVM latency will have a direct impact due to constant accesses. For unified memory approaches, the latency of these technologies directly constrains the CPU clock frequency.

Endurance: Intermittent schemes often use NVM for checkpointing in ways that would exhaust some NVM cells within a matter of months or even days. In Hibernus++ [2], for example, the authors demonstrate their scheme with a Seiko watch power trace, interrupting every 0.4s. With FRAM, this scheme could checkpoint successfully for 6.34 million years (ignoring all other factors), but STT-MRAM and ReRAM with 10^5 endurance would last only 4.63 days.

For devices to be truly long-running, NVM technologies with greater write endurance must be used, or IC schemes must consider checkpoints as a finite resource that should be more carefully allocated. Schemes such as Hibernus++, which save the entire RAM contents to NVM, must be replaced by schemes such as allocated/managed state [9] which greatly reduce the number of writes required. Additionally, methods that write minimal/small checkpoints must use wear-levelling approaches to avoid over-using regions of NVM. Due to their greater access frequency, unified memory approaches remain unsuitable for many NVM technologies owing to their endurance limitations. Many task-based IC approaches depend on unified memory models, so this could present a significant challenge, unless endurance can be improved.

3 CONCLUSIONS

For IC devices to go beyond research, into deployment, it is necessary to consider the impact of the NVM on system performance. Some NVMs which may initially appear suitable, such as SOT-MRAM, have not yet been commercialised, typically due to fabrication challenges or their early state of development. The drawbacks of available technology must be factored in to IC design, e.g. schemes with frequent checkpoints must sacrifice performance to target greater endurance.

The ideal case for IC is NVM that can be used as unified memory, as it has high endurance and low latency similar to SRAM. Many schemes in the literature utilise unified memory approaches, however this paper has identified how the increased latency/energy cost over SRAM cannot be ignored, and that low endurance is prohibitive with many existing NVM technologies. This paper aims to promote awareness of these issues, so that future research into IC can consider the importance of NVM to the success of these schemes.

ACKNOWLEDGEMENTS

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/P010164/1. Experimental data used in this article can be found at: https://doi.org/10.5258/SOTON/D1594.

REFERENCES

[1] Antaios. 2020. *Spin-Orbit Torque MRAM.* Technical Report. Antaios, 51 Avenue Jean Kuntzmann, 38830 Montbonnot – France. 3 pages. https://www.antaios.fr/IMG/pdf/web_site_sot_whitepaper.pdf

[2] Domenico Balsamo et al. 2016. Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 35, 12 (March 2016), 1968–1980. https://doi.org/10.1109/TCAD.2016.2547919

[3] Yu-Der Chih et al. 2020. 13.3 A 22nm 32Mb Embedded STT-MRAM with 10ns Read Speed, 1M Cycle Write Endurance, 10 Years Retention at 150° C and High Immunity to Magnetic Field Interference. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 222–224.

[4] Laura M Grupp et al. 2009. Characterizing flash memory: anomalies, observations, and applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 24–33.

[5] Texas Instruments. 2009. FRAM – New Generation of Non-Volatile Memory. https://www.ti.com/lit/ml/szzt014a/szzt014a.pdf

[6] Pulkit Jain et al. 2019. 13.2 A 3.6Mb 10.1Mb/mm Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5V with Sensing Time of 5ns at 0.7V. In *2019 IEEE Int. Solid- State Circ. Conf. - (ISSCC)*. IEEE, San Francisco, CA, USA, 212–214. https://doi.org/10.1109/ISSCC.2019.8662393

[7] Hrishikesh Jayakumar et al. 2016. Energy-Aware Memory Mapping for Hybrid FRAM-SRAM MCUs in IoT Edge Devices. In *2016 29th Int. Conf. on VLSI Design (VLSID)*. IEEE, Kolkata, India, 264–269. https://doi.org/10.1109/VLSID.2016.52

[8] Benjamin C Lee et al. 2009. Architecting phase change memory as a scalable dram alternative. In *Proc. 36th annual int. sym. on Comp. arch.* 2–13.

[9] Sivert T Sliper et al. 2019. Efficient state retention through paged memory management for reactive transient computing. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

References

[1] Texas Instruments . MSP430 Ultra-low-power Microcontrollers - Product Bulletin, 1999. URL https://www.ti.com/sc/docs/products/micro/msp430/39113.pdf.

[2] Texas Instruments . Intelligent System State Restoration After Power Failure With Compute Through Power Loss Utility, 2015. URL http://www.ti.com/lit/ug/tidu885/tidu885.pdf.

[3] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. Battery-less Zero-maintenance Embedded Sensing at the Mithræum of Circus Maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 368–381, New York, NY, USA, November 2020. Association for Computing Machinery. ISBN 978-1-4503-7590-0. URL https://doi.org/10.1145/3384419.3430722.

[4] Antaios. Spin-Orbit Torque MRAM, 2020. URL https://www.antaios.fr/IMG/pdf/web_site_sot_whitepaper.pdf. Pages: 3 Place: 51 Avenue Jean Kuntzmann, 38830 Montbonnot – France.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. ISSN 13891286. doi: 10.1016/j.comnet.2010.05.010. URL https://linkinghub.elsevier.com/retrieve/pii/S1389128610001568.

[6] Abu Bakar, Alexander G. Ross, Kasim Sinan Yildirim, and Josiah Hester. REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3):87:1–87:42, September 2021. doi: 10.1145/3478077. URL https://doi.org/10.1145/3478077.

[7] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):1968–1980, 2016. ISSN 0278-0070. doi: 10.1109/TCAD.2016.2547919.

[8] Domenico Balsamo, Alex S. Weddell, Geoff V. Merrett, Bashir M. Al-Hashimi, Davide Brunelli, and Luca Benini. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters*, 7(1):15–18, March 2015. ISSN 1943-0663, 1943-0671. doi: 10.1109/LES.2014.2371494. URL http://ieeexplore.ieee.org/document/6960060/.

[9] Domenico Balsamo, Anup Das, Alex S. Weddell, Davide Brunelli, Bashir M. Al-Hashimi, Geoff V. Merrett, and Luca Benini. Graceful Performance Modulation for Power-Neutral Transient Computing Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):738–749, 2016. ISSN 02780070. doi: 10.1109/TCAD.2016.2527713. ISBN: 0278-0070.

[10] Domenico Balsamo, Michele Magno, Kacper Kubara, Bogdan Lazarescu, and Geoff V. Merrett. Energy Harvesting Meets IoT: Fuelling Adoption of Transient Computing in Embedded Systems. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 413–417, April 2019. doi: 10.1109/WF-IoT.2019.8767302.

[11] S. C. Bartling, S. Khanna, M. P. Clinton, S. R. Summerfelt, J. A. Rodriguez, and H. P. McAdams. An 8MHz 75uA/MHz zero-leakage non-volatile logic-based Cortex-M0 MCU SoC exhibiting 100% digital state retention at VDD=0V with <400ns wakeup and sleep transitions. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 432–433, San Francisco, CA, February 2013. IEEE. ISBN 978-1-4673-4516-3 978-1-4673-4515-6 978-1-4673-4514-9. doi: 10.1109/ISSCC.2013.6487802. URL http://ieeexplore.ieee.org/document/6487802/.

[12] Stefano Basagni, M Yousof Naderi, Chiara Petrioli, and Dora Spenza. Wireless sensor networks with energy harvesting. *Mobile Ad Hoc Networking: Cutting Edge Directions, Second Edition*, pages 701–736, 2013.

[13] S. Beeby and N. White. *Energy Harvesting for Autonomous Systems*. Artech House series smart materials, structures, and systems. Artech House, 2010. ISBN 978-1-59693-719-2. URL https://books.google.co.uk/books?id=7H9xdFd4sikC.

[14] Steve P Beeby, RN Torah, MJ Tudor, P Glynne-Jones, T O'donnell, CR Saha, and S Roy. A micro electromagnetic generator for vibration energy harvesting. *Journal of Micromechanics and microengineering*, 17(7):1257, 2007.

[15] Marc Belleville, Herve Fanet, Paolo Fiorini, P Nicole, MJM Pelgrom, Christian Piguet, R Hahn, Chris Van Hoof, Ruud Vullers, Marco Tartagni, and others. Energy autonomous sensor systems: Towards a ubiquitous sensor technology. *Microelectronics Journal*, 41(11):740–745, 2010.

[16] Habbati Bellia, Ramdani Youcef, and Moulay Fatima. A detailed modeling of photovoltaic module using MATLAB. *NRIAG Journal of Astronomy and Geophysics*, 3

(1):53–61, June 2014. ISSN null. doi: 10.1016/j.nrjag.2014.04.001. URL https://doi.org/10.1016/j.nrjag.2014.04.001. Publisher: Taylor & Francis _eprint: https://doi.org/10.1016/j.nrjag.2014.04.001.

[17] Naveed Anwar Bhatti and Luca Mottola. Efficient State Retention for Transiently-powered Embedded Sensing. In *International Conference on Embedded Wireless Systems and Networks*, page 12, 2016.

[18] Mattias Borg. Lecture 6 - NEW TYPES OF MEMORY, 2018. URL https://www.eit.lth.se/fileadmin/eit/home/mabo7006/MtM/2017_MtM_Lecture_6.pdf.

[19] Jhon Alexander Gómez Caicedo. CMOS Low-Power Threshold Voltage Monitor Circuits and Applications. 2016. URL https://www.lume.ufrgs.br/bitstream/handle/10183/144080/000998456.pdf?sequence=1.

[20] Caltech. Thermoelectrics, 2018. URL http://www.thermoelectrics.caltech.edu/thermoelectrics/engineering.html.

[21] Sravanthi Chalasani and James M Conrad. A survey of energy harvesting sources for embedded systems. In *Southeastcon, 2008. IEEE*, pages 442–447. IEEE, 2008.

[22] Wei-Ming Chen, Pi-Cheng Hsiu, and Tei-Wei Kuo. Enabling Failure-resilient Intermittently-powered Systems Without Runtime Checkpointing. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 104:1–104:6, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6725-7. doi: 10.1145/3316781.3317816. URL http://doi.acm.org/10.1145/3316781.3317816. event-place: Las Vegas, NV, USA.

[23] Y. Chen. Challenges and opportunities of internet of things. In *17th Asia and South Pacific Design Automation Conference*, pages 383–388, January 2012. doi: 10.1109/ASPDAC.2012.6164978.

[24] Yu-Der Chih, Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Meng-Chun Shih, Kuei-Hung Shen, Harry Chuang, and Tsung-Yung Jonathan Chang. 13.3 A 22nm 32Mb Embedded STT-MRAM with 10ns Read Speed, 1M Cycle Write Endurance, 10 Years Retention at 150°C and High Immunity to Magnetic Field Interference. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 222–224, San Francisco, CA, USA, February 2020. IEEE. ISBN 978-1-72813-205-1. doi: 10.1109/ISSCC19947.2020.9062955. URL https://ieeexplore.ieee.org/document/9062955/.

[25] Alexei Colin and Brandon 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*, pages 514–530, Amsterdam, Netherlands, 2016. ACM Press. ISBN 978-1-4503-4444-9.

doi: 10.1145/2983990.2983995. URL http://dl.acm.org/citation.cfm?doid=2983990.2983995.

[26] Tim Daulby, Anand Savanth, Alex S Weddell, and Geoff V Merrett. Comparing NVM Technologies through the Lens of Intermittent Computation. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ENSsys '20, pages 77–78, New York, NY, USA, November 2020. Association for Computing Machinery. ISBN 978-1-4503-8129-1. doi: 10.1145/3417308.3430268. URL https://doi.org/10.1145/3417308.3430268.

[27] Tim Daulby, Anand Savanth, Geoff V. Merrett, and Alex S. Weddell. Improving the Forward Progress of Transient Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):444–452, March 2021. ISSN 1937-4151. doi: 10.1109/TCAD.2020.2999913.

[28] Jean Jacques DeLisle. 9 IoT-Enabling RF Transceivers, 2014. URL http://www.mwrf.com/active-components/9-iot-enabling-rf-transceivers.

[29] M.C. Di Piazza and G. Vitale. *Photovoltaic Sources: Modeling and Emulation*. Green Energy and Technology. Springer London, 2012. ISBN 978-1-4471-4378-9. URL https://books.google.es/books?id=99JyYBOxQF0C.

[30] Waleed Ejaz, Muhammad Naeem, Adnan Shahid, Alagan Anpalagan, and Minho Jo. Efficient Energy Management for the Internet of Things in Smart Cities. *IEEE Communications Magazine*, 55(1):84–91, January 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600218CM. URL http://ieeexplore.ieee.org/document/7823343/.

[31] Arrow Electronics. The Realities of RF Power Harvesting, October 2015. URL https://www.arrow.com/en/research-and-events/articles/the-realities-of-rf-power-harvesting.

[32] Electronics Tutorials. Sensors and Transducers, 2018. URL https://www.electronics-tutorials.ws/io/io_1.html.

[33] Joakim Eriksson, Adam Dunkels, and Niclas Finne. MSPsim – an Extensible Simulator for MSP430-equipped Sensor Boards. page 2, 2007.

[34] Digi-key Europe. Extend IoT Sensor Node Battery Life with Energy Harvesting, December 2015. URL https://www.digikey.sk/en/articles/techzone/2015/dec/extend-iot-sensor-node-battery-life-with-energy-harvesting.

[35] James M. Gilbert and Farooq Balouchi. Comparison of energy harvesting systems for wireless sensor networks. *International Journal of Automation and Computing*, 5(4):334–347, October 2008. ISSN 1476-8186, 1751-8520. doi: 10.1007/s11633-008-0334-2. URL http://link.springer.com/10.1007/s11633-008-0334-2.

[36] Andres Gomez, Lukas Sigrist, Thomas Schalch, Luca Benini, and Lothar Thiele. Efficient, Long-Term Logging of Rich Data Sensors Using Transient Sensor Nodes. *ACM Transactions on Embedded Computing Systems*, 17(1):1–23, September 2017. ISSN 15399087. doi: 10.1145/3047499. URL http://dl.acm.org/citation.cfm?doid=3136518.3047499.

[37] Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. Characterizing flash memory: anomalies, observations, and applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, page 24, New York, New York, 2009. ACM Press. ISBN 978-1-60558-798-1. doi: 10.1145/1669112.1669118. URL http://portal.acm.org/citation.cfm?doid=1669112.1669118.

[38] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, September 2013. ISSN 0167739X. doi: 10.1016/j.future.2013.01.010. URL https://linkinghub.elsevier.com/retrieve/pii/S0167739X13000241.

[39] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, December 2001. doi: 10.1109/WWC.2001.990739.

[40] Paul Hernday. Field Applications for IV Curve Tracers, 2011. URL http://doczz.net/doc/8664554/field-applications-for-iv-curve-tracers.

[41] Josiah Hester, Lanny Sitanayah, and Jacob 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*, pages 5–16, Seoul, South Korea, 2015. ACM Press. ISBN 978-1-4503-3631-4. doi: 10.1145/2809695.2809707. URL http://dl.acm.org/citation.cfm?doid=2809695.2809707.

[42] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P. Burleson, and Jacob Sorber. Persistent Clocks for Batteryless Sensing Devices. *ACM Trans. Embed. Comput. Syst.*, 15(4):77:1–77:28, August 2016. ISSN 1539-9087. doi: 10.1145/2903140. URL http://doi.acm.org/10.1145/2903140.

[43] Josiah Hester, Kevin Storer, and Jacob Sorber. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems - SenSys '17*, pages 1–13, Delft, Netherlands, 2017. ACM Press. ISBN 978-1-4503-5459-2. doi: 10.1145/3131672.3131673. URL http://dl.acm.org/citation.cfm?doid=3131672.3131673.

[44] M. Hicks. Clank: Architectural support for intermittent computation. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 228–240, June 2017. doi: 10.1145/3079856.3080238.

[45] Matthew Hicks. Thumbulator: Cycle accurate ARMv6-m instruction set simulator, 2016.

[46] Matthew Hicks. impedimentToProgress/MiBench2: MiBench ported for IoT devices, 2018. URL https://github.com/impedimentToProgress/MiBench2.

[47] J.L. Hill and D.E. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002. ISSN 0272-1732. doi: 10.1109/MM.2002.1134340. URL http://ieeexplore.ieee.org/document/1134340/.

[48] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, 2015. ISBN 978-0-521-80926-9. URL https://books.google.co.uk/books?id=LAiWPwAACAAJ.

[49] Jia-Mian Hu, Zheng Li, Long-Qing Chen, and Ce-Wen Nan. High-density magnetoresistive random access memory operating at ultralow voltage at room temperature. *Nature communications*, 2:553, 2011.

[50] Li Huang, Valer Pop, Ruben de Francisco, Ruud Vullers, Guido Dolmans, Harmke de Groot, and Koji Imamura. Ultra low power wireless and energy harvesting technologies - An ideal combination. In *2010 IEEE Int. Conf. on Commun. Syst. (ICCS)*, pages 295–300. IEEE, 2010.

[51] Texas Instruments. FRAM – New Generation of Non-Volatile Memory, 2009. URL https://www.ti.com/lit/ml/szzt014a/szzt014a.pdf. Publication Title: Texas Instruments.

[52] Bashima Islam and Shahriar Nirjon. Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(3):82:1–82:29, September 2020. doi: 10.1145/3411808. URL https://doi.org/10.1145/3411808.

[53] Pulkit Jain, Umut Arslan, Meenakshi Sekhar, Blake C. Lin, Liqiong Wei, Tanaya Sahu, Juan Alzate-vinasco, Ajay Vangapaty, Mesut Meterelliyoz, Nathan Strutt, Albert B. Chen, Patrick Hentges, Pedro A. Quintero, Chris Connor, Oleg Golonzka, Kevin Fischer, and Fatih Hamzaoglu. 13.2 A 3.6Mb 10.1Mb/mm ^{2} Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5V with Sensing Time of 5ns at 0.7V. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 212–214, San Francisco, CA, USA, February 2019. IEEE. ISBN 978-1-5386-8531-0. doi: 10.1109/ISSCC.2019.8662393. URL https://ieeexplore.ieee.org/document/8662393/.

[54] Hrishikesh Jayakumar, Kangwoo Lee, Woo Suk Lee, Arnab Raha, Younghyun Kim, and Vijay Raghunathan. Powering the Internet of Things. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ISLPED '14, pages 375–380, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2975-0. doi: 10.1145/2627369.2631644. URL http://doi.acm.org/10.1145/2627369.2631644. event-place: La Jolla, California, USA.

[55] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. *Proceedings of the IEEE International Conference on VLSI Design*, pages 330–335, 2014. ISSN 10639667. doi: 10.1109/VLSID.2014.63. ISBN: 9781479925124.

[56] Hrishikesh Jayakumar, Arnab Raha, and Vijay 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)*, pages 264–269, Kolkata, India, January 2016. IEEE. ISBN 978-1-4673-8700-2. doi: 10.1109/VLSID.2016.52. URL http://ieeexplore.ieee.org/document/7434963/.

[57] Ashutosh Jogalekar. *Moore's Law and battery technology: No dice*. April 2013. URL https://blogs.scientificamerican.com/the-curious-wavefunction/moores-law-and-battery-technology-no-dice/.

[58] Dionisis Kandris, Christos Nakas, Dimitrios Vomvas, and Grigorios Koulouras. Applications of Wireless Sensor Networks: An Up-to-Date Survey. *Applied System Innovation*, 3(1):14, March 2020. doi: 10.3390/asi3010014. URL https://www.mdpi.com/2571-5577/3/1/14. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[59] Imran Khan. DMA operation, January 2017. URL https://www.slideshare.net/ImranKhan2088/dma-operation.

[60] Junaid Ahmed Khan, Hassaan Khaliq Qureshi, and Adnan Iqbal. Energy management in Wireless Sensor Networks: A survey. *Computers & Electrical Engineering*, 41: 159–176, January 2015. ISSN 00457906. doi: 10.1016/j.compeleceng.2014.06.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0045790614001773.

[61] Vibhav KumarSachan, Syed Akhtar Imam, and M. T. Beg. Energy-Efficient Communication Methods in Wireless Sensor Networks: A Critical Review. *International Journal of Computer Applications*, 39(17):35–48, February 2012. ISSN 09758887. doi: 10.5120/4915-7484. URL http://research.ijcaonline.org/volume39/number17/pxc3877484.pdf.

[62] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, 2009.

[63] Arm Limited. ARM Cortex-M3 DesignStart Eval RTL and Testbench User Guide r0p0. URL https://developer.arm.com/documentation/100894/0000/simulation-and-integration-tests/simulation-environment/tarmac-trace.

[64] ARM Limited. PrimeCell uDMA Controller (PL230) Technical Reference Manual, 2007. URL https://static.docs.arm.com/ddi0417/a/DDI0417A_udmac_pl230_r0p0_trm.pdf.

[65] ARM Limited. ARM®v6-M Architecture Reference Manual - Issue E, June 2018. URL https://documentation-service.arm.com/static/5f8ff05ef86e16515cdbf826.

[66] Yongpan Liu, Fang Suy, Zhibo Wangy, and Huazhong Yang. Design exploration of inrush current aware controller for nonvolatile processor. In *2015 IEEE Non-Volatile Memory System and Applications Symposium (NVMSA)*, pages 1–6, Hong Kong, Hong Kong, August 2015. IEEE. ISBN 978-1-4673-6688-5. doi: 10.1109/NVMSA.2015.7304357. URL http://ieeexplore.ieee.org/document/7304357/.

[67] Brandon Lucia and Benjamin 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*, PLDI '15, pages 575–585, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3468-6. doi: 10.1145/2737924.2737978. URL http://doi.acm.org/10.1145/2737924.2737978. event-place: Portland, OR, USA.

[68] Giedrius Lukosevicius, Alberto Rodriguez Arreola, and Alex 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 - ENSsys'17*, pages 31–36, Delft, Netherlands, 2017. ACM Press. ISBN 978-1-4503-5477-6. doi: 10.1145/3142992.3142998. URL http://dl.acm.org/citation.cfm?doid=3142992.3142998.

[69] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan. Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications. *IEEE Micro*, 35(5):32–40, September 2015. ISSN 0272-1732. doi: 10.1109/MM.2015.88.

[70] 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), pages 526–537, February 2015. doi: 10.1109/HPCA.2015.7056060.

[71] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, Y. Xie, J. J. Sampson, and V. Narayanan. Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power. *IEEE Micro*, 36(3):72–83, May 2016. ISSN 0272-1732. doi: 10.1109/MM.2016.35.

[72] Kaisheng Ma, Xueqing Li, Srivatsa Rangachar Srinivasa, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. Spendthrift: Machine learning based resource and frequency scaling for ambient energy harvesting nonvolatile processors. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 678–683, Chiba, Japan, January 2017. IEEE. ISBN 978-1-5090-1558-0. doi: 10.1109/ASPDAC.2017.7858402. URL http://ieeexplore.ieee.org/document/7858402/.

[73] Kiwan Maeng and Brandon 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 - PLDI 2019*, pages 1101–1116, Phoenix, AZ, USA, 2019. ACM Press. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314613. URL http://dl.acm.org/citation.cfm?doid=3314221.3314613.

[74] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages*, 1 (OOPSLA):1–30, October 2017. ISSN 24751421. doi: 10.1145/3133920. URL http://dl.acm.org/citation.cfm?doid=3152284.3133920.

[75] M. Makdessi, A. Sari, and P. Venet. Metallized polymer film capacitors ageing law based on capacitance degradation. *Microelectronics Reliability*, 54(9-10):1823–1827, September 2014. ISSN 00262714. doi: 10.1016/j.microrel.2014.07.103. URL https://linkinghub.elsevier.com/retrieve/pii/S0026271414002996.

[76] S. J. Marinkovic and E. M. Popovici. Nano-Power Wireless Wake-Up Receiver With Serial Peripheral Interface. *IEEE Journal on Selected Areas in Communications*, 29(8): 1641–1647, September 2011. ISSN 0733-8716. doi: 10.1109/JSAC.2011.110913.

[77] Oscar E. Mattia, Hamilton Klimach, and Sergio Bampi. Sub-1 V supply nano-watt MOSFET-only threshold voltage extractor circuit. In *2014 27th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, September 2014. doi: 10.1145/2660540.2660991.

[78] Oscar E. Mattia, Hamilton Klimach, Sergio Bampi, and Marcio Schneider. 0.7 V supply self-biased nanoWatt MOS-only threshold voltage monitor. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 497–500, May 2015. doi: 10.1109/ISCAS.2015.7168679. ISSN: 2158-1525.

[79] Jagan Meena, Simon Sze, Umesh Chand, and Tseung-Yuen Tseng. Overview of emerging nonvolatile memory technologies. *Nanoscale Research Letters*, 9(1):526, 2014. ISSN 1556-276X. doi: 10.1186/1556-276X-9-526. URL http://nanoscalereslett.springeropen.com/articles/10.1186/1556-276X-9-526.

[80] Paul D Mitcheson. Energy harvesting for human wearable and implantable bio-sensors. In *2010 Int. Conf. of the IEEE Eng. in Med. and Biol*, pages 3432–3436. IEEE, 2010.

[81] Next. solar cell : Repository-circuits. URL http://www.next.gr/circuits/solar-cell-l47255.html.

[82] Pavan Nukala. *Crystal-amorphous transformation via defect templating in phase-change materials*. University of Pennsylvania, 2015.

[83] James Pallister, Simon Hollis, and Jeremy Bennett. BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms. Technical Report arXiv:1308.5174, arXiv, September 2013. URL http://arxiv.org/abs/1308.5174. arXiv:1308.5174 [cs] type: article.

[84] Chen Pan, Mimi Xie, and Jingtong Hu. ENZYME: An Energy-Efficient Transient Computing Paradigm for Ultralow Self-Powered IoT Edge Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2440–2450, November 2018. ISSN 0278-0070, 1937-4151. doi: 10.1109/TCAD.2018.2858478. URL https://ieeexplore.ieee.org/document/8493580/.

[85] Joseph A Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive computing*, 4(1):18–27, 2005.

[86] J. Polastre, R. Szewczyk, C. Sharp, and D. Culler, editors. *The Mote revolution: Low power wireless sensor networks*. Proceedings of the 16th Symposium on High Performance Chips (HotChips), August 2004.

[87] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 364–369, Los Angeles, CA, USA, 2005. IEEE. ISBN 978-0-7803-9201-4. doi: 10.1109/IPSN.2005.1440950. URL http://ieeexplore.ieee.org/document/1440950/.

[88] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 457–462, Los Angeles, CA, USA, 2005. IEEE. ISBN 978-0-7803-9201-4. doi: 10.1109/IPSN.2005.1440973. URL http://ieeexplore.ieee.org/document/1440973/.

[89] Benjamin Ransford, Jacob Sorber, and Kevin 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, pages 159–170, New York, NY, USA, March 2011. Association for Computing Machinery. ISBN 978-1-4503-0266-1. doi: 10.1145/1950365.1950386. URL https://doi.org/10.1145/1950365.1950386.

[90] Alberto Rodriguez. Sustaining Computation during Intermittent Supply for Energy Harvesting Systems.

[91] Alberto Rodriguez Arreola, Domenico Balsamo, Anup K. Das, Alex S. Weddell, Davide Brunelli, Bashir M. Al-Hashimi, and Geoff V. Merrett. Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation. *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems - ENSsys '15*, pages 3–8, 2015. doi: 10.1145/2820645.2820652. URL http://dl.acm.org/citation.cfm?id=2820645.2820652. ISBN: 9781450338370.

[92] Maurizio Rossi, Luca Rizzon, Matteo Fait, Roberto Passerone, and Davide Brunelli. Energy Neutral Wireless Sensing for Server Farms Monitoring. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(3):324–334, September 2014. ISSN 2156-3357, 2156-3365. doi: 10.1109/JETCAS.2014.2337171. URL http://ieeexplore.ieee.org/document/6866235/.

[93] Shad Roundy, Eli S Leland, Jessy Baker, Eric Carleton, Elizabeth Reilly, Elaine Lai, Brian Otis, Jan M Rabaey, Paul K Wright, and V Sundararajan. Improving power output for vibration-based energy scavengers. *IEEE Pervasive computing*, 4(1):28–36, 2005.

[94] Noboru Sakimura, Yukihide Tsuji, Ryusuke Nebashi, Hiroaki Honjo, Ayuka Morioka, Kunihiko Ishihara, Keizo Kinoshita, Shunsuke Fukami, Sadahiko Miura, Naoki Kasai, Tetsuo Endoh, Hideo Ohno, Takahiro Hanyu, and Tadahiko Sugibayashi. 10.5 A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 184–185, San Francisco, CA, USA, February 2014. IEEE. ISBN 978-1-4799-0920-9 978-1-4799-0918-6. doi: 10.1109/ISSCC.2014.6757392. URL http://ieeexplore.ieee.org/document/6757392/.

[95] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith. Design of an RFID-Based Battery-Free Programmable Sensing Platform. *IEEE Transactions on Instrumentation and Measurement*, 57(11):2608–2615, November 2008. ISSN 0018-9456. doi: 10.1109/TIM.2008.925019.

[96] Smruti R. Sarangi, Rajshekar Kalayappan, Prathmesh Kallurkar, Seep Goel, and Eldhose Peter. Tejas: A java based versatile micro-architectural simulator. In *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 47–54, September 2015. doi: 10.1109/PATMOS.2015.7347586.

[97] Anand Savanth, Alex S. Weddell, James Myers, David Flynn, and Bashir M. Al-Hashimi. Integrated Reciprocal Conversion With Selective Direct Operation for Energy Harvesting Systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64 (9):2370–2379, September 2017. ISSN 1549-8328, 1558-0806. doi: 10.1109/TCSI.2017.2707304. URL http://ieeexplore.ieee.org/document/7948804/.

[98] S. Sengupta. An input-free NMOS V/sub T/ extractor circuit in presence of body effects. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 1, pages I–912, May 2004. doi: 10.1109/ISCAS.2004.1328344.

[99] Faisal Karim Shaikh and Sherali Zeadally. Energy harvesting in wireless sensor networks: A comprehensive review. *Renewable and Sustainable Energy Reviews*, 55: 1041–1054, March 2016. ISSN 13640321. doi: 10.1016/j.rser.2015.11.010. URL https://linkinghub.elsevier.com/retrieve/pii/S1364032115012629.

[100] Priyanka Singla, Shubhankar Suman Singh, and Smruti R. Sarangi. FlexiCheck: An Adaptive Checkpointing Architecture for Energy Harvesting Devices. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 546–551, March 2019. doi: 10.23919/DATE.2019.8715130. ISSN: 1558-1101.

[101] Sivert T. Sliper, Domenico Balsamo, Nikos Nikoleris, William Wang, Alex S. Weddell, and Geoff V. Merrett. Efficient State Retention through Paged Memory Management for Reactive Transient Computing. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 1–6, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6725-7. doi: 10.1145/3316781.3317812. URL https://doi.org/10.1145/3316781.3317812.

[102] Sivert T. Sliper, William Wang, Nikos Nikoleris, Alex S. Weddell, and Geoff V. Merrett. Fused: Closed-Loop Performance and Energy Simulation of Embedded Systems. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 263–272, August 2020. doi: 10.1109/ISPASS48437.2020.00046.

[103] Andrey Somov and Raffaele Giaffreda. Powering IoT Devices: Technologies and Opportunities. *IEEE IoT Newsletter*, November 2015.

[104] Philip Sparks. The route to a trillion devices. Technical report, Arm Limited, 2017. URL https://community.arm.com/arm-community-blogs/b/internet-of-things-blog/posts/white-paper-the-route-to-a-trillion-devices.

[105] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. A Ferroelectric Nonvolatile Processor with 46 us System-Level Wake-up Time and 14 us Sleep Time for Energy Harvesting Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(3):596–607, March 2017. ISSN 1549-8328, 1558-0806. doi:

10.1109/TCSI.2016.2616905. URL http://ieeexplore.ieee.org/document/7726049/.

[106] Fang Su, Kaisheng Ma, Xueqing Li, Tongda Wu, Yongpan Liu, and Vijaykrishnan Narayanan. Nonvolatile Processors: Why is it Trending? *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 966–971, March 2017. doi: 10.23919/DATE.2017.7927131.

[107] Dean Takashi. SoftBank CEO Masayoshi Son sees a future with 1 trillion Internet of Things devices, October 2016. URL https://tinyurl.com/38zcy945.

[108] Yen Kheng Tan and Sanjib Kumar Panda. Energy harvesting from hybrid indoor ambient light and thermal energy sources for enhanced performance of wireless sensor nodes. *IEEE Transactions on Industrial Electronics*, 58(9):4424–4435, 2011.

[109] Andrew S Tanenbaum. *Modern operating system*, volume 3. Pearson Education, Inc, 2009.

[110] Christine Taylor. SRAM vs. DRAM, November 2018. URL https://www.enterprisestorageforum.com/storage-hardware/sram-vs-dram.html.

[111] Texas Instruments. *MSP430FR573x Mixed-Signal Microcontrollers*. Texas Instruments, December 2017. URL http://www.ti.com/lit/ds/symlink/msp430fr5739.pdf.

[112] TI Texas Instruments. MSP-EXP430FR5739 FRAM Experimenter Board User's Guide, 2011. URL https://www.ti.com/lit/ug/slau343b/slau343b.pdf.

[113] Erick Omar Torres. *An electrostatic CMOS/BiCMOS Lithium ion vibration-based harvester-charger IC*. PhD Thesis, Georgia Institute of Technology, 2010.

[114] Marcelo Gradella Villalva, Jonas Rafael Gazoli, and Ernesto Ruppert Filho. Comprehensive Approach to Modeling and Simulation of Photovoltaic Arrays. *IEEE Transactions on Power Electronics*, 24(5):1198–1208, May 2009. ISSN 1941-0107. doi: 10.1109/TPEL.2009.2013862. Conference Name: IEEE Transactions on Power Electronics.

[115] S. Vlassis and C. Psychalinos. Low-voltage CMOS VT extractor. *Electronics Letters*, 43 (17):921–923, August 2007. ISSN 1350-911X. doi: 10.1049/el:20070917. URL https://digital-library.theiet.org/content/journals/10.1049/el_20070917. Publisher: IET Digital Library.

[116] RJM Vullers, Rob van Schaijk, Inge Doms, Chris Van Hoof, and R Mertens. Micropower energy harvesting. *Solid-State Electronics*, 53(7):684–693, 2009.

[117] J. Wang, Y. Liu, H. Yang, and H. Wang. A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications. In *The 2010 International Conference on Green Circuits and Systems*, pages 646–650, June 2010. doi: 10.1109/ICGCS.2010.5542984.

[118] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M. 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)*, pages 149–152, September 2012. doi: 10.1109/ESSCIRC.2012.6341281.

[119] Yanbin Wang, G. Tarr, and Yanjie Wang. Input-free cascode V/sub thn/ and V/sub thp/ extractor circuits. In *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004. ICECS 2004.*, pages 282–285, December 2004. doi: 10.1109/ICECS.2004.1399673.

[120] Alex S. Weddell, Michele Magno, Geoff V. Merrett, Davide Brunelli, Bashir M. Al-Hashimi, and Luca Benini. A Survey of Multi-Source Energy Harvesting Systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 905–908, Grenoble, France, 2013. IEEE Conference Publications. ISBN 978-1-4673-5071-6. doi: 10.7873/DATE.2013.190. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6513636.

[121] Harrison Williams, Michael Moukarzel, and Matthew 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)*, pages 665–678, June 2021. doi: 10.1109/ISCA52012.2021.00058. ISSN: 2575-713X.

[122] Joel Van Der Woude and Matthew Hicks. Intermittent Computation without Hardware Support or Programmer Intervention. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 17–32, Savannah, GA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/vanderwoude.

[123] Xiaofan Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 463–468, Los Angeles, CA, USA, 2005. IEEE. ISBN 978-0-7803-9201-4. doi: 10.1109/IPSN.2005.1440974. URL http://ieeexplore.ieee.org/document/1440974/.

[124] Daniel J Yeager, Alanson P Sample, Joshua R Smith, and Joshua R Smith. WISP: A passively powered UHF RFID tag with sensing and computation. *RFID handbook: Applications, technology, security, and privacy*, 2008. Publisher: Citeseer.

[125] Jie Zhan, Geoff V. Merrett, and Alex S. Weddell. Exploring the Effect of Energy Storage Sizing on Intermittent Computing System Performance. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(3):492–501, March 2022. ISSN 1937-4151. doi: 10.1109/TCAD.2021.3068946. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.