Title
Making the Most of Thin Data: A Hardware-Software Approach

Permalink
https://escholarship.org/uc/item/9h69s1xx

Author
Martin, Paul Daniel

Publication Date
2013

Peer reviewed|Thesis/dissertation
Making the Most of Thin Data: A Hardware-Software Approach

A thesis submitted in partial satisfaction of the requirements for the degree Master of Science in Electrical Engineering

by

Paul Daniel Martin

2013
With the explosion of sensing platforms and modalities in recent decades and the growing interest in gathering and disseminating this data came corresponding difficulties in energy management and, more broadly, information management. To combat this deluge of information, there has been considerable research in fields spanning the entire spectrum of computing, from data centers to integrated circuits. In the realm of embedded systems and low power sensing, many have focused on exploiting sensor streams characterized by a very low information rate, i.e. when the data source is highly compressible and thus the phenomenon to be sensed produces a Thin Data stream. This work focuses on several architectures and techniques for reducing power consumption for sensing Thin Data streams. Specifically, the focus of the this work is divided three-fold: (1) We consider an operating systems perspective for allowing duty-cycled scheduling of periodic low power sensing tasks in the face of hardware and temperature variability; (2) we demonstrate low power sensing strategies for sporadic sources in embedded hardware and software with applications in water flow monitoring; (3) finally, we consider low power, low information sensor architectures from a hardware point of view, constructing a low power front-end for compressed sensing.
The thesis of Paul Daniel Martin is approved.

William J. Kaiser

Greg J. Pottie

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2013
# Table of Contents

1 Preface ................................................................. 1

2 Application Quality for Periodic Sensing Tasks .................. 4
   2.1 The Problem with Variation .................................. 4
   2.2 Related Work .................................................... 6
   2.3 Power Variability .............................................. 8
   2.4 Optimizing Utility with Variable Hardware .................. 10
      2.4.1 Task Modeling .......................................... 10
      2.4.2 Maximizing Application Utility ....................... 13
   2.5 VaRTOS, the Variability Aware Real Time Operating System 17
      2.5.1 Online Modeling of Sleep and Active Power .......... 17
      2.5.2 Online Modeling of Task Computation ............... 20
      2.5.3 Controlling Task Active Time ......................... 20
      2.5.4 Temperature Models .................................... 21
      2.5.5 User Programming Model ................................ 22
      2.5.6 Operation .................................................. 22
   2.6 Experimental Setup ............................................ 24
   2.7 Evaluation ....................................................... 26
      2.7.1 Minimizing Energy Consumption Error ................ 26
      2.7.2 Utility and Oracle Comparison ........................ 28
      2.7.3 Energy and Memory Overhead ........................... 29
   2.8 Case Studies .................................................... 30
      2.8.1 Multi-Agent Applications ............................... 31
2.8.2 Prediction-type Applications ............................................. 33
2.8.3 Block Processing Applications ............................................. 35

3 Sensing Sporadic Data Streams .................................................. 38
3.1 Event-triggered Sensing for Water Monitoring ................................. 38
3.1.1 Low Power Wakeup ............................................................. 39
3.2 Related Work ............................................................................. 41
3.2.1 Thermoelectric Harvesting ...................................................... 41
3.2.2 Water Flow Monitoring .......................................................... 43
3.3 System Architecture ................................................................. 44
3.3.1 Thermoelectric Generators and the Seebeck Effect ......................... 44
3.3.2 Mechanical Design ................................................................. 45
3.3.3 Electrical Design ................................................................. 46
3.3.4 Power Management ............................................................... 46
3.3.5 Low-Power Wakeup ............................................................... 47
3.4 Evaluation ................................................................................. 49
3.4.1 Energy Harvesting Characterization .......................................... 49
3.4.2 Fine-Grained Test Data Collection ........................................... 50
3.4.3 Wakeup Power, Latency, and Accuracy ...................................... 53
3.4.4 Subsystem Power Characterization .......................................... 56
3.4.5 Net Energy Profile ............................................................... 57
3.4.6 Efficiency .............................................................................. 60
3.5 Discussion and Future Work ....................................................... 62

4 Low Power Sensing Hardware for Sparse Signals ............................. 64
4.1 Signal Acquisition and Sparsity .................................................... 64
# List of Figures

2.1 ITRS predictions for processor power variation for years to come ........ 6
2.2 potential results of variability in terms of system quality and lifetime .... 6
2.3 Sleep and active power for three processor instances in 45nm ............... 10
2.4 Example utility curves ........................................... 12
2.5 Maximizing utility for two tasks and different values of system duty cycle, $d^*_\text{sys}$ 15
2.6 Example optimal duty cycle points for 3 example tasks with different values for $\{k_{\text{min}}, k_{\text{max}}, p_t\}$ .................................... 15
2.7 Modeling sleep and active power through linearization. ........................ 18
2.8 Error convergence for sleep power modeling ................................. 19
2.9 Error in average power estimation for temperature models ................. 19
2.10 A tool for guiding developers using VaRTOS ............................... 23
2.11 VaRTOS state chart, showing model convergence and optimization states. 23
2.12 Error in energy consumption for various optimal duty cycles .............. 26
2.13 Errors in energy consumption for a single-task application ................. 28
2.14 Average duty cycles of a single-task application ............................. 28
2.15 Total utility, $u_{\text{sys}}$, for VaRTOS vs. the oracle system. .................. 28
2.16 Multi-agent localization application ...................................... 31
2.17 Reduced error for multi-agent localization using instance power modeling with VaRTOS .......................................................... 32
2.18 Increased error for multi-agent localization assuming worst-case power ... 32
2.19 Kalman filter prediction of velocity from noisy position measurements .... 34
2.20 Error in Kalman predictions .............................................. 34
2.21 Energy consumption errors for a mult-block signal processing application . 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Sample data of pipe temperature, wakeup triggers from DoubleDip, and pipe vibration</td>
</tr>
<tr>
<td>3.2</td>
<td>Diagram of DoubleDip node attached to water pipe</td>
</tr>
<tr>
<td>3.3</td>
<td>Peltier module including aluminum thermal coupler and heatsink</td>
</tr>
<tr>
<td>3.4</td>
<td>DoubleDip circuitry schematic</td>
</tr>
<tr>
<td>3.5</td>
<td>Comparator-based low power wakeup trigger</td>
</tr>
<tr>
<td>3.6</td>
<td>Open circuit output voltage for a CUI60333 Peltier module vs. differential temperature</td>
</tr>
<tr>
<td>3.7</td>
<td>Short circuit output current of boost regulator vs. input temperature differential, $\Delta T$</td>
</tr>
<tr>
<td>3.8</td>
<td>Testing PCB</td>
</tr>
<tr>
<td>3.9</td>
<td>DoubleDip node PCB</td>
</tr>
<tr>
<td>3.10</td>
<td>Power harvested for an example weekday during March</td>
</tr>
<tr>
<td>3.11</td>
<td>Current consumption for low power wakeup circuitry and deep sleep</td>
</tr>
<tr>
<td>3.12</td>
<td>Example calculation of wakeup latency</td>
</tr>
<tr>
<td>3.13</td>
<td>Cumulative distribution function (CDF) for wakeup latencies across all nodes</td>
</tr>
<tr>
<td>3.14</td>
<td>CDF of wakeup latency for a single node before and after tuning the wakeup threshold</td>
</tr>
<tr>
<td>3.15</td>
<td>Current consumption of an ADXL35 3-axis analog accelerometer</td>
</tr>
<tr>
<td>3.16</td>
<td>Current consumption of a Nordic nRF24L01+ radio</td>
</tr>
<tr>
<td>3.17</td>
<td>Energy harvested and consumed per day for 2 example weeks</td>
</tr>
<tr>
<td>3.18</td>
<td>Example temperature curves for hot water bursts</td>
</tr>
<tr>
<td>3.19</td>
<td>Temperature change rate for varying contact areas</td>
</tr>
<tr>
<td>4.1</td>
<td>Visual representation of compressed sensing</td>
</tr>
<tr>
<td>4.2</td>
<td>Software implementation of Compressed Sensing</td>
</tr>
</tbody>
</table>
4.3 Hardware implementation of Compressed Sensing using parallel random de-
modulation ................................................................. 70
4.4 Hardware implementation of Compressed Sensing in CapMux using a single
time multiplexed signal processing chain. .................................. 72
4.5 Implementing the time-multiplexed integration. ......................... 73
4.6 Empirical recovery performance with $16 \times 64$ sparse binary sampling matrices
of varying density compared to traditional Bernoulli sampling matrices. .... 75
4.7 Error between measured and expected values before and after calibration .. 77
4.8 An example of per-column error for analog matrix multiplication after cali-
bration ................................................................. 78
4.9 The 16-channel board with a ECG front end as application example. ....... 79
4.10 An example of compressed sensing and recovery of a 3-sparse signal in the
frequency domain using the CapMux hardware ............................ 80
4.11 Simulated and measured SNR performance for signals sparse in the time domain 81
4.12 Simulated and measured SNR performance for signals sparse in the frequency
domain ................................................................. 81
4.13 Simulated and measured SNR performance for signals sparse in the wavelet
domain ................................................................. 82
4.14 Measured current from the CapMux Hardware ........................... 82
4.15 Average current consumption as a function of sampling frequency ........ 83
4.16 Comparison of total current consumption used by the hardware implementa-
tion and a software implementation ....................................... 83
4.17 Potential for improvement for different compression ratios ............... 84
4.18 Empirical recovery performance with $64 \times 256$ sparse binary sampling matrices
of low density compared to Bernoulli sampling matrices. ............... 86
## List of Tables

2.1 Summary of selected variables .................................................. 12  
2.2 Task performance for multi-agent localization with and without VaRTOS . 33  
2.3 Processor cycle counts for a multi-block signal processing application .... 35  
3.1 Total deployment statistics .......................................................... 52  
3.2 Wakeup latencies and accuracies per day, per node. .......................... 55  
3.3 Coupler efficiencies ................................................................. 61
Acknowledgments

I would like to thank Zainul Charbiwala and Lucas Wanner for mentoring me in several aspects of my work and helping prepare me for my pursuit of a doctoral degree. I would also like to give my thanks to my Advisor Mani Srivastava for his insights and continued support. The work presented herein was supported in part by the NSF under awards # CNS-0905580, CCF-1029030, CNS-0910706, CNS-0905580, CNS-1143667, and OIA-0963183.
CHAPTER 1
Preface

As mobile devices and embedded sensors grow increasingly abundant, managing energy expenditure in these devices is a necessary step towards enabling sustainable growth. The work presented herein describes three angles of attack for managing energy expenditure in low power embedded devices for thin data streams—data that is characterized by a low information rate and is potentially compressible in some domain.

At the highest level, applications running on embedded systems must control the ratio of active computation time to total time, managing power by controlling a system duty cycle. Duty cycling mechanisms focus system resources on specific time intervals of interest, e.g. where the phenomenon being sensed is behaving in an interesting way, using prior knowledge of the behavior of these phenomena to select appropriate active time frames. Chapter 2 describes work by Paul Martin, Lucas Wanner, & Mani Srivastava under submission to ACM Transactions on Embedded Computing Systems (TECS) Special Issue on Parallel Programming and Run-Time Management for Embedded Systems. This work describes a new method for duty cycle scheduling of multiple-task applications for embedded systems designed to sense a phenomenon of a periodic and temporally sparse nature. In systems where computation is severely limited by anemic energy reserves and where a long overall system lifetime is desired, maximizing the quality of a given application subject to these constraints is both challenging and an important step towards achieving high quality deployments. Chapter 2 describes VaRTOS, an architecture and corresponding set of operating system abstractions that provide explicit treatment of both idle and active power variations for tasks running in real time operating systems. Tasks in VaRTOS express elasticity by exposing individual knobs—shared variables that the operating system can tune to adjust
task quality and correspondingly task power, maximizing application utility both on a per-
task and system-wide basis. We provide results regarding online learning of instance-specific
sleep power, active power, and task-level power expenditure on simulated hardware with
demonstrated effects for several prototypical applications. The results show that VaRTOS
can reduce variability-induced energy expenditure errors from over 70% in many cases to
under 2% in most cases and under 5% in the worst-case.

Periodic system activation as described in Chapter 2 is a particular solution to reducing
energy consumption for tasks sensing slowly changing or infrequent phenomena. In general,
these phenomena are not always periodic. For applications sensing more general sporadic
signals, intelligent wakeup mechanisms can serve to refine active system time frames in a
more reactive manner. Chapter 3 describes a potential solution for an event-driven wakeup
architecture with applications in water flow monitoring, as published by Paul Martin, Zainul
Charbiwala, & Mani Srivastava in [MCS12]. The system described in Chapter 2, DoubleDip
(DD), is a low power monitoring system for enabling non-intrusive water flow detection.
DoubleDip taps into minute thermal gradients in pipes for both replenishing energy reserves
and performing low power wakeup. We observe that water use in homes and offices is
incredibly sporadic, making continuous monitoring both impractical and wasteful. Instead,
DD puts a thermoelectric harvester into double duty. It uses thermal gradients not only
for gathering energy but also for extremely low power ($< 1\mu A$) wakeup. In Chapter 3 we
describe in detail the design of DoubleDip and demonstrate that thermoelectric wakeup is
essential for longevity and accuracy. Since DD wakes up from its low power state only when
there is a water flow event, it replenishes the energy it uses in sensing and transmitting data
by the energy it harvests from the corresponding heat gradient. While DD nodes installed on
cold water pipes harvest far less than those installed on hot water pipes, our pilot deployment
over four weeks and five locations suggests that thermoelectric wake up is only slightly worse
in latency for cold water monitoring and there is sufficient energy harvested from the hot
water that it can be shared to extend the lifetime of nearby cold water nodes too.

By reducing active computational time when sensing temporally sparse signals we can
save energy without losing very much information in the process. In many cases, however,
the signal itself exhibits sparsity in another dimension within these information-bearing time intervals. Compressed sensing (CS) is a new technique that promises to directly produce a compressed version of a signal by projecting it to a lower dimensional but information-preserving domain before the sampling process, exploiting sparsity to save energy. Chapter 4 describes a new hardware-oriented approach to performing low power compressed sensing for sparse (compressible) signals, published by Zainul Charbiwala, Paul Martin, & Mani Srivastava in [CMS12]. Designing hardware to accomplish this projection has remained problematic and while some hardware architectures do exist, they are either limited in signal model or scale poorly for low power implementations. Chapter 4 describes CapMux, a scalable hardware architecture for a compressed sensing analog front end. CapMux is low power and can handle arbitrary sparse and compressible signals, i.e. it is universal. The key idea behind CapMux’s scalability is time multiplexed access to a single shared signal processing chain that projects the signal onto a set of pseudo-random sparse binary basis functions. We demonstrate the performance of a proof-of-concept 16-channel CapMux implementation for signals sparse in the time, frequency and wavelet domains. This circuit consumes 20µA on average while providing over 30dB SNR recovery in most instances.
CHAPTER 2

Application Quality for Periodic Sensing Tasks

2.1 The Problem with Variation

In many cases, embedded sensing platforms are designed to meet a certain lifetime specification given knowledge of how much energy is available, the power consumption of the platform on average, and perhaps even the temperature profile for a certain location. Often times power management for such systems is accomplished by controlling the frequency and duration of active computation time intervals, exploiting the temporal sparsity of the phenomenon of interest. In recent years, however, newer integrated circuit fabrication technologies have introduced several additional variables into the energy management game; as feature sizes continue to shrink, power variation on a per-instance level has become a non-trivial factor [BKN03]; [GK03].

Per-instance power variations are particularly exaggerated for idle power consumption, motivating the need to mitigate the effects of variability in systems whose operation is dominated by long idle states. Figure 2.1 provides more insight into the matter: the International Technology Roadmap for Semiconductors (ITRS) predicts as much as 600% variation in static (idle) power and over 100% in total power by the year 2022 [ITR10]. One domain that stands to benefit from research into combatting hardware variation is that of low power embedded systems and low power sensors, where the application is often that of sensing, routing, or processing data.

In systems where computation is severely constrained by anemic energy reserves and where a long overall system lifetime is desired, maximizing the utility of a given application subject to these constraints is both challenging and an important step towards achieving high
quality deployments. Currently, developers assume some power consumption model prior to deployment, and this can have several undesired effects. Underestimation of system power consumption can lead to a reduction in lifetime which will eventually impact quality of service, while guardbanding against worst-case power consumption by using overly conservative estimates can reduce application quality for the entirety of the lifetime. The potential solution space is shown in Figure 2.2, where the optimal solution is one that maximizes quality without decreasing lifetime. Furthermore, the distribution of power in systems comprised of multiple heterogeneous tasks is oftentimes fixed in software prior to deployment as well, placing the burden of optimizing energy usage on developers who may remain oblivious to variations in power consumption altogether.

Perhaps the most widely used and most effective strategies for extending the lifetime of energy-constrained systems are those based on controlling the ratio of system active time to total system time, or duty cycling a system. Duty cycled systems take advantage of the disparity between active and idle power consumption, greatly increasing the lifetime of systems where latency and throughput constraints can be relaxed. Because of temperature and instance dependencies in power consumption, however, arriving at an optimal system-wide duty cycle ratio to achieve a lifetime goal given an energy constraint is difficult to do without a priori knowledge of instance-specific power models and temperature statistics for the target deployment location [WAB12]. Furthermore, applications involving more than one task necessitate notions of fairness and utility—specifically, how should active processor time be distributed between each task so as to maximize the utility of the application and still meet the desired lifetime goal?

In this chapter we explore the interplay between variable active and idle power consumption, deployment-specific temperature profiles, and multiple heterogeneous tasks. Specifically, we seek an answer to the question posed above; in an environment where power and temperature are measurable quantities, we seek an optimal strategy for distributing energy between arbitrary tasks in order to maximize application utility. In answering these questions, we introduce the notion of task knobs. These knobs offer both a way for tasks to express elasticity in terms of utility and processing time and a way in which an operating
system can fine-tune task energy consumption. Developers provide bounds on the values that each knob can assume and decide in what ways each knob is used, but optimization of these knob values is offloaded to the operating system and is done at runtime after accurate power and computational models have been constructed. These operating system abstractions are implemented in VaRTOS, a variability-aware operating system built as an extension to an open-source embedded operating system. In order to evaluate the abstractions and architectures that make up VaRTOS, we use custom variability extensions to a popular hardware simulation suite.

2.2 Related Work

Hardware-level approaches to address variability have included statistical design approaches [NS05, DBM05, KPR06], post-silicon compensation and correction [GC07, KS07, TKN02], and variation avoidance [CPR04, BMR07, GBR07]. Furthermore, variation-aware adjustment of hardware parameters (e.g., voltage and frequency), whether in context of adaptive circuits (e.g., [BKN03, GBR07, APM05]), adaptive micro architectures (e.g., [SBK06, EKD03, MJ06, TST07]) or software-assisted hardware power management (e.g., [DVA10, CLR09, TT08]) has been explored extensively in literature.

While low-level treatment of hardware variation is a necessary step forward, application- and process-level adaptations have proven effective methods for combating variation. The
range of actions that software can take in response to variability includes: altering the computational load by adjusting task activation; using a different set of hardware resources (e.g., using instructions that avoid a faulty module or minimize use of a power hungry module); changing software parameters (e.g., tuning software-controllable variables such as voltage/frequency); and changing the code that performs a task, either by dynamic recompilation or through algorithmic choice. Examples of variability-aware software include video codec adaptation [PGS12], memory allocation [BDN12], procedure hopping [RBG12], and error tolerant applications [CLM12]. In embedded sensing, [MTY06] and [GM07] provide lifetime analyses for wireless sensor networks when considering variability power models, offering insights into what such systems stand to gain from explicit treatment of hardware variation. Garg et al. estimated that a 37% system lifetime improvement could be achieved through redundancy efforts that totaled a 20% increased deployment cost.

This work attempts to mitigate and exploit variations in power consumption through the management of elasticity in application quality by a variability-aware real-time scheduler. Energy and longevity management in wireless sensor networks and low power embedded systems in general has long been an active area of research. Most previous work in this field, however, ignores the effects of power variations. Of these variability-agnostic techniques, many have focused on the tradeoff between energy and utility or performance. For example, [BC10, GAS12] represent attempts at making quality energy-proportional and tunable. Specifically, [BC10] introduces an architecture that allows developers to specify multiple versions of functions whereby the operating system can sacrifice quality when possible to reduce computational costs. Similarly, [GAS12] proposes tunable feature selection for wearable embedded systems, where less accurate feature computation can be used at the cost of inference quality. In real-time systems, [LSL94] represents one of many efforts at using approximate computing to save energy where marginal losses in quality can be afforded. In ECOSystem [ZEL02] and Cinder [RSL09], energy resources are periodically distributed to tasks which must spend the resources to perform system calls. In these systems, applications adjust their computational load according to energy availability. Our work differs from the previous approaches in that applications need not manage energy directly but instead
expose their elasticity in the form of a variable knob that is controlled by the operating system scheduler. Power consumption characteristics for each individual sensor are learned over time, and the system maximizes quality of service across tasks in a variability-aware fashion.

This work is closely related to that of [WAB12]. There, the authors describe a method for calculating a system-wide optimal duty cycle ratio given known models for active and idle power as well as probability density functions for deployment temperatures. Here we provide an extension to the work in [WAB12], showing methods for online learning of power models and providing notions of utility in multi-task applications.

2.3 Power Variability

As fabrication technologies improve and feature sizes decrease, hardware variation plays an increasingly important role in determining the power consumption and therefore lifetime of computer systems. This variability can be attributed to: manufacturing (due to scaling of physical feature dimensions faster than optical wavelengths and equipment tolerances [BFG06, CGK02]); environment (e.g. voltage and temperature); aging (e.g. due to negative bias temperature instability [ZVR09]); and vendors (multi-sourcing of parts with identical specifications from different manufacturers).

Power consumption in an embedded processor can be classified as either active power or sleep (idle) power. Active power includes switching and short-circuit power and can be modeled as in [RCN96] and [Vee84]:

\[ P_a = CV_{dd}^2f + \eta(V_{dd} - V_{thn} - V_{thp})^3f \]  

where \( C \) is the switching capacitance, \( V_{dd} \) is the operating voltage, \( f \) is the clock frequency, \( \eta \) is a technology- and design-dependent parameter, \( V_{thn} \) is the threshold voltage for NMOS, and \( V_{thp} \) is the threshold voltage for PMOS. The threshold voltage \( V_{thp} \) is subject to wear-out due to negative bias temperature instability (NBTI) as described in [CWC12, BWV06,
Sleep power can be modeled as

\[ P_s = V_{dd}(I_{sub} + I_g) \]  

(2.2)

where \( I_{sub} \) is the sub-threshold leakage current and \( I_g \) is the gate leakage current. Sub-threshold leakage current models can be derived from the device model in [UC 13], and simplified to extract its temperature and voltage dependency:

\[ I_{sub} = a_1 T^2 \left( \exp \left( -\frac{a_2 V_{thp}}{T} \right) + \exp \left( -\frac{a_2 V_{thn}}{T} \right) \right) \exp \left( -\frac{a_3 V_{dd}}{T} \right) \]  

(2.3)

where \( T \) is temperature in Kelvin and \( \{a_1, a_2, a_3\} \) are empirically fitted parameters that capture part-to-part variations. Gate leakage current is defined in [KAB03] as:

\[ I_g = a_4 V_{dd}^2 \exp(-a_5/V_{dd}) \]  

(2.4)

where \( a_4 \) and \( a_5 \) are empirically fitted parameters.

While the large baseline in active power consumption relative to idle power consumption amortizes variations to some degree ([WAB12] cites a 10% variation in active power while [BMG12] cites between 7% and 17% variation), the low baseline in idle power consumption renders it highly susceptible to fabrication-induced variations ([WAB12] reports a 14x range in measured idle powers across 10 instances of ARM Cortex M3 processors in 130nm technology).

In this work, we use 45nm process technology and libraries as our baseline for evaluation. Power model parameters are fitted to the SPICE simulation results of an inverter chain using the device model given in the technology libraries. The final power values are normalized to the measured data obtained from an M3 test chip using the same technology. Figure 2.3 shows active and sleep power across temperature for three instances representing the range of power variation for this technology (nominal, worst-case, and best-case). At room temperature, there is approximately 6x variation in sleep power consumption between the worst- and best-case instances. This magnitude of variation matches measurements with
Figure 2.3: Sleep (left) and Active (right) power for three processor instances in 45nm representing the range of variation for the technology.

off-the-shelf embedded class processors fabricated in 130nm shown in [WAB12], and hence represents a conservative estimation of the variation that may be found in processors in newer technologies.

Power consumption in energy-constrained embedded systems is often dominated by time spent in an idle state. Consequently, the lifetimes of these systems can be widely variant due to instance-to-instance variation in idle power, resulting in either premature system death or suboptimal system quality and an energy surplus.

2.4 Optimizing Utility with Variable Hardware

One way to combat increasing power variability is for an application to decrease or increase quality, thereby decreasing or increasing energy consumption. In this section we explore an architecture for adapting quality when applications have some degree of elasticity—that is, when quality is not a hard constraint. This will be accomplished by introducing task knobs—task-specific expressions of quality and power elasticity.

2.4.1 Task Modeling

We start first by introducing an application as a set of $N$ tasks denoted $\tau_i, i = 1, \ldots, N$, where each task represents a periodic application subprocess. We associate with each task
and with the application as a whole a utility \( u_i \) (or \( u_{sys} \) for the entire application) with the understanding that utility represents some notion of quality that the user is interested in. While some efforts espouse an architecture wherein each \( u_i \) is defined by an arbitrary function ([BC10] for example), we advocate a simplified model where the OS constructs \( u_i \) based on a few key inputs from the developer. In doing so, we assume that \( u_i \) is a monotonically non-decreasing function of the active computational time for \( \tau_i \) denoted \( t_{a,i} \) or, equivalently, the duty cycle ratio specific to \( \tau_i \) denoted \( d_i \) and defined as \( d_i = \frac{t_{a,i}}{t_{a,i} + t_{s,i}} \) where \( t_{s,i} \) is the amount of time that \( \tau_i \) is inactive. These variables along with additional key variables used throughout the text are summarized in Table 2.1.

**Task Knobs:** In order to tune the active time used per task and thus the task-specific duty cycle ratio \( d_i \), we introduce the notion of task knobs. In practical terms, a task knob is a variable that will govern either (1) the period of a task or (2) the frequency with which a task is activated. We argue that a large portion of tasks found in embedded applications will fall in one of these two classes, and those that require both frequency and period modulation can often be divided into two legal subtasks coupled with inter-process communications. For example, tasks that fall under class 1 include variable length sensing tasks, tasks that listen for inbound communication, and variable length processing chains. Those that fall under class 2 include variable frequency transmission, variable frequency sensor sampling, time synchronization handshaking, control and actuation events, and more.

We define task knobs, denoted \( k_i \), such that increasing \( k_i \) will increase \( t_{a,i} \), \( d_i \), and consequently \( u_i \). Task knobs are created by passing a variable address to the OS, allowing direct manipulation of knob values by an optimization routine. In addition, the developer specifies a minimum and maximum knob value, \( k_{i,min} \) and \( k_{i,max} \). The value \( k_{i,min} \) specifies the minimum value of \( k_i \) that yields a nonzero utility. Below this value, a task offers no utility. The value \( k_{i,max} \) specifies a value after which increasing \( k_i \) further will yield no added utility.

**Generating Utility Curves:** Changing each knob value \( k_i \) will cause a corresponding change in duty cycle ratio \( d_i \) based on the nature of \( \tau_i \). Given \( k_{i,min} \) and \( k_{i,max} \) as well as a mapping
Table 2.1: Summary of selected variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_i$</td>
<td>Task $i$, $i = 1, \ldots, N$</td>
</tr>
<tr>
<td>$t_{a,i}$</td>
<td>Active time for $\tau_i$</td>
</tr>
<tr>
<td>$t_{s,i}$</td>
<td>Inactive time for $\tau_i$</td>
</tr>
<tr>
<td>$d_i$</td>
<td>D.C. ratio for $\tau_i$</td>
</tr>
<tr>
<td>$d$</td>
<td>the vector $[d_1 \ldots d_N]$</td>
</tr>
<tr>
<td>$d_{sys}$</td>
<td>System-wide D.C.</td>
</tr>
<tr>
<td>$k_i$</td>
<td>Knob value for $\tau_i$</td>
</tr>
<tr>
<td>$k$</td>
<td>the vector $[k_1 \ldots k_N]$</td>
</tr>
<tr>
<td>$K_i$</td>
<td>model of $k_i \rightarrow d_i$</td>
</tr>
<tr>
<td>$u_i$</td>
<td>utility fcn. for $\tau_i$</td>
</tr>
<tr>
<td>$p_i$</td>
<td>priority scalar for $\tau_i$</td>
</tr>
<tr>
<td>$E$</td>
<td>energy budget</td>
</tr>
<tr>
<td>$L$</td>
<td>desired lifetime</td>
</tr>
</tbody>
</table>

from $k_i$ to $d_i$ (to be discussed later) we can construct a utility function $u_i = f(d_i)$ as a modified logistic (Sigmoid) function of the form $f(d_i) = \frac{1}{1+e^{-c_i d_i}}$, $c_i \geq 0$. A logistic function is used because the convex portion of the characteristic s-like curve offers a convenient form for modeling diminishing returns on $k_i$. The convex portion of the logistic function can be isolated by choosing $u_i$ to be of the particular form

$$u_i(d_i) = \frac{2}{1 + e^{-c_i d_i}} - 1, \quad c_i \geq 0, \quad d_{i,\text{min}} \leq d_i \leq d_{i,\text{max}}$$  \hspace{1cm} (2.5)$$

where $d_{i,\text{min}}$ and $d_{i,\text{max}}$ are task duty cycles corresponding to $k_{i,\text{min}}$ and $k_{i,\text{max}}$. Here, $c_i$ governs the convergence rate of $u_i$ from the minimum utility to the maximum utility and is calculated as a function of $k_{i,\text{min}}$ and $k_{i,\text{max}}$ such that 99% of the utility has been reached by $k_{\text{max}}$. Increasing the percentage of $u_{i,\text{max}}$ realized by $k_{i,\text{max}}$ has the effect of steepening the utility curve and thus increasing the rate at which returns diminish. The constant $c_i$ can be calculated from Equation 2.5 by enforcing $u_i(d_{i,\text{max}}) - u_i(d_{i,\text{min}})$ to be $\epsilon = 0.99$ as shown below:

$$c_i = -\log\left(\frac{\epsilon}{\epsilon + 1} - 1\right) \frac{1}{(d_{i,\text{max}} - d_{i,\text{min}})}, \quad \epsilon = 0.99$$ \hspace{1cm} (2.6)$$

Figure 2.4: Example utility curves. Task 1 has priority scalar $p_1 = 1$ with useful range $d_1 = [0.2, 0.7]$, task 2 with $p_2 = 2$ and $d_2 = [0.1, 0.3]$, and finally task 3 with $p_3 = 1.5$ and $d_3 = [0, 0.5]$. 
Finally, each utility curve can be arbitrarily increased or decreased by a priority scalar $p_i \in \mathbb{R}^+$ for tasks with intrinsically higher or lower utility than others. This offers a level of customizability in addition to specifying $k_{i,min}$ and $k_{i,max}$, allowing the developer to give preference to one task over another. Figure 2.4 shows three example utility curves corresponding to three tasks with various priorities and duty cycle ranges (resulting from various $k_{i,min}$ and $k_{i,max}$).

**Learning $K_i$, the $k_i \rightarrow d_i$ Relation:** Because the developer has free reign to use the knob $k_i$ for each task as desired, the function mapping $k_i$ to active time $t_{a,i}$ and thus $d_i$ is not known 	extit{a priori}. Instead, the transformation $K_i$ that maps $k_i$ to $d_i$ is assumed linear and is learned through regression at runtime. Should the developer misuse $k_i$ in a way that is nonlinear or that results in non-increasing values of $t_{a,i}$, the linear model will introduce errors that will affect the optimization process. Dividing active time accumulated per task by a fixed supervisory time interval $t_{super}$ yields task-specific duty cycle ratios, $d_i$.

### 2.4.2 Maximizing Application Utility

Given the set of tasks $\{\tau_1, \ldots, \tau_N\}$, our ultimate goal is to optimize $u_{sys} = \sum_{i=1}^{N} u_i$, the overall system utility. That is, we seek a solution to the convex optimization problem

$$
\begin{align*}
u_{sys}^* &= \max_k \sum_{i=1}^{N} \frac{2}{1 + e^{-c_i K_i[k_i]}} - 1, \quad (2.7) \\
\text{subject to:} \quad \sum_{T=T_{min}}^{T_{max}} f_T \mathcal{L}[k] &\leq \frac{E}{L} = \bar{P}
\end{align*}
$$

Where $T_{min}$ and $T_{max}$ are the minimum and maximum temperatures for a given location, $k = [k_1 \cdots k_N]$ is the vector of task knobs, and $\mathcal{L}$ is a linear mapping from $k$ to power consumption. The parameters $E$ and $L$ are the energy budget and desired lifetime as specified.
by the user and resulting in an average power goal, $P$. In general, $L$ is a function of temperature, and thus summing over the range of temperatures $[T_{min} \ T_{max}]$ and scaling by the probability density of each temperature $f_T$ gives the predicted power consumption corresponding to the knob vector $k$. When using external peripherals (such as radios, analog to digital converters, and flash storage), $L$ incorporates both external power and internal power (i.e., power consumed by the processor). In the remainder of this chapter, we will focus on optimizing utility when $L$ includes the power consumption of the processor alone. We leave inclusion of peripheral power models as a natural and straightforward extension to the proposed architecture.

We shift our focus now from that of optimizing utility with a power constraint to that of optimizing utility with a duty cycle constraint. In other words, power consumption of the system as a whole can take on values in the range $[P_s(T) \ P_a(T)]$ for a given temperature $T$ dictated by the overall system duty cycle ratio, $d_{sys} = \sum_{i=1}^{N} d_i$: $P = (1 - d_{sys})P_s + d_{sys}P_a$. Again, both $P_s$ and $P_a$ are functions of temperature so that, replacing $L$ with the processor power models, the optimization problem becomes

$$u^*_{sys} = \max_d \sum_{i=1}^{N} \frac{2}{1 + e^{-c_i d_i}} - 1,$$

subject to:

$$\sum_{T=T_{min}}^{T_{max}} f_T \left[ \sum_{i=1}^{N} (d_i P_a(T) + (1 - d_i) P_s(T)) \right] \leq \frac{E}{L} = \bar{P}$$

Here we have replaced the knob vector $k$ with the duty cycle vector $d$ similarly defined. The system-wide duty cycle can be arrived at if we know a priori the future environmental temperatures, the function mapping temperature to sleep power $P_s$, and the function mapping temperature to active power $P_a$. Given these, the optimal (maximum) duty cycle $d^*_{sys} \in [0, 1]$ follows naturally from Equation 2.8:
\[ d^*_\text{sys} = \max d \]

subject to: \[ \sum_T f_T [dP_a(T) + (1-d)P_s(T)] \leq \frac{E}{L} = \bar{P} \]

Under practical conditions as outlined in [WAB12], a close approximation for the optimal solution to Equation 2.9 can be obtained algebraically using Equation 2.10, where we have introduced the temperature-averaged power quantities \(\bar{P}_s\) and \(\bar{P}_a\):

\[ d^*_\text{sys} = \frac{E - L\bar{P}_s}{L(\bar{P}_a - \bar{P}_s)} \]  

Given \(d^*_\text{sys}\), we now seek an efficient solution to Equation 2.8. Because we have chosen \(u_i\) to be a logistic function, we can use a greedy approach when optimizing utility. The optimization routine will be a two step process: (1) attempt to assign the minimum duty cycle \(d_{i,min} = K_i[k_{i,min}]\) needed for each task in order of decreasing priority, and (2) continue distributing computational time in small increments to those tasks yielding the largest marginal utility until no active computational time is left. This process is outlined in Algorithm 1. Duty cycle is incrementally added by a small fraction \(\delta\) (chosen sufficiently small to ensure accuracy) to those tasks with the largest marginal utility defined as
ALGORITHM 1: Greedy Utility Optimization

Input: System duty cycle, \(d_{\text{sys}}\), linear functions \(\{K_1, \ldots, K_N\}\), and utility curves \(\{u_1, \ldots, u_N\}\)
Output: The optimal task duty cycles \(d^* = \{d_1^*, \ldots, d_N^*\}\)

\(\tau \leftarrow \text{sort}(\{\tau_1, \ldots, \tau_N\})\) by decreasing \(p_i\)

\(d_{\text{remaining}} \leftarrow d_{\text{sys}}\).

// Assign minimum knob values for tasks that can be scheduled:

\(\tau_{\text{scheduled}} \leftarrow \{\} //\) empty set

for \(i \in \tau\) do

if \(K_i[\tau_{i,\text{min}}] < d_{\text{remaining}}\) then

\(d_i = K_i[\tau_{i,\text{min}}]\)

\(d_{\text{remaining}} \leftarrow d_{\text{remaining}} - d_i\)

append \(\tau_i\) to \(\tau_{\text{scheduled}}\)

else

stop

end

end

// Allocate remaining duty cycle fairly:

while \(d_{\text{remaining}} > 0\) do

// Find highest marginal utility \(\text{max}_\mu\) and the set \(\tau_{\text{max}}\) of tasks yielding \(\text{max}_\mu\):

\([\text{max}_\mu, \tau_{\text{max}}] \leftarrow \text{Find Maximum Marginal Utility}(\tau_{\text{scheduled}})\)

\(d_{\text{requested}} \leftarrow \min\{\delta \cdot |\tau_{\text{max}}|, d_{\text{remaining}}\}\)

for \(i \in \tau_{\text{max}}\) do

\(d_i \leftarrow d_i + \frac{1}{\tau_{\text{max}}} \cdot d_{\text{requested}}\)

end

end

\(d^* \leftarrow \{d_1, \ldots, d_N\}\)

\[\text{max}_\mu = \max\{\mu_1, \ldots, \mu_N\}, \quad \mu_i = \frac{u_i[d_i + \delta] - u_i[d_i]}{\delta}\] (2.11)

until \(d_{\text{sys}}^*\) as calculated from Equation 2.10 is exhausted. Figure 2.5 shows an example of the utility maximization algorithm for two example tasks with the total system duty cycle on the \(x\)-axis and utility on the \(y\)-axis. At point (a), \(k_{1,\text{min}}\) is met and task 1 begins to receive active time. At point (b), task 1 begins to plateau as \(\mu_1\) diminishes. At point (c), both \(k_{1,\text{min}}\) and \(k_{2,\text{min}}\) can be met; starting with task 1 with the highest \(\mu\), utility is increased until point (d) where task 2 and 1 go back and forth bidding for active time. The dashed curve illustrates the total system utility, \(u_{\text{sys}} = u_1 + u_2\).

Each point in Figure 2.5 represents a different environmental set up—that is, a different \(d_{\text{sys}}^*\) (\(x\)-axis) resulting from, perhaps, different values for \(E, L,\) and \(f_T\). At each \(d_{\text{sys}}^*\), all tasks
are assigned a specific duty cycle ratio $d_i$. For example, Figure 2.6 shows the resulting duty cycles $\{d_1, d_2, d_3\}$ for three tasks when $d^*_{sys} = 0.2$. The vector $d$ that maximizes Equation 2.8 is denoted $d^* = \{d^*_1, \ldots, d^*_N\}$.

With a method in hand to calculate an optimal $d^*_{sys}$ offline, we seek a method for both calculating $d^*_{sys}$ and achieving $d_i \in \{d_1, \ldots, d_N\}$ in an efficient, online manner. Our implementation of this architecture is called VaRTOS and is the subject of the following section.

2.5 VaRTOS, the Variability Aware Real Time Operating System

In this section we outline the implementation of the architecture and algorithms presented in Section 2.4 as a series of extensions to an existing real time operating system (RTOS). The results shown are using a modified version of the FreeRTOS operating system [Fre13], though the architecture is easily applied to other embedded operating systems as well. The design of VaRTOS must accomplish several key aspects of Section 2.4 while remaining lightweight and energy-efficient. In particular, VaRTOS includes the following functionality: (1) a method for online modeling of $P_s(T)$ and $P_a(T)$; (2) a method for online modeling of $K_i$; (3) OS-level control over task knobs $\{k_1, \ldots, k_N\}$; and (4) a tool for evaluating the effects of user inputs $\{k_{i,min}, k_{i,max}, p_i, E, L\}$ as well as deployment location (temperature profile). We will describe each of these subsystems in detail below, with various prototypical case studies discussed in Section 2.8.

2.5.1 Online Modeling of Sleep and Active Power

As discussed in Section 2.3, both sleep power and active power are nonlinear functions of temperature. The vast majority of this nonlinearity comes from leakage and sub-threshold currents which dominate in $P_s$. In general, modeling these nonlinear curves could prove difficult with limited resources and without, in many cases, fully fledged math libraries. For example, nonlinear regression is often performed as an optimization problem using a specialized library such as NLopt, requiring more than 300 kB of program space in order to do even rudimentary optimization routines [NLo13] and prohibiting its use in many low power
platforms. Fortunately, the models in Section 2.3 describing $P_s$ result in a function that is very closely exponential. Knowledge of the shape of this function allows us to linearize the model which in turn allows the use of linear regression to accurately model $P_s$. Specifically, linear regression is run on $\log(P_s)$, giving offset $b_s$ and slope $m_s$. The desired sleep power model is likewise computed as $P_s(T) \approx \exp(b_s + m_sT)$. After $P_s(T)$ has been computed, $P_a(T)$ can be modeled by subtracting $P_s(T)$ from active power measurements and continuing with a second linear fit.

The error between the models described in Section 2.3 and the linear approximation methods described above is shown in Figure 2.7 for three separate power instances representing the best-case (BC), nominal-case (NC), and worst-case (WC) for a 45nm Cortex M3 processor—(a) shows the sleep power model with the corresponding error in (b), and (c) shows the active power model with the corresponding error in (d). For the linear approximation of $P_s$ on the temperature range $[-20^\circ C, 100^\circ C]$, the worst case error is around -15% while on a more temperate range of $[0^\circ C, 80^\circ C]$ the worst case error is around 5%. For most temperature profiles this accuracy will be adequate, but deployments in extreme

Figure 2.7: Modeling sleep and active power through linearization.
environments can experience the detriments of errors in the linear model of $P_s$. Because of the added baseline in $P_a$, the corresponding prediction error is drastically reduced—less than 2% across [-20°C, 100°C] for the best-case and nominal instances and less than 5% for worst-case. These errors can be further reduced using nonlinear regression methods if the computational resources are not a limiting factor; for VaRTOS we have chosen a lightweight design so that resource-constrained low power processors—those that are likely to be used in long lifetime sensing tasks—can easily perform the necessary computations.

Models for both $P_s$ and $P_a$ take some time to converge, before which an accurate prediction for the optimal duty cycle $d^*_{sys}$ cannot be calculated. Convergence is aided by variations in temperature, giving a variety of points on the $T \rightarrow \{P_s, P_a\}$ curves, and hurt by noise variance in power sensors. For example, if our sensor for $P_s$ takes hourly measurements with additive white Gaussian noise $\sim \mathcal{N}(0, 5\mu W)$, the percentage error of our model has reached a reasonable accuracy after 40 hours and is nearly fully converged after 60 hours. This is shown in Figure 2.8 for 190 different locations within the United States with between 1 and 9 years of hourly data in all locations and for processor instances with best-case, nominal-case, and worst-case power consumption. For the results that follow, both sleep and power models will be fit after 40 points (40 hours) of data have been collected.
2.5.2 Online Modeling of Task Computation

Active time per task $t_{a,i}$ can be measured in a number of ways (for example, using hardware timer snapshots at the context swap level). Given a method for measuring $t_{a,i}$, $K_i$ is arrived at by systematic perturbation of $k_i$ within the range $[k_{i,min}, k_{i,max}]$. Specifically, $k_i$ is repeatedly increased by $\Delta = \frac{k_{i,max} - k_{i,min}}{n}$ where $n$ is the number of points in the regression, kept sufficiently low $(n = 4$ in our case) to minimize memory footprint. Between each perturbation in $k_i$, the task is allowed to run for a time period sufficiently long enough to capture active time measurements for tasks with very infrequent activity. In the applications presented here, this supervisory period is set at $t_{super} = 1$ hour, meaning the mappings $K_i$ are calculated after 4 hours. Task duty cycles are calculated as $d_i = \frac{\sum t_{a,i}}{t_{super}}$. Note that $K_i$ is a linear transformation from $k_i$ to $d_i$ and thus $k_i$ should translate linearly into active time for that task. In Section 2.8 we explore the effects of violating this assumption.

Many tasks are likely to make heavy use of interrupt subroutines (e.g. for analog to digital conversion, radio transmission, serial communication, etc.). In order for this time to be accounted during the supervisory period, we provide functionality for assigning each subroutine to a particular task using a handle provided during task creation. For example, on entering a subroutine the `taskEnterISR( taskHandle)` command is invoked with a matching `taskExitISR` upon finishing the subroutine. Again, as mentioned in Section 2.4.2, in some cases additional peripheral power will be expended during these subroutines. The metric $t_{a,i}$ reflects only processor power expenditure, and thus peripheral power usage must be modeled separately by modifying $L_i$.

2.5.3 Controlling Task Active Time

In the same way that $k_i$ is perturbed to model $K_i$ in the previous section, $k_i$ is also commanded by the operating system to achieve $d_i^*$ as calculated by Algorithm 1. Knob control is passed from user to operating system at task creation, making the full task creation call using the modified FreeRTOS kernel:

```c
xTaskCreate( TaskFunction, "name", StackSize, Priority, &TaskHandle,
```
By its nature, TaskKnob serves as a discrete representation of $k_i$ and therefore introduces quantization errors into the optimization routine. In particular, the smaller the difference between $k_{i,\text{min}}$ and $k_{i,\text{max}}$, the coarser the granularity of TaskKnob becomes and therefore the poorer the achievable resolution of $d_i$ becomes. As an extreme example, if $k_{i,\text{min}} = 1$ and $k_{i,\text{max}} = 2$, then TaskKnob can only take on 1 of 2 values and thus 1 of 2 $d_i$ values, perhaps far away from $d_i^\star$. Similarly, even if TaskKnob is constructed in such a way that it has fine granularity, $d_i^\star$ might not be within the range $[K_i[k_{i,\text{min}}], K_i[k_{i,\text{max}}]]$. When $d_i^\star$ is less than $K_i[k_{i,\text{min}}]$, task $\tau_i$ consumes more energy than it is allotted and the system is likely to die prematurely. If $d_i^\star$ is greater than $K_i[k_{i,\text{max}}]$, however, it may simply mean that even though additional energy can be allotted to task $\tau_i$, no additional utility would be gained and so achieving a lifetime greater than $L$ is acceptable.

2.5.4 Temperature Models

Equations 2.9 and 2.10 require that we know the temperature distribution $f_T$ in order to calculate $d^\star_{\text{sys}}$ and the individual task ratios $\{d_1^\star, \ldots, d_N^\star\}$. We performed several simulations with results indicating that learning $f_T$ online is infeasible, as it takes the entirety of a year to develop an accurate histogram of the temperature values seen at a given location. Fortunately, similar simulations show that a very coarse representation of the temperature profile suffices for accurate calculations of $d^\star_{\text{sys}}$, and furthermore temperature profiles change very little from year to year for a given location. Figure 2.9 shows how certain temperature models affect the error in predicting average power consumption for $P_s$ across the lifetime of the system (in this case, 1 year). The $x$-axis here represents the number of years of temperature data used to train the model before testing on a single year. Each line represents a certain number of bins used in a histogram representing $f_T$ for a given location. This figure makes two noteworthy points: first, the decreasing estimation error indicates that temperature profiles change very little from year to year, and because of this using multiple years to build $f_T$ only serves to decrease the prediction error in years to come; second, while
a 3-bin histogram is inadequate to fully represent the temperature profile for a given location, there is very little benefit in representing $f_T$ with more bins than 5 and even less so with more than 10. Because of this, for a given location we train with as many previous years as are available and we use a 10-bin histogram to represent $f_T$.

### 2.5.5 User Programming Model

Much of the effort in creating VaRTOS is in making the process transparent to the developer and easing the burden of accounting for variable task power consumption. In addition to the challenges that come with embedded programming in general, developers need only provide the following information: (1) Energy budget $E$ measured in Joules; (2) Lifetime goal $L$ measured in hours; (3) Deployment location if it belongs in the VaRTOS database or corresponding coarse $f_T$ if not; (4) $k_i,min$, $k_i,max$, and priority scaling factors $p_i$;

Information required from the developer is therefore very minimal, though in many circumstances it is not readily apparent how different user inputs—particularly for knob values and priorities—will affect the operation of the system. In order to provide more intuition regarding the various parameters the developer is tasked with supplying, we have developed a simple tool in MATLAB as shown in Figure 2.10. This tool allows the developer to specify an energy budget and lifetime goal to guide the optimization process. Developers further specify clock frequency, instance type, a certain geographical location, and the various tasks to be scheduled. Perhaps the most difficult part of this tool is in estimating how many cycles each task will take per knob value. This tool only gives a rough estimate of how the true deployment will behave, but it helps guide the developer’s choices along the way.

### 2.5.6 Operation

In this section we discuss the operation of VaRTOS from a broader perspective, using the state chart depicted in Figure 2.11. To begin, the system is initialized with task creations, energy and lifetime specifications, and a location-specific temperature model. If at least one task has been created, the scheduler begins operation and we enter a model convergence
Figure 2.10: A tool for guiding developers using VaRTOS. Users input various system specifications along with task prototypes and a geographical location, and the corresponding optimal duty cycle $d_{sys}^*$ and knobs $k_i^*$ are displayed.

Figure 2.11: VaRTOS state chart, showing model convergence and optimization states.
While in this state, hourly temperature and power measurements are collected and knob values are incremented every $t_{super}$ seconds (see Section 2.5.2) to construct $\mathcal{K}_i$. The optimization routine cannot complete until both models have converged, after which linear regression and linearization are used to fit the knob to duty cycle and temperature to power curves, respectively. This brings us to the optimization state. Here the various $d_i^*$ are calculated as per Algorithm 1, and the corresponding knob values (calculated by inverting $\mathcal{K}_i$) are assigned to the appropriate tasks. At this point we begin steady state operation in the ‘Scheduler Running’ state. Potential reasons for leaving this state include task creation (necessitating learning the new task’s $\mathcal{K}_i$ and re-optimizing) or task deletion (requiring only re-optimization). Because the modeling tasks only run on an as-needed basis, these are implemented as OS tasks with null-valued knobs. This allows for easy suspension and resumption of these tasks as necessary.

The dashed line on Figure 2.11 represents an optional feedback error-checking mechanism that can help for online readjustment of poor initial power model construction (e.g. for cases where measurement of $P_s$ and $P_a$ is particularly noisy). This can be done by comparing true energy expenditure with predicted expenditure, if such a sensor exists, and using the error to apply proportional feedback, though we present no results regarding this extension in this work.

2.6 Experimental Setup

We implemented VaRTOS as a series of architecture-independent extensions to FreeRTOS [Fre13], a popular open-source real-time operating system. FreeRTOS provides typical operating system abstractions such as preemptive scheduling of multiple tasks, synchronization primitives, and dynamic memory allocation with low overhead and small memory footprint. For our evaluation, we use the TI Stellaris LM3S6965 port of FreeRTOS. The LM3S6965 is a microcontroller based on an ARM Cortex-M3 core, and is representative of the low-power platforms targeted by VaRTOS.

VaRTOS relies on a temperature and instance-dependent power model to perform its op-
timizations and requires appropriate sensors from its underlying hardware platform to build this model. Temperature sensors are typically embedded into most sensing platforms. Energy consumption and power in various processor modes may be measured directly (e.g. as in [MHY06]), or indirectly estimated from remaining battery capacity (e.g. with a “smart” battery or as in [LMM07]). Low-cost probes for variability vectors (including aging, frequency, and leakage power) may be embedded into processor cores, and exposed to software as digital counters [CGK12].

We evaluate VaRTOS with a series of case study applications under different hardware instances and deployment scenarios (temperature profiles) across a lifetime of 1 year. Because it would be impractical to physically deploy these applications, we rely on VarEMU [Var13], a variability-aware virtual machine monitor.

VarEMU is an extension to the QEMU virtual machine monitor [Bel05] that serves as a framework for the evaluation of variability-aware software techniques. VarEMU provides users with the means to emulate variations in power consumption in order to sense and adapt to these variations in software. In VarEMU, timing and cycle count information is extracted from the code being emulated. This information is fed into a variability model, which takes configurable parameters to determine energy consumption in the virtual machine. Through the use (and dynamic change) of parameters in the power model, users can create virtual machines that feature both static and dynamic variations in power consumption. A software stack for VarEMU provides virtual energy monitors to the operating system and processes. With the exception of the driver that interfaces with the VarEMU energy counters, VaRTOS running in VarEMU is unmodified from its version that runs on physical hardware.

When starting VarEMU, we provide a configuration file with parameters for the power model described in Section 2.3. For most test cases, we evaluate the system with three instances (nominal, best-case, and worst-case) as shown in Figure 2.3. When necessary for the evaluation, further instances are generated according to SPICE simulation results as described in 2.3. We also provide a trace of temperature based on hourly temperature data from the National Climactic Data Center [US12] for three locations: Mauna Loa, HI (‘best-case’: mild temperature, very little variation), Sioux Falls, SD (‘nominal-case’: aver-
Figure 2.12: Error in energy consumption for various optimal duty cycles and for mild, medium, and harsh environments. From left to right, figures represent the best case, nominal case, and worst case power instances in 45 nm.

...age temperature and variation), and Death Valley, CA (‘worst-case’: extreme temperature and variation). For every hour elapsed on the Virtual Machine, VarEMU reads a new line from the temperature trace file and changes the temperature parameter in the power model accordingly. In order to accelerate the simulation (which would otherwise run in real-time), we use a time scale of 1:3600, resulting in a total simulation time of approximately 2.5 hours for a lifetime of one year.

2.7 Evaluation

In this section we present results regarding the ability of VaRTOS to maximize utility by modification of task knob values as well as the corresponding error in energy consumption vs. the specified energy budget. Finally, we evaluate the VaRTOS architecture in terms of both energy and memory overheads.

2.7.1 Minimizing Energy Consumption Error

In order to achieve accurate energy consumption to meet a lifetime goal, VaRTOS needs to accurately be able to achieve the overall system duty cycle $d_{sys}^*$. To test this, we constructed a simple application with only a single task containing a knob with fine granularity values. Then, using the tool shown in Figure 2.10, we specified various values for $E$ and $L$ that would ideally lead to a particular $d_{sys}^*$ for each of the power instance models (best-, nominal-, and worst-case) as well as three temperature profile instances (harsh, medium, and mild). The
target duty cycles were $d_{sys}^{*} \in \{0.002, 0.005, 0.01, 0.05, 0.1\}$, or from 0.2% up to 10%, and the resulting errors in energy consumption are shown in Figure 2.12. Note that errors are larger in harsher environments, where any errors in the power models will be magnified. In the worst case, an error of 4.9% in energy consumption is seen for a harsh environment and for the worst-case power instance (far right plot in Figure 2.12). This means that, in the worst case, a 5% guard band in lifetime or in energy is necessary if the lifetime goal is to be treated as a hard constraint.

To give more intuition into what this error in energy consumption means, we compared energy consumption for tasks running in VaRTOS (modeling power on a per-instance basis) with those assuming ‘worst-case’ power consumption. True worst-case power consumption is difficult to define, due to the long tail distribution for power across temperature. Because of this, we define worst-case power as the average power consumption for the worst-case instance from Section 2.6 across the temperature range $[0^\circ C, 45^\circ C]$, particularly $P_s = 330 \mu W$ and $P_a = 1.187$ mW. Figure 2.13 shows the disparity between the two. Without per-instance power modeling, energy consumption is in some cases over 70% off, and in only one case is it below 10% error. With per-instance power modeling using VaRTOS, the error has dropped to below 2% in most cases and around 5% in the worst case. Note that a positive percent error means a surplus in energy after the lifetime has been met while a negative means an energy deficit (and likewise a premature death). Energy errors are mostly positive here because of the worst-case power assumption.

Similarly, Figure 2.14 shows the cause of this energy error disparity—the duty cycle ratio remains constant if worst-case power is assumed while it is allowed to vary when instance-specific power modeling is introduced. This will translate into an increase in the quality of service for a particular application (e.g. more data collected, a higher communication rate) by using what would have been surplus energy.
2.7.2 Utility and Oracle Comparison

The results in the previous section showed that VaRTOS is able to meet a given energy budget with low error, resulting in an accurate system lifetime. In Section 2.4 we argued that spending would-be surplus energy will increase system utility. In this section, we substantiate this claim by comparing the utility of the single task app running in VaRTOS to that of an all-knowledgeable oracle system. Unlike the true VaRTOS system, the oracle system is allowed the following privileges: (1) complete knowledge of the temperature profile for the test year; (2) perfect knowledge of task behavior (i.e. \( K_i \)); (3) full accuracy models for \( P_s \) and \( P_a \); and (4) zero overhead for optimization routines. Figure 2.15 shows the utilities for both the oracle and VaRTOS. In most cases VaRTOS achieves within 10% of the oracle utility, and is as much as 20% off in the worst case. Note that this comparison is specific to
the construction of utility \( u_i \) as defined in Equation 2.5, and other utility curves may cause variations in this metric.

### 2.7.3 Energy and Memory Overhead

Energy consumption by the various VaRTOS subsystems must be minimized in order to prevent worsening the very thing we are trying to correct. Similarly, the memory required for VaRTOS must be kept reasonably low in order to make it a viable option for resource-constrained platforms.

**Memory Overhead:** The amount of program memory (.text, .data) and volatile memory required for VaRTOS depends on the application that the developer is designing. As a baseline, VaRTOS requires a modest increase in the ‘.text’ section over the vanilla FreeRTOS framework from 2.29 kB to 6.80 kB (a 4.51 kB increase). This includes a lightweight library for math functions required for optimization routines (including exponential, logarithmic, and square root functions) as well as a preemptive scheduler. If a full math library needs to be used for the application itself, these functions can be replaced and the overhead amortized.

In terms of volatile memory, an additional 508 bytes baseline is required (480 bytes of this is due to the power learning procedure and, if the developer is so motivated, can be reused after the models have converged). An additional 46 bytes per task is also required for knob modeling and other parameters. Finally, the temperature profile is stored in program ROM as a constant array and consumes only 10 additional bytes.

**Energy Overhead:** The largest energy overhead in VaRTOS comes from the scheduler itself, which, if context swaps occur every 10 ms, causes a baseline system duty cycle of 0.1%. This ratio can be decreased if coarser granularity context swaps are acceptable. The power consumption attributed to this 0.1% depends on the power consumption of the processor and the environmental temperature, but in the worst case it consumes 

\[
P_{os} = 0.001 \cdot (1.187 \text{ mW}) + 0.999 \cdot (330 \mu \text{W}) = 331 \mu \text{W} \approx P_s.
\]

In other words, the scheduler adds only marginal power
consumption on top of the baseline sleep power.

Other potential energy consuming processes attributed to VaRTOS include knob modeling, power measurement and fitting, finding the optimal $d_{sys}^*$, and finding the optimal knob values. The amount of processing time spent in these tasks is negligible: reading power and temperature takes 250 $\mu$s and occurs only 40 times over the course of a deployment (10 ms total); knob perturbations take 48 $\mu$s and occur $4 \cdot N$ times (for $N$ tasks); performing a 40-point linear regression (for power curves and as an upper bound for modeling $K_i$) takes 40 ms and occurs twice ($P_s$ and $P_a$) per deployment and once per task; finding $d_{sys}^*$ takes 54 ms and occurs once unless tasks are deleted and created after the initial optimization; and finally finding optimal $d_i^*$ and $k_i^*$ values takes 345 $\mu$s. In total, these added tasks consume less than 1 mJ in the worst-case for a 1 year deployment, a negligible overhead if our energy budget is 12960 Joules (2 AAA batteries) as in the following section. Note, however, that (1) taking power and temperature measurements is likely to consume additional power for analog to digital conversions and (2) a more difficult calculation in energy overhead comes from the result of perturbing knob values in the modeling phase for $K_i$. The latter depends on the nature and number of tasks, as well as the length of $t_{super}$.

2.8 Case Studies

We have shown in Section 2.7 that VaRTOS can accurately achieve a desired lifetime goal given an energy budget, and we made the claim that using would-be surplus energy will increase application performance. Here we provide several simulated case studies to substantiate this claim, using the same experimental setup described in Section 2.6 with additional virtual peripherals as needed. For these case studies, the energy budget is set at a constant 12960 Joules, corresponding roughly to that of 2 AAA batteries in parallel. In addition, the lifetime goal is set at 8760 hours, or 1 year.
2.8.1 Multi-Agent Applications

Oftentimes a system is composed of multiple nodes connected by either wired or wireless communication. As an example, consider a network of 8 nodes with wireless radio capabilities. Each node is capable of sensing a noisy measurement that corresponds roughly to the distance of some unknown object to the node, whose location is known. For example, these nodes could be taking audio measurements and inferring the distance of a vehicle traveling around a track. From these distance measurements, we would like to estimate the \((x, y)\) coordinate of the unknown vehicle. This system is illustrated in Figure 2.16, where the unknown object makes a counter-clockwise path and the known sensor nodes take noisy measurements if the object is within their radius of observation, shown with dashed circles. Here the unknown object is traveling on a track of 300 m circumference at the speed of 0.4 km/h. The 8 known nodes can ‘observe’ a linear distance to the unknown node if it is within 80 m. Each node operates two tasks: (1) a radio with a variable frequency transmission, and (2) a sensor that samples a variable number of points and averages the samples. Allowing the radio more active time will reduce the latency in reported estimates of the unknown
node, while allowing the sensing task more time will generate more reliable estimates. The task priorities $p_i$ along with the tool described in Section 2.5.5 would help a developer give preference to one or the other. For the sake of our comparison, we keep the priorities the same and choose knob ranges to allow the sensor to average between 1 and 100 samples and to allow the radio to transmit anywhere from 10 Hz to 0.1 Hz. Because these peripherals are simulated, each task has been padded with NOP instructions in order to simulate work that an actual system might be doing. If we look at a 1000 minute time slice of the estimation process as shown in Figures 2.17 and 2.18, we see that the variance of the estimation error is much greater if we assume worst-case power and thus average fewer samples and much less if we use VaRTOS with instance-specific power models. When many samples are averaged, the noise is reduced and each estimate is the result of consensus between more reliable measurements. In other words, for the same lifetime specifications, the system using VaRTOS greatly outperforms the system that assumes worst-case power consumption. While the deployment assuming worst-case consumption suffers from an average error variance of 59.7, the VaRTOS deployment has an average error variance of only 26.9, a 54.9% improvement. The periodic nature of the high variance peaks in estimation error for both Figures 2.17 and 2.18 can be attributed to the circular nature of the application setup (Figure 2.16). When the unknown object comes within view of nodes with less reliable measurements, the estimation error is much poorer.

Furthermore, the reduction in error variance when using VaRTOS does not come at the

Figure 2.17: Reduced error variance for multi-agent localization using instance power modeling with VaRTOS

Figure 2.18: Increased error variance for multi-agent localization assuming worst-case power
Table 2.2: Task performance for multi-agent localization with and without VaRTOS with a harsh temperature profile

<table>
<thead>
<tr>
<th>Node ID</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
</tr>
</thead>
<tbody>
<tr>
<td>VaRTOS # Avgs.</td>
<td>35</td>
<td>34</td>
<td>31</td>
<td>30</td>
<td>28</td>
<td>27</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>WC # Avgs.</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>VaRTOS Freq (Hz)</td>
<td>1.798</td>
<td>1.719</td>
<td>1.583</td>
<td>1.527</td>
<td>1.459</td>
<td>1.380</td>
<td>1.176</td>
<td>0.525</td>
</tr>
<tr>
<td>WC Freq (Hz)</td>
<td>0.287</td>
<td>0.287</td>
<td>0.287</td>
<td>0.287</td>
<td>0.287</td>
<td>0.287</td>
<td>0.287</td>
<td>0.287</td>
</tr>
</tbody>
</table>

price of increased radio latency; the radio latency on the average will improve by using VaRTOS as well, as shown for the case of harsh temperature profiles in the resulting optimal task performance in Table 2.2. The errors in energy consumption for this application are equivalent to those shown in Figure 2.13, and thus we omit them here for the sake of brevity.

2.8.2 Prediction-type Applications

We now move away from wireless sensor network applications and look at systems of just a single node. In particular, we consider an application where we would like to predict one quantity from another correlated but noisy quantity: in our case, we will predict velocity from position, perhaps again on a vehicle of some kind. Here our tool of choice will be the Kalman filter, as it has become such a widely used tool even for resource-constrained applications. We are interested in calculating an estimate $\hat{v}$ of the velocity $v$ from measurements of the position, $y$, at various frequencies controlled again by a knob. The system evolves according to the simple state space recursion:

\[
x_{k+1} = A_k x_k + B_k u_k; \quad y_k = C_k x_k; \quad A_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}; \quad B_k = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \quad C_k = \begin{bmatrix} 1 & 0 \end{bmatrix}
\]

For our case, $u_k$ will be a sinusoidal velocity input, and the goal is for $\hat{v}$ to track this input. Intuitively, a faster sample rate for $y_k$ (meaning $\Delta t$ changes in $A$ as well) should give more accurate predictions of $\hat{v}$, because it is easier to predict states within the near future than it is to predict them in the distant future. The Kalman recursion itself is omitted here, but
we note that the Kalman gain and error covariance matrices (commonly denoted $K_{p,k}$ and $P_{k+1|k}$, respectively) have to be modified on a per-instance basis in order to accommodate the varying sampling period, $\Delta t$.

Figure 2.19 shows an example of the velocity input, $u_k$, as well as the estimated velocity as calculated by the Kalman filter on position $y_k$. Here the position readings are subjected to additive white Gaussian noise $\sim \mathcal{N}(0, 50 \text{ m})$, and likewise the Kalman estimation of velocity contains some noise as well. If we allow $\Delta t$ to assume values within the range $\Delta t \in [0.1 \text{ s}, 10 \text{ s}]$ by letting our knob vary as before, the quality of the estimate $\hat{v}$ will increase or decrease in accordance with energy surpluses and deficits, respectively. Figure 2.20 shows the error (RMSE) in estimating the velocity from noisy position measurements for systems assuming worst-case power and for those using instance-specific modeling with VaRTOS. As before, the $x$-axis represents combinations of power instances (best-, nominal-, and worst-case) as well as temperature profiles (mild/best, medium/nominal, and harsh/worst). While the worst-case system has a constant error across all combinations, the VaRTOS results show a reduction in prediction error when additional work can be performed without sacrificing lifetime (i.e. those cases where weather and power instance result in a reduction in energy consumption over the worst-case). This improvement can be as much as 42.5% in many cases.
Table 2.3: Processor cycle counts for a multi-block signal processing application

<table>
<thead>
<tr>
<th>Block #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor 1 Block Cycles</td>
<td>20000</td>
<td>5000</td>
<td>10000</td>
<td>25000</td>
<td>6000</td>
</tr>
<tr>
<td>Sensor 2 Block Cycles</td>
<td>14000</td>
<td>9000</td>
<td>15000</td>
<td>24000</td>
<td>8000</td>
</tr>
</tbody>
</table>

Figure 2.21: Energy consumption errors for a multi-block signal processing application

2.8.3 Block Processing Applications

In some cases, a developer may have a number of signal processing (or similar) routines that would help to clean up or extract data from a particular signal. In most cases, the total number of potential sensor processing blocks is likely to be small, and more importantly it is unlikely that each of these blocks will take the same time and thus the same power to complete. In other words, the functions $K_i$ that map knob values into duty cycles are no longer a very good fit, as the curve $k_i \rightarrow d_i$ is no longer linear. As an example, consider a system with two sensor streams, each with 5 potential sensing blocks that all increase the quality of the application as a whole. These tasks each take a distinct number of computer cycles to complete, as summarized in Table 2.3.

The cycles here have been arbitrarily chosen in part to show how VaRTOS responds to violations in architecture assumptions (linearity and granularity of task knobs). If we try to schedule these two tasks using VaRTOS with $E = 12960 \ J$ and $L = 8760 \ h$ as before, the resulting errors in energy consumption will be those shown in Figure 2.21. In this case, the number of blocks executed using VaRTOS is on average 2.8 for sensor 1 and
3.5 for sensor 2 while worst-case power assumption resulted in 1 block between both sensors combined. In other words, while VaRTOS gives over a 64% signal processing improvement in this application, the errors in energy expenditure are on average much larger than the errors shown in Figure 2.13; indeed they would be even larger had our cycle counts shown in Table 2.3 varied even more or had the number of blocks been even fewer than 5. While VaRTOS can help reduce energy errors to some extent in applications similar to this multi-block signal processing example, care should be taken to ensure that the assumptions made in Section 2.4 are not completely ignored.

Discussion

We developed an architecture for maximizing application quality while meeting lifetime and energy constraints in the face of power and temperature variation. We achieve this through application elasticity as defined by a knob—a tunable variable that increases the quality of a given task at the expense of increased power consumption. This knob serves to shape the utility curve of each task as well as offer a means by which the operating system can control the amount of active time and thus power given to individual tasks. We implemented online per-instance power modeling and task modeling in VaRTOS, a series of kernel extensions to the FreeRTOS operating system. Our simulations using VaRTOS show that we can accurately meet a specified lifetime goal with less than 2% error in most cases and less than 5% error in the worst case, whereas had we assumed worst-case power consumption errors would range from 5% to over 70%. We further demonstrated the ease with which a developer can adopt the VaRTOS architecture; very minimal user input is required, and the effects of these inputs can be tested using a graphical task modeling tool. Finally, we presented case studies for multi-node localization applications using wireless sensor networks, estimation problems using Kalman filtering, and multi-block signal processing applications, illustrating how VaRTOS can increase application quality while maintaining lifetime requirements. Knobs in VaRTOS represent flexible notions of elasticity—we make the assumption in this work that they have a linear relationship with computational time, but in general the only
assumption required is that an increase in knob value will increase utility. In other words, if more advanced modeling techniques are used, we can extend this notion of knobs to adapting many other system parameters, not just task frequency and duration. The VaRTOS architecture allows traditional, non-adaptive tasks to co-exist with adaptable tasks, and it is therefore suitable for a large class of applications where a notion of elasticity in quality exists.
CHAPTER 3
Sensing Sporadic Data Streams

3.1 Event-triggered Sensing for Water Monitoring

In the previous chapter we proposed an operating system abstraction to maximize the quality of applications by exploiting periodicity in temporally sparse signals and in application tasks. In general, however, signals of interest and applications running on embedded processors need not be periodic. In many cases, phenomena that are temporally sparse may also occur irregularly, requiring new strategies to capture these sporadic events. We now explore energy saving techniques for sporadic signals in more detail, using water flow monitoring as an example application for the proposed techniques. Specifically, we will show how event-driven architectures along with opportunistic energy harvesting can drastically improve longevity in low power sensor deployments.

Water monitoring in homes and office buildings promises to help alleviate some of the water reserve burdens we face today [RHM96]. Reliable measurements of consumption are essential for evaluating the impacts of conservation projects on urban water demand [MDT03]. Providing consumers with a fixture-level understanding of their usage through appropriate interfaces provides opportunities to make informed decisions about investments in high efficiency fixtures and appliances [GEB83] and serves to encourage them toward more effective conservation. Sadly, a significant portion (~30%) of water use is actually wasted through leaks and oversight [MDT03] which might have been detected early with continuous water monitoring coupled with intelligent analytics.

This chapter describes DoubleDip, a source-aware architecture and platform designed to enable non-intrusive water flow monitoring. DoubleDip senses vibrations during a water
Figure 3.1: Sample data of pipe temperature, wakeup triggers from DoubleDip, and pipe vibration. Spurious vibrations would cause frequent false wakeups with vibration-based LPL even though water use is sporadic.

“event” much in the way that Kim et al. outline in NAWMS [KSC08], except that DoubleDip leverages small thermal gradients that accompany water flow for both energy and wake up. The flow of water and thus the heat gradients from which DoubleDip harvests energy is incredibly sporadic, due to the nature of human interaction with both hot and cold water in both residential and office environments. The hot water input feeders to a kitchen or bathroom sink, for example, might only see a dozen uses over the course of a day or experience blackouts for a month at a time during user absences. The top half of Figure 3.1 shows the sparse temperature profile over a sample day for the surface of a hot water pipe leading to a faucet in a residential apartment. The profile follows a signature pattern of rapid transient when the faucet is turned on and slow settling after shut off as the water near the faucet loses its heat to the ambient environment. The total time that the faucet is on is almost imperceptible in this plot – 8 min over the 24 hour period.

3.1.1 Low Power Wakeup

Using unpredictable, bursty, and sporadic energy sources as a wake-up source as well as a means to harvest energy presents a host of challenges. Since the phenomenon being sensed is also the one being harvested from, keeping the sensors active continuously to monitor a rare, ephemeral event is both impractical and wasteful [DGA05]. One alternative to waking the system up from sleep for a water flow event is to use a form of low power listening (LPL)
with the vibration sensors already in place. A hypothetical vibration-based LPL mechanism can be imagined that activates an accelerometer (\(\sim 300\mu\text{A} \) current) for a short period of time to estimate whether a water flow event has started. If not, the sensor is deactivated and the node goes back to sleep for the duty cycle period. A concern with this approach is that an event can only be detected with some latency, which could be the duty cycle period in the worst case. Assuming that our target average current consumption is 1\(\mu\text{A} \) and that it takes 250ms to detect the event reliably, the duty cycle period would have to be 75s to meet the 1\(\mu\text{A} \) target. Another issue is that of false wakeup. The bottom half of Figure 3.1 shows the vibration (standard deviation of accelerometer) readings over the same time duration as the top half, which shows many high amplitude spurious vibrations that would trigger even a finely tuned vibration-based LPL system. These vibrations originate from adjoining pipes in the plumbing network, the HVAC system, and general activity around the residence. While the NAWMS [KSC08] system cancels out these “crosstalks,” the false wakeup is still a problem for power management.

To resolve these issues, DoubleDip employs a thermoelectric generator (TEG) in double duty. It uses thermal gradients not only for gathering energy but also for extremely low power (< 1\(\mu\text{A} \)) but sensitive wakeup. When a water flow event begins, there is a small but detectable voltage change across the TEG due to minute thermal gradients in the water stored in the pipe. Although this voltage is too small to harvest from, it is large and reliable enough to wake from. Figure 3.1 shows the triggers generated by DD’s wakeup circuitry compared to both the temperature profile and the vibration readings. While vibration has a number of excursions above a reasonable activity threshold even when temperature does not change, the DD triggers correctly identify water flow events (as evidenced by the temperature transient subsequent to each trigger and a simultaneous spike in vibration). Additionally, DoubleDip wakes up almost instantaneously even while readings reported by temperature sensors are still below their noise floor, for example near the 21 hr mark in Figure 3.1. See Figure 3.12 and Section 3.3.5 for a detailed handling of wakeup performance and why temperature sensors cannot be used for wakeup directly.

In all, the key idea of the DoubleDip design is that it relies on an artifact of the phe-
nomenon it is trying to capture to wake the system out of deep sleep through thermoelectric harvesting, removing the need for duty cycling, saving state, and preventing cold-boot delays [YCB12].

3.2 Related Work

The past decade has witnessed a sharp rise in the popularity of energy harvesting solutions for wireless sensor networks and power management in general. For small scale systems, harvesting solutions in the form of thermal, kinetic, solar, and piezoelectric transducers have found applications in new and exciting arenas. Improvements in low power digital and analog ICs such as the trail of low power motes marking the past decades [Sen, mic07a, mic07b, DTJ08] as well as standardization of popular energy-aware embedded operating systems [LMP04] have helped elucidate new low power computing paradigms and increase the lifetime of sensor networks, in some cases, into years.

The combination of today’s energy harvesting techniques and the state-of-the-art in low power processing and wireless communication is indeed exciting, and there has been considerable research on this front in recent years. EnOcean has developed a range of commercially available products spanning solar and thermal harvesting and supporting various wireless protocols [EnO12], and both Heliomote [LYH05] and Yerva et al. [YCB12] were successful in routing wireless packets despite frequently having to cold boot [YCB12]. In contrast, the extreme scarcity of energy in water pipes dictates that DD nodes forgo routing altogether.

3.2.1 Thermoelectric Harvesting

Within the realm of thermoelectric energy generators (TEGs), much of the prior research can be divided into three categories – characterization and viability of TEGs for use in powering wireless sensor nodes, development of wearable body warmth energy harvesters for Body Area Networks (BANs), and application-specific systems for gleaning power from various waste sources.
Towards the characterization of TEGs for use in autonomous sensors, Ferrari et al. have measured the open circuit voltage and output power characterization for a range of devices, concluding by powering a low power mote with wireless transmitter when supplying the TEG with a temperature gradient [FFG07]. The careful measurement and design of the system presented in [FFG07] lends credence to the efficacy of TEGs (and Peltier junctions) in powering sensor boards in addition to providing theoretical groundwork to compare expected output power from a given harvesting architecture and by which we are able to calculate several efficiency metrics herein.

There have been numerous attempts to create ‘wearable’ or ‘comfortable’ TEGs for body area networks (BAN), such as those described in [LGH08, HTC09, LTF07] though in most cases the inclusion of a heatsink creates an unfortunate trade-off in power harvested and comfort. It must be noted that TEGs harvest energy from a temperature differential across two surfaces, but without an effective heatsink the cold surface slowly attains the temperature of the warm side by thermal conduction through the Peltier junctions reducing the power harvested despite a continuous heat source. Finding an appropriate heatsink is a challenge in DD as well—perhaps the most promising in this regard is a new flexible Peltier junction created by spraying doped silicon onto a flexible substrate [per12] and air-cooled over a large area.

Many existing systems have the luxury of constant or pseudo-stationary energy sources—time constants may be in terms of hours or days. Zhang et al. demonstrated a “steam-powered” thermoelectric sensor network [ZSC11] using a Mica-2 [mic07a] node and a Peltier device much like the one used in DD. In contrast, [ZSC11] has a fairly constant temperature gradient reaching as much as 80°K and is able to generate over 0.5 W from an industrial steam pipe.

Despite the dissimilarities between DD and other platforms, DoubleDip faces many of the same obstacles encountered in other architectures and applications. In order to provide cold-boot power and effectively bootstrap their system, TwinStar uses a secondary solar panel [ZZG09]. Similarly, DD uses a low leakage lithium battery to avoid bootstrapping and race-condition issues. DoubleDip could be considered to be battery powered when opportunities
for harvesting do not occur but recharges its battery when excess energy is available. Whereas TwinStar accepts leakage and designs control algorithms to counter it, DD reduces leakage by using a battery in lieu of a supercapacitor—a luxury afforded it given the low discharge depths of DD nodes on average and thus high cycle lifetimes of the rechargeable batteries used.

Recharging batteries from thermal harvesting has also been explored by Sodano et al. [SSD07] for the application of Structural Health Monitoring (SHM). By directing solar irradiation to heat up one side of a TEG and using a large mechanical structure such as a bridge to cool the other side, [SSD07] was able to charge a 300 mAh to capacity in a mere 3.5 min. An exhaustive summary of energy harvesting techniques for SHM including TEGs is presented by Park et al. in [PRT08]. Therein, the authors cite interesting uses of thermal harvesting from such non-traditional sources as soil–air temperature gradients, vehicle exhaust pipe heat, and radioactive decay of isotopes in space (NASA) [PRT08].

3.2.2 Water Flow Monitoring

Water flow monitoring and leakage detection continue to be active areas of research. Proposed techniques range from disaggregation of load monitors to distributed sensing architectures and even pipe topology mapping [LC10], each having inherent advantages and disadvantages. Non-intrusive disaggregation approaches such as Hydrosense [FLC09, CLC10] use signature analysis approaches to determine the temporal profile of individual device usage as well as water flow rates. This approach offers high accuracy for detecting the on/off signatures for a number of devices after a manual training phase. While such systems have demonstrated success for controlled experiments, Froehlich et al. note that it is unclear how well these techniques scale when many devices overlap in time and in the presence of variable flow loads such as sinks [FLC09]. On the other end of the spectrum, distributed approaches seek to give fixture-level accuracy without requiring a burdensome installation. TriopusNet [LCL12], Fogarty et al. [FAH06], and NAWMS [KSC08] fall in this latter camp. In TriopusNet, sensors are deposited inside the feeder to a pipe network (residence, office,
etc.) and then routed to unique pipe segments through a systematic turning on and off of faucets in order to direct nodes to appropriate branches for initial deployment and any maintenance thereafter. While this could potentially be automated, the infrastructure for doing so does not yet exist. Fogarty et al. attempt to map auditory signatures to water flow using distributed microphones, though they face problems with ambient noise and cross-talk.

NAWMS makes inferences about fixture-level water flow from individual pipe vibration data, self-calibrated using water flow information from a single feeder meter. The resulting system is non-intrusive and highly accurate, providing fine-grained flow estimates in environments where in-line sensors, manual calibration, and load disaggregation are prohibitively difficult.

### 3.3 System Architecture

The architectural design of DoubleDip is necessarily interdisciplinary – care must be taken not only to minimize electrical transients and steady state current but also to design thermal conduction paths to maximize temperature differentials across the TEG surfaces.

#### 3.3.1 Thermoelectric Generators and the Seebeck Effect

As is the case with other thermoelectric devices, DoubleDip relies on the Seebeck effect – the corollary to the Peltier effect responsible for converting electricity into temperature differentials. DoubleDip uses a Peltier junction, an array of n-p junctions, to convert the difference in temperature across the array into an electrical signal, as shown in Figure 3.2 and given by:

\[
V_{TEG} \approx \int_{T_{hot}}^{T_{cold}} (S_{cold} - S_{hot})dT = (S_{cold} - S_{hot})\Delta T
\]  

(3.1)

where \(S_{hot}\) and \(S_{cold}\) are the Seebeck coefficients of the materials used in the Peltier junction. This is a slight approximation, because in reality the Seebeck coefficients are non-linear functions of the temperature on an absolute scale.

\(V_{TEG}\) is therefore a function of the temperature gradient across the Peltier device, and so
the use of a proper heatsink is of utmost importance whether producing temperature via the Peltier effect or harvesting it via the Seebeck effect. This further complicates matters when trying to analyze the lifetime of such a system, as transients in temperature in the pipes will produce a power dependent on the temperature of the heatsink at any given time.

3.3.2 Mechanical Design

As shown in Figure 3.2, the TEG is attached to a pipe using a thermal coupler. In order to maximize heat transfer from the pipe to the aluminum coupler and Peltier module, the coupler-pipe contact must be of maximum surface area. This implies that the coupler itself must be designed to tightly fit a particular pipe. Any variation in this fit will result in variable coupler efficiencies as noted in Section 3.4. Fortunately, of the pipes surveyed in this study, each had a fairly standard right angle valve connecting the feeder pipe to a flexible hose pipe leading to a faucet. Coupling to this right angle valve requires a custom milled aluminum piece, and deployments on other locations such as shower heads would require a different design. Each DD node includes an aluminum thermal coupler designed to fit around these valves, and the resulting module with coupler, Peltier device, and heatsink is shown in Figure 3.3.

Of equal importance to the performance of DD is the heatsink used to bring the external side of the Peltier junction as close to ambient temperature as possible, and thus increase
The Peltier module includes an aluminum thermal coupler and heatsink. Four arms extend to fasten a heatsink to the Peltier module and coupler. The 0.75”-wide octagonal portion is covered in thermal compound prior to attachment to a right angle valve.

ΔT. DD uses commercially-available air-cooled fin-style heatsinks secured to the external side of the Peltier device with thermal compound.

### 3.3.3 Electrical Design

Each DD node makes use of the state-of-the-art in both processor design and RF communication; the processor used is the MSP430F2410, boasting 300 nA low power mode with clock (LPM3) and 100 nA low power mode with RAM retention but requiring an external interrupt to wakeup. We have designed the node around the Nordic nRF24L01+ radio, the barebone radio used in recent Nordic ANT modules. The nRF24L01+ has a 900 nA deep power down mode and a full power (0 dBm) transmit current of 11.3 mA. In addition to processor and RF, the DD design consists of a power management section and low power wakeup. The interaction of each of these subunits can be seen in Figure 3.4, and is described in detail below.

### 3.3.4 Power Management

Both the scarcity and amplitude of the energy events from which DD harvests dictate that the power regulation circuitry operate from very low start-up voltages and with very little quiescent current. Linear Technology offers a line of step-up DC converters and power management ICs for energy harvesting applications. Of these, the LTC3109 offers the ability to
Figure 3.4: DoubleDip circuitry, including low power wakeup, RF, accelerometer, buddy plug for inter-node energy transfers, temperature sensors, and analog switches for power-gating and duty cycling.

harvest from both voltage polarities (and consequently both temperature polarities, $\pm \Delta T$), and it has an input impedance similar to that found in most Peltier devices ($\sim 2 - 100\Omega$). This IC has one main output for powering a processor or other device as well as a storage output for placing excess energy harvested in a capacitor or battery. Though this is a convenient architecture, the complexity of the design has implications in terms of quiescent current draw. When not harvesting, the LTC3109 will draw 7 $\mu$A from the storage if available and from the output otherwise. To combat this, we use the IC on an as-needed basis, isolating it from the processor and sensing circuit completely when there is no energy to harvest. Figure 3.4 shows how this is implemented—switch $SW1$ opens when the measured $\Delta T$ is below a certain threshold, indicating there is no energy to harvest. Additionally, this reduces the LTC3109 from an application-specific energy harvesting device to a load-matched step-up regulator. That is, the DD architecture is generalizable to any such step-up regulator or power management IC.

3.3.5 Low-Power Wakeup

An ideal water monitoring circuit is one that detects water flow immediately, wakes up to sense that flow, and then returns to sleep in an energy proportional manner. Detecting that a water flow event has occurred is a non-trivial challenge for external sensors, and the
Figure 3.5: Comparator-based low power wakeup trigger. *Hot Trigger* or *Cold Trigger* will activate when there is any small deviation in $\Delta T$, indicating water flow.

solution we have chosen is non-obvious, requiring due explanation.

Detection and trigger circuits can be divided into two classes—those that are actively polled and those that drive asynchronous interrupts. Those belonging to the former include such mechanisms as low power listen (LPL) accelerometers and temperature sensors, requiring a clock to periodically wake up and poll the sensors. In order to have a low latency trigger, we must perform this as frequently as possible. As mentioned in Section 3.1, duty cycling the accelerometer to reduce power will introduce an intolerable latency in the system, and accelerations caused by nearby pipes and other physical activity around the pipe will cause frequent false positives.

If we use a low power analog temperature sensor such as the MCP9700A, we can potentially sample at 1 Hz with a quiescent of $6\mu A$ for only 1–2 ms. This gives a quiescent of $6\mu A \times 2\text{ms} + 0.3\mu A = 0.312\mu A$. At first glance, this seems like a very desirable wakeup method, but such low power temperature sensors come at a price in terms of resolution and accuracy, and thus false negatives will undoubtedly abound. We will see in subsequent sections that even more power hungry temperature sensors (with resolutions of around $0.125^\circ K$ vs. $0.5^\circ K$ for the MCP9700A) often times do not detect events that the wakeup trigger solution settled on in this work does until many seconds after the fact, potentially missing the event altogether. Those wakeup triggers belonging to the second camp—those that actively
drive an interrupt pin—include battery monitoring circuits, temperature alert ICs, comparators, and the like. Of these, the battery monitoring circuits and temperature monitoring circuits are prohibitively power-hungry. Instead, DoubleDip employs low power comparators with finely tuned thresholds to meet its lifetime and accuracy goals.

The DD wakeup method, shown in Figure 3.5, combines a passive sensor and an active IC (ultra-low power comparators) to create an interrupt driving temperature monitor that consumes a mere 600 nA quiescent current. We leverage the fact that the very device from which we are harvesting energy creates small excursions in voltage (less than 10 mV in many cases) in the presence of a differential temperature. By passing this voltage through a low leakage (highly resistive) high pass filter, we can make use of the low power comparator ICs to send a wakeup trigger for both a positive and negative excursion from the baseline temperature. The cutoff frequency here (experimentally chosen to be $f_s = 10\text{s}$) is chosen to allow the baseline to readjust between events, allowing for detection of multiple subsequent events. Because this solution drives an active interrupt pin, we are able to wait in the MSP430’s lowest power setting, LPM4, without requiring power for a clock. This reduces the quiescent of the MSP430 from 300nA for LPM3 to 100nA for LPM4, giving a nominal wakeup current of 700 nA. Measured values indicate that this is closer to around 800nA. This means that, without any energy events to wake up and measure and with no additional harvested energy, DD will diligently wait for an event for 6.4 years given a 3V, 45 mAh battery.

### 3.4 Evaluation

The following section discusses results from hardware implementations of the proposed architecture as well as energy, accuracy, and lifetime statistics from the described deployment.

#### 3.4.1 Energy Harvesting Characterization

In order to better understand the relationship between differences in temperature and energy harvested, we first characterized both the Peltier device and the step-up converter used.
Figure 3.6: Open circuit output voltage for a CUI60333 Peltier module vs. differential temperature across the ceramic plates. The relationship is approximately linear, as per the physics governing the effect. For the heating and cooling phase, $r^2 = 98.9$ and $95.8$, respectively.

To characterize the Peltier device, a known temperature differential was applied across its two ceramic plates and the resulting open-circuit output voltage ($V_{OC}$) was measured. As described by Ferrari et al. in [FFG07], $V_{OC}$ is linearly dependent on $\Delta T$, the difference in Seebeck coefficients of the materials involved, and the thermal conductivity across the ceramic plates. This linear relationship for a $3\times3$ cm Peltier device, the CUI60333, is shown in Figure 3.6.

Similarly, if we apply a known input voltage to the step-up converter such that $V_{in} = V_{OC}$, we can determine the short circuit current output of the power management sector and thus a complete mapping from $\Delta T$ to $I_{SC}$. Figure 3.7 shows this mapping, where temperature differences as little as $\pm1^\circ K$ effect a modest $I_{SC}$. From this, given a known load resistance, we can calculate the expected output power based on $\Delta T$. In the following section, we use this model and the actual energy collected to compute the efficiency of the thermal couplers used.

3.4.2 Fine-Grained Test Data Collection

In order to verify the DD architecture, we designed a separate testing board for fine-grained data collection with identical harvesting and triggering circuitry but with a different proces-
**Figure 3.7:** Short circuit output current of boost regulator vs. input temperature differential, $\Delta T$.

**Figure 3.8:** Testing PCB for 24/7 fine-grained monitoring of temperatures, accelerations, and energy harvested from both hot and cold water pipes.

This board, shown in Figure 3.8, samples pipe temperature, coupler temperature, heatsink temperature, accelerometer variance, output voltage, and system status (showing wakeup triggers, etc.) with one second granularity. The output of the harvesting circuitry on the testing board feeds into a large (15 mF) capacitor from which we can infer both the amount of energy harvested and the amount that would have been lost due to quiescent current draw of the boost regulator, had we not switched it off. If the capacitor on this board goes above a threshold voltage (2.5 V in these experiments), the capacitor is drained and the system status indicates that there was a discharge event. This keeps the capacitor from saturating to prevent it from rejecting any available harvested energy. We deployed this board on 6...
**Figure 3.9:** DoubleDip node—ultra low power energy harvesting node with low latency wakeup, pipe vibration sensing, temperature sensors, and RF. The board size is roughly $3 \times 3$ in.

<table>
<thead>
<tr>
<th></th>
<th>Node Time (hrs)</th>
<th>On Time (hrs)</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Univ. 1 (Hot)</td>
<td>295.6</td>
<td>2.4</td>
<td>389</td>
</tr>
<tr>
<td>Univ. 1 (Cold)</td>
<td>289.2</td>
<td>1.9</td>
<td>215</td>
</tr>
<tr>
<td>Univ. 2 (Hot)</td>
<td>239.2</td>
<td>2.3</td>
<td>249</td>
</tr>
<tr>
<td>Res. 1 (Hot)</td>
<td>99.2</td>
<td>0.5</td>
<td>29</td>
</tr>
<tr>
<td>Res. 2 (Hot)</td>
<td>1127.1</td>
<td>2.7</td>
<td>291</td>
</tr>
<tr>
<td>Res. 3 (Cold)</td>
<td>435</td>
<td>2.8</td>
<td>183</td>
</tr>
</tbody>
</table>

Table 3.1: Total deployment statistics for 6 nodes across 4 buildings, showing the total amount of time each node was installed (Node Time) and active during an event (On Time) as well as the total number of events sensed.

Distinct pipes feeding into 5 different sinks in 4 buildings (3 residential and 1 at a university). Of these pipes, 4 were hot water intake pipes to sinks, and 2 were cold water intakes. The statistics of this deployment are summarized in Table 3.1. Data was collected over a total of 103.5 node-days over which a total of 1,356 water flow events and 12.6 hours of induced vibrations were recorded.

Figure 3.10 shows data from a test board deployed for an example weekday during the month of March. The top plot shows temperature curves for the water pipe as well as the difference in temperature between the pipe and heatsink while the second and third plots show output voltage and cumulative energy harvested, respectively. Energy harvested during this day totaled around 1000 mJ, while energy expended is equal to energy spent waiting in
Figure 3.10: Power harvested for an example weekday during March (24 hours starting at 12:00am). From top to bottom, plots show (1) Pipe temperature and $\Delta T = T_{\text{coupler}} - T_{\text{heatsink}}$; (2) output voltage across a 15 mF capacitor with discharge signals when the capacitor gets close to saturation; and (3) cumulative energy harvested over the entire day, in mJ. Cumulative energy is shown when the step-up DC-DC converter is always on and when it is switched on an as-needed basis.

This number will be explained in detail in the following sections.

3.4.3 Wakeup Power, Latency, and Accuracy

The low power wakeup mode draws quiescent current from two comparators operating at a nominal 300 nA and the MSP430F2410 operating at a nominal 100 nA, making the nominal current 700 nA. As Figure 3.11 shows, this current is closer to around 800 nA, with an average of 820 nA. Figure 3.11 also shows that if the pins are configured to communicate with the radio, the average current goes up to around 900 nA, and if we were to use the MSP430’s LPM3 (with low power oscillator), the average current would be closer to 1.2 $\mu$A. The comparators used (the LTC1540) have a high offset voltage, meaning that for best performance a comparator’s reference voltage (also generated by the LTC1540) should be tuned to just above the triggering threshold. Even with all comparators tuned, DD nodes
**Figure 3.11:** Current consumption for low power wakeup circuitry and deep sleep MSP430 (LPM4). Current is shown with and without SPI pins configured for RF and compared to a sleep mode with low power clock (LPM3).

**Figure 3.12:** Example calculation of wakeup latency. The green circle indicates the start of the water event, as determined by the vibration variance (y-axis). The red square indicates the time the wakeup trigger fired. The difference between these two is the latency.

In different physical locations and buildings are likely to see a range of wakeup times, due in large part to the pipe’s distance from the main water heater, the ambient temperature of the building, the fitting of the coupler, and even the frequency with which water flows through that pipe—these factors and others contribute to the rate of heat transfer from pipe to TEG. Figure 3.12 explains the procedure for estimating the wakeup latency from the vibration data and the triggers from the wakeup cycle. Figure 3.13 shows a CDF of the wakeup latencies across all nodes, and Table 3.2 shows the number of false positives and the average amount of water flow time missed per day. There occur many more false negatives than false positives (none of which we observed), but by relating the false negatives
Figure 3.13: Cumulative distribution function (CDF) for wakeup latencies across all nodes.

Table 3.2: Wakeup latencies and accuracies per day, per node.

<table>
<thead>
<tr>
<th>Location</th>
<th>%Extra Events</th>
<th>%Missed Events</th>
<th>%Missed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Univ. 1 (Hot)</td>
<td>0.00</td>
<td>3.62</td>
<td>1.54</td>
</tr>
<tr>
<td>Univ. 1 (Cold)</td>
<td>0.00</td>
<td>2.40</td>
<td>0.73</td>
</tr>
<tr>
<td>Univ. 2 (Hot)</td>
<td>0.00</td>
<td>1.61</td>
<td>1.79</td>
</tr>
<tr>
<td>Res. 1 (Hot)</td>
<td>0.00</td>
<td>13.79</td>
<td>20.31</td>
</tr>
<tr>
<td>Res. 2 (Hot)</td>
<td>0.00</td>
<td>34.60</td>
<td>19.32</td>
</tr>
<tr>
<td>Res. 3 (Cold)</td>
<td>0.00</td>
<td>15.38</td>
<td>17.59</td>
</tr>
</tbody>
</table>

to the corresponding length of the water event missed, we see that the actual percentage of water flow left unobserved is fairly small (< 2%) in a university setting and moderate in residential settings. Where the pipes are farther from the main boiler or the water pressure is lower (such as residential housing) the latency observed and % water flow time unobserved is considerably higher. However, this can be ameliorated to some extent through careful tuning. For example, the excellent latency for all University deployments is in part due to better tuning given their ease of access. Residential deployments, on the other hand, received no further tuning after the initial deployment. Figure 3.14 shows how drastically wakeup latency can be improved by fine-tuning the comparator threshold. Furthermore, this tuning can be completely automated. We have created a prototype autotuning system using a digital potentiometer, yielding much finer resolution tuning and thus lower latency wakeup. This comes at only a marginal cost in energy—about 50 nA additional—and will thus be incorporated into future versions.
Figure 3.14: CDF of wakeup latency for a single node before and after tuning the wakeup threshold.

3.4.4 Subsystem Power Characterization

Since the quiescent current of DD is relatively minuscule and the voltage versus capacity curve of the battery is fairly flat, we see little or no change in battery voltage as energy is consumed or returned to the battery. To get a good estimate of the energy harvested, therefore, the test board shown in Figure 3.8 is used. In order to determine the overall system lifetime and the effect that harvesting has on the lifetime, however, we must perform a careful analysis of the DD node itself, shown in Figure 3.9. Specifically, we characterized the operation of the Nordic nRF24L01+ 2.4 GHz radio while operating from a capacitor for transient loads and the power consumption of the MSP430F2410 when transitioning from sleep to active mode and sampling an accelerometer at 100 Hz. By doing so, we can calculate the amount of energy spent per day by knowing the amount of time spent sampling events and transmitting RF. The total amount of energy consumed per day is then given by:

$$E_{day} = I_{idle} * t_{idle} + I_{adxl} * t_{on} + I_{RF-Boot} * N + I_{TX}(t_{on} - N) \quad (3.2)$$

Where $N$ is the number of events detected in that day, $t_{on}$ is the total amount of system on-time during that day, $I_{RF-Boot}$ is the average cold boot wakeup current of the nRF24L01+ radio after power-gating the radio, and $I_{TX}$ refers to the average transmission cost of the radio when one packet is sent every second.
Figure 3.15: Current consumption of an ADXL335 3-axis analog accelerometer, sampling 32 bytes every second at 100 Hz. Current for a constantly on ADXL335 vs. one that is power-gated (‘Duty Cycled’) are shown, offset in phase for clarity. Power-gating reduces average current consumption from 296 µA to 120 µA.

Figure 3.15 shows the instantaneous and average currents consumed by both the accelerometer and the MSP430 while sampling the accelerometer for 32 samples. If we naively sample the accelerometer, an ADXL335, by power-gating only at the very beginning and end of the sampling session, the system consumes 296 µA on average. If however we power-gate the accelerometer in between samples (something that is only possible if the decoupling capacitance is kept below 0.1 µF), we obtain an average power of 120 µA.

Figure 3.16 shows the operation of the radio from a transient load capacitor. A modest reduction in capacitor voltage occurs during each transmission, and the capacitor must be recharged in a periodic fashion to send multiple packets or retransmissions. The current consumed by the radio as measured by an ammeter and verified from Figure 3.16 is 4.96 µA with σ = 0.99µA for power-gating and 3.43 µA with σ = 0.37µA for power-down (RAM retention) for one 32 Byte packet per second. By moving away from LPL radio reception and heavily duty cycling stackless radios, we can greatly reduce the consumption of what was once and in some systems remains the power bottleneck [DBV11, HPF10].

3.4.5 Net Energy Profile

Combining these numbers as per Equation 3.2 and comparing with the total amount of energy harvested per day given the test boards yields the net harvested per day, per node. Figure 3.17 shows these numbers plotted per day-of-the-week (left) and the net energy per
Figure 3.16: Current consumption of a Nordic nRF24L01+ radio operating from a capacitor. The radio sends one 32 Byte packet every second and then is turned off (power-gated). The right-most plot shows a zoomed in version of what occurs to the voltage across the transient capacitor during a single transmission.

day taken by subtracting energy consumed from energy harvested (right) along with the average net energy per day. As expected, the most energy gained per day is realized by the two hot water intake pipes in the university restroom. Additionally, these two numbers are heavily influenced by both the day of the week (weekend vs. weekday) and the season – the weekdays leading up to the second week are during a university break. Residential 1 (a single residence), for which only 6 days of data were collected, is also marginally in positive numbers, despite a low number of total events. This is due to the energy proportionality of DD, scaling down to just 820 nA when no activities are present. University 1 Cold and Residential 2 and 3, however, are in negative numbers, indicating that their lifetime is limited by more than just the shelf life and cycle count of the battery chosen. The DD system lifetime can then be estimated by:

\[
\begin{align*}
    t_{life} &= \begin{cases} 
        t_{bat-shelf-life} & \text{if } E_{avg} \geq 0 \\
        \frac{A \cdot h(bat) \times V_{cc} \times 3600}{|E_{avg}| \times 365.25} & \text{if } E_{avg} < 0
    \end{cases}
\end{align*}
\]

(3.3)

where the capacity of the rechargeable lithium battery used is 45mAh, and \( V_{cc} = 3V \) during normal operation. The energy burned per day for a node where no events occur and
no energy is harvested is $228 mJ$ and the total lifetime is 6.4 years. On the other hand, the worst case $E_{avg}$ per day from Figure 3.17 is $-128 mJ$ / day, belonging to Residential 3 cold. This has a total lifetime of $45 mAh \times 3 \times 3600 / 128 mJ \times 365.25 = 10.4$ years, almost 10 years shy of the shelf life of the ML2020 battery used. This energy can be made up for if we request energy from a nearby node in order to meet the full shelf life specification of the battery, though in some cases nodes with energy deficits may be isolated.

Finally, it is also particularly interesting to note unexpected trends (or even lack thereof) in terms of power harvested by week-day. There are many confounding variables responsible

**Figure 3.17:** Energy harvested and consumed per day for 2 example weeks. By subtracting energy consumed from energy harvested (left) we get the net energy per day (right). Mean daily energy and lifetime estimates are calculated using the entire data set for each node (not shown). University 1 (Hot), University 2 (Hot), and Residential 1 (Hot) show a net positive in energy gained, indicating that the node lifetime is limited only by the battery shelf life. The others (while suffering a net loss in energy) are still considerably better than they would have been had the same circuitry been used without harvesting.
for some of the change in energy harvested during the first week (vacation) and over the weekends, not the least of which is the status of HVAC in the building. For example, if air conditioning is turned off to save power, the ambient temperature of the cold water close to the valve might rise while the water in walled-in pipes remains cool, causing an increased $\Delta T$ when water begins to flow.

### 3.4.6 Efficiency

There are many conversions from the path of temperature to usable energy, and it is worth noting the various inefficiencies at each junction and the effect each has on the system as a whole. Perhaps the most important of these is the pipe→coupler→heatsink junction. The inefficiencies here are very observable; an ideal heatsink would remain at ambient temperature while an ideal coupler would track the pipe temperature with zero error. Instead, the heatsink converges to within some $\Delta T$ of the coupler given time and the coupler resembles a low-pass filtered version of the pipe temperature. Figure 3.18 shows an example of the temperature curves observed during two bursts of hot water on University 1 Hot.

By combining the temperature measurements with the model derived in Figure 3.7, we can calculate the pipe→coupler efficiency as the ratio between power delivered to the TEG and ideal power given the pipe and ambient temperatures. These values are given in Table 3.3, ranging from 65% all the way down to 18%. The variation can be explained in part...
<table>
<thead>
<tr>
<th>Location</th>
<th>Coupler Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>University 1 (Hot)</td>
<td>65.82</td>
</tr>
<tr>
<td>University 1 (Cold)</td>
<td>19.55</td>
</tr>
<tr>
<td>University 2 (Hot)</td>
<td>41.59</td>
</tr>
<tr>
<td>Residential 1 (Hot)</td>
<td>27.09</td>
</tr>
<tr>
<td>Residential 2 (Hot)</td>
<td>17.96</td>
</tr>
<tr>
<td>Residential 3 (Cold)</td>
<td>21.65</td>
</tr>
</tbody>
</table>

**Table 3.3:** Coupler efficiencies

![Temperature change rate for varying contact areas.](image)

**Figure 3.19:** Temperature change rate for varying contact areas.

by the installations—each installation requires fastening the coupler to the pipe, and some installations were more successful than others due to manufacture variations in the right angle valves and couplers as well as variations in how tightly each coupler was fastened to the right angle valve. Improving this efficiency can be accomplished by increasing the surface area in contact with the coupler, decreasing the mass and radiant surface area of the coupler, and improving the fastening process. The calculation of the efficiency of converting the thermal energy to electrical energy, on the other hand, is quite complex and contains many opposing forces. In order to capture more heat from the pipe, more surface area must be in contact with the coupler as stated above. Yet in increasing the surface area in contact one necessarily increases the mass of the coupler itself, increasing the rise time of the coupler and decreasing the responsiveness of the wakeup triggers. Measures can be taken to decrease the thermal mass of the coupler while retaining a larger surface area in contact with the pipe, but in doing so care should be taken to not increase the radiant surface area of the coupler, thereby creating a virtual heatsink on the side on you wish to retain heat. **Figure 3.19** shows
how the current thermal coupler responds to changes in contact area. Decreasing contact area clearly decreases the rate at which heat is transferred to the TEG, which will decrease energy harvested and increase wakeup latency.

The conversion of the small electrical signal ($V_{OC}$) produced by the Peltier (and indeed the Peltier junction itself) to a usable voltage is another significant source of inefficiency. The specified efficiency of the step-up converter used (and as validated by tests described earlier) is at most 25% and perhaps better estimated at around 15% given the range of input voltages seen. This efficiency increases dramatically if we can afford to increase the operational start-up voltage. That is, if we can increase $V_{OC}$ by means described above, we can not only harvest more but lose less in the boost conversion process. Finally, there is a loss in the diode used to drop the output voltage of the step-up converter from 3.3 V to 3.1 V—the nominal charging voltage of the lithium battery—and in the charging / discharging routine of the battery itself. The efficiency of the diode can be calculated as $100\% \times \frac{3.1V}{3.3V} \approx 94\%$, and despite lengthy tests of charge & discharge cycles at varying currents, no appreciable inefficiency in the battery could be detected. Given these inefficiencies, the potential to harvest from energy embedded in water pipes is clearly much greater than we are able to achieve with our current prototype, and improvements in technologies and methods mentioned throughout this chapter will likely allow for a host of similar applications.

### 3.5 Discussion and Future Work

Despite the success of DD in meeting the shelf life limits of the battery used for those nodes with a net positive energy per day and those in close proximity and able to share surplus energy reserves, there remain interesting challenges to both increasing the efficiency and quality of DD and applying these lessons elsewhere. Potential future work currently under consideration includes:

1. A finite element analysis of heat transfer and dissipation via the thermal coupler.

2. A more advanced two-way communication protocol between proximal harvesting nodes,
allowing nearby nodes access to battery state of all neighboring nodes and ways to more elegantly meet lifetime specs.

3. A continued deployment across more residencies to better understand the variations of wakeup latencies, temperatures, and power numbers from location to location.

4. Developing a hybrid capacitor system to amortize the effect of battery shelf-life on overall system lifetime.

We have demonstrated an architecture—DoubleDip—capable of sensing pipe vibrations and enabling non-intrusive water flow detection for, in some cases, the entire lifetime of the lithium battery used (20+ years). We accomplished this by leveraging the energy embedded in the differential temperature between the pipe (hot or cold) and the ambient room temperature for both compensating energy expenditures and triggering the system to awaken from a deep sleep mode. DD espouses an extremely low power architecture, wherein the sacrifice of mesh networking, time synchronization, and other luxuries is a necessity given the anemia and irregularity of the energy (water) events.

Our pilot deployment of 6 nodes across 5 sinks and 4 buildings showed that our low power wakeup trigger had a reasonable latency (around 4 seconds after calibration) and a high accuracy of detection, with virtually no false positives and a small fraction of water flow time missed with respect to the total time of water flow per day. We observed that some pipes experience much greater changes in temperature and thus were able to harvest much more than others, even given physical proximity. To mitigate this, we developed a method by which a node with excess energy reserves can wake up a nearby node and transfer a fraction of its harvested energy. Net energy gain from day to day varied from an average of +316 mJ/day on a university hot water pipe feeding a sink faucet to -128 mJ/day on one residential cold water pipe, meaning the lifetime of some nodes was limited only by the battery shelf life, and even those nodes with net losses in energy saw an improvement through harvesting and the low power DD architecture.
CHAPTER 4

Low Power Sensing Hardware for Sparse Signals

4.1 Signal Acquisition and Sparsity

Chapters 2 and 3 demonstrated that by reducing active computation time when sensing temporally sparse signals we can save energy without greatly reducing quality of service in the process. In many cases, however, the information describing the phenomenon being sensed during active time frames exhibits sparsity in another dimension and can benefit from additional compression. In this chapter we describe a hardware approach to exploiting general sparsity in signals, performing energy-saving compression in the analog domain before digital quantization.

Many physical phenomena follow specific patterns and models that can be exploited to parametrize and compress the sensed data. This compression step, however, is traditionally performed in the digital domain only after the data has already been sampled at the Nyquist rate. Nyquist rate sampling is rather wasteful for compressible signals since their information content may be vastly lower than that assumed by the Nyquist criterion. Furthermore, it is not always practical to perform compression at low power sensing sites and the data might need to be transported uncompressed (also wasteful!) to a high powered collection center.

Compressed sensing is a recent breakthrough in signal processing that allows one to acquire a compressible signal at much below its Nyquist rate [CRT06b]. While the Shannon-Nyquist theorem provides sufficient conditions for recovering a periodically sampled signal based on its bandwidth, it is unable to include knowledge of other signal characteristics. Compressed sensing (CS) is a mathematical tool that can utilize a priori knowledge of the sparseness or compressibility of a signal of interest within a model framework to acquire a
signal at essentially its “information rate”. The basic tenet of CS is that if a signal is known to be sparse (contains few non-zero values) or compressible (values decay quickly to zero) in a known domain, it is wasteful to sample the signal at the Nyquist rate because most of the data will be thrown away subsequent to compression.

There are three aspects to implementing CS – knowing the domain in which the signal is sparse, low dimensional signal acquisition and the recovery process. The sparse domain is a transform space, typically a set of linear functions, that facilitates a compressed representation of the input signal. That is, when the signal is transformed from its native domain (say, time or space) to the sparse domain (say, frequency or wavelets), a few values in the sparse domain may be sufficient to describe the signal with high fidelity. This compaction property is the cornerstone of compressed sensing.

Signal acquisition in CS involves projecting the signal to a lower dimensional domain that is “incoherent” to the sparse domain before sampling [CR07]. This incoherent projection maximizes the information collected in each sample with respect to the sparse domain and the lower dimensionality yields compression. The projection process generates a set of compressed measurements and can be accomplished at relatively low complexity, making CS an attractive alternative for energy efficient sensing. This chapter describes a low power hardware implementation of the projection process.

The recovery process exploits the fact that there are few information bearing components in the sparse domain in order to identify them. Generally speaking, it looks for the most compact (sparsest) solution that meets the constraints set by the compressed measurements. The quality of reconstruction depends approximately on the ratio of the number of compressed measurements acquired to the number of information bearing (non-zero) components. The catch, however, is that the recovery process is computationally intensive. A number of recovery algorithms exist that each trade off computation for accuracy differently [CWB08, TG07, CDS98].

CS could be viewed as an asymmetric compression scheme with economical encoding but expensive recovery. This makes compressed sensing particularly well suited for low
power physical sensing, where sensor devices are highly constrained, both in energy and computational resources. It is expected that the compressed domain samples would be delivered to a capable backend for signal recovery or inferencing.

**Sampling Compressively**

Returning to signal acquisition, the key intuition behind CS’s incoherent sampling strategy is that of spreading information content in the signal vector (with respect to the sparse domain) across the compressed domain samples. In some sense, one could view each compressed sample as providing an independent summary of the input signal. Researchers have identified that domains constructed from certain random distributions have high incoherence with virtually any sparsity inducing basis with high probability. This is termed the universality property of compressed sensing [CT06]. Fig. 4.1 shows a visual representation of a conceptual CS acquisition. In this case, we assume that the input signal, \( x \), shown rightmost, is sparse in the time domain with an exaggerated sparsity of just one non-zero component. The sampling matrix, \( \Phi \), is generated from a random \(+1/−1\) (black/white) Bernoulli distribution, i.e., we toss an unbiased coin for each element of the matrix. The measurement vector, \( y \), smaller in dimensionality than \( x \), is computed through \( y = \Phi x \) and constitutes the compressed domain samples we seek.

The effect of using a random kernel is that regardless of where the information bearing components lie in the sparse domain, there is a statistical chance that the sampling matrix captures it. Observe, for example in Fig. 4.1, how one non-zero value in \( x \) manifests itself

---

**Figure 4.1:** Visual representation of compressed sensing through pseudo-random projections from a \(+1/−1\) (black/white) Bernoulli distribution. The measurement vector \( y \in \mathbb{R}^m \) is computed by projecting the sparse input vector \( x \in \mathbb{R}^n \) using a sensing matrix \( \Phi \in \mathbb{R}^{m \times n} \) by \( y = \Phi x \).
Figure 4.2: Software implementation of Compressed Sensing needs pre-digitized data and $O(mn)$ operations.

in each of the compressed measurements in $y$. It must be mentioned that while randomized projections are not the most efficient\(^1\), the flexibility afforded by the universality property is overwhelmingly attractive.

Although CS holds promise for many applications, the need for incoherent projections has limited its direct impact to instances where the sampling domain is inherently incoherent to the signal of interest. For example, this holds for magnetic resonance imaging (MRI), where sampling is performed in the frequency domain while the signal being recovered is an image in the spatial domain [LDP07]. In other sensing applications such as EEG or ECG monitoring, where sampling is traditionally performed in the time domain, the incoherent projections have to be applied explicitly [KMK11]. Fig. 4.2 depicts a software approach to CS. The inner loop performs $m$ row-wise multiply-and-accumulate (blue downward arrow in Fig. 4.1) and the outer loop executes across columns for every sample acquired (red rightward arrow in Fig. 4.1). The output $y$ is read every $n$ samples and the accumulators are then reset.

There are three practical issues with this software based CS technique. First, it requires that the signal be explicitly sampled before projections can be computed. Since CS was developed as a solution to avoid unnecessary sampling, while viable in some instances [KMK11], this technique does not fully exploit the advantages that compressed sensing offers. Second, an order of $O(mn)$ explicit mathematical operations are typically required to compute the projection. The computational cost of explicit compression for some transform

\(^1\)Ideally, the rows of the measurement matrix should be orthogonal. Randomly generated matrices only have weak orthogonality guarantees [LKR12].
domains may actually be lower ($O(n \log n)$ for FFT), although software CS operations are simpler (add/subtract). Third, since samples are quantized prior to the projection, quantization error accumulates as $n$ increases. That is, as the compression ratio ($n/m$) increases, so does the effect of quantization [SBB06].

This chapter describes a novel hardware approach to compressive sampling. Our system, called CapMux, combines digital logic with a custom analog front end to realize randomized projections at very low power. CapMux takes as input a time domain signal of variable duration (i.e. variable $n$) and produces a fixed number of compressed domain analog measurements (i.e. fixed $m$) over that duration. These compressed measurements can be fed directly to an analog-to-digital converter to be digitized, transported and processed for signal recovery or inference. The key idea that makes our architecture low power is being able to amortize the quiescent current consumption costs of high performance analog components through time-multiplexing. CapMux is constructed from just one active analog signal processing chain that can be shared across an arbitrary number of channels. CapMux not only leads to a lower average power per measurement channel, but also admits low complexity scaling.

4.2 Compressed Sensing and Related Work

We first briefly describe the usual compressed sensing procedure: Assume that the signal of interest $x \in \mathbb{R}^n$ and a set of measurements $y \in \mathbb{R}^m$, $m \ll n$ are available to us, such that $y = \Phi x$, where $\Phi \in \mathbb{R}^{m \times n}$ is a sensing matrix. Then, under the condition that $x$ is sufficiently sparse, the solution to the following combinatorial optimization problem recovers the signal exactly:

$$\hat{x} = \arg\min_x ||x||_0 \quad \text{s.t.} \quad y = \Phi \hat{x}$$

(4.1)

where $||x||_0 \triangleq |\{i : x_i \neq 0\}|$, the number of non-zero elements in $x$. Finding a solution to Eq. (4.1) is NP-complete in general. Instead, a convex relaxation is proposed [CRT06a]:

$$\hat{x} = \arg\min_x ||x||_1 \quad \text{s.t.} \quad y = \Phi \hat{x}$$

(4.2)
where \( \|x\|_{\ell_1} \triangleq \sum_{i=1}^{n} |x_i| \). It is shown in [CRT06a] that under the sparsity condition and when \( \Phi \) satisfies the so-called Restricted Isometry Property (RIP), the reconstruction \( \hat{x} \) is exact with overwhelming probability [CRT06a]. Practically, this means that if the signal is sparse in the sensing domain, then taking \( m \) measurements through a suitable linear transformation \( \Phi \) will be sufficient to reconstruct the signal. If the signal is not sparse in the sensing domain, but in another known domain, the reconstruction must be performed in two steps. Assume a separate invertible linear transformation \( \Psi \) under which the signal is rendered sparse, \( z = \Psi x \) where \( \Psi \in \mathbb{C}^{n \times n} \). For example, if the signal was a single frequency tone, then its time domain representation \( z \) is not sparse, but with \( \Psi \) as the Inverse Fourier transform, \( x \) is sparse. The equivalent reconstruction procedure is then:

\[
\hat{x} = \arg\min_{\tilde{x}} \|\tilde{x}\|_{\ell_1} \quad \text{s.t.} \quad y = \Phi \Psi \tilde{x}
\] (4.3)

An additional requirement for reconstruction to succeed is that \( \Psi \) be incoherent [DH01] with the projection matrix \( \Phi \) or equivalently that the combination \( \Phi \Psi \) satisfies the RIP. This holds with high probability, for example, when the elements of \( \Phi \) are independent realizations of a Gaussian random variable, \( \mathcal{N}(0, \frac{1}{n}) \) or of an equiprobable \( \pm \frac{1}{\sqrt{n}} \) Bernoulli random variable, regardless of \( \Psi \) (w.h.p.) [CRT06a].

Implementing randomized projections \( y = \Phi x \) has been the subject of much research and in some domains, such as MRI, happens naturally through non-uniform sampling [LDP07]. In other applications, where the sensing domain is temporal or spatial, projections have to be performed explicitly. Fig. 4.2 depicts the software approach taken by numerous prototype implementations [BG09, KMK11, CCS12] when sampling rates and power permit. As explained in Sec. 4.1, software CS does not provide ADC sampling rate reduction and its consequent savings in power.

Direct hardware implementations do exist as well. Some of the earliest demonstrations of CS were developed by Duarte, et. al. [DDT08] for compressive imaging using a single pixel. The randomized projections were accomplished using a digital micro-mirror array that was programmed with the random vectors. While it has limited use in natural photography,
due mainly to aperture time requirements, certain non-visible light applications are quite promising [CMB08].

For time domain signals, Kirolos and Laska, et. al. [KLW06, LKD07, TLD10] introduced the concept of analog-to-information conversion that constitutes randomized demodulation (RD). This system entails multiplying the input signal with a random bit stream of ±1s at a rate higher than the Nyquist rate of the input signal followed by a low pass filter of known impulse response. The compressed measurements are acquired by sampling the output of the low pass filter at regular intervals. While an elegant architecture, RD suffers from an important drawback. As Yu, et. al. [YHS08] point out, because the measurements are obtained by sampling the output of the analog filter sequentially, they are no longer independent due to the convolution in the filter.

Yu suggests a parallel structure that applies RD to the signal simultaneously across a number of branches that each use a different random bit stream for the multiplication. The advantage is that each branch now produces independent compressed measurements but at the cost of immense hardware complexity. Mishali, et. al. [ME10] follow a similar architecture in their modulated wideband converter, except that they reduce the required number of independent measurement channels by assuming that the signal is structured in such a way that it only occupies some discrete bands in the frequency spectrum. A simplified representation of the parallel RD implementation is shown in Fig. 4.3. Here, $f_c$ is the “chipping” rate for the random demodulation and $f_s$ is the sub-Nyquist sampling rate.
per channel. The discrete-time equivalent of the sampling matrix specified over a window of time $T$ is of dimensionality $mf_sT \times f_cT$ with each row corresponding to one bit stream.

An interesting recent paper by Slavinsky, et. al. [SLD11] describes a novel architecture called CSMux that captures simultaneously multiple signals that are jointly sparse. The signals are multiplied each with an independent random bit stream and then summed before sampling at the Nyquist rate of one of the signals. In some ways, this is equivalent to the modulated wideband converter, except that CSMux requires that the signals be split, through band pass filtering, into individual frequency bands prior to processing. In all the above approaches, the requirement of multiple analog processing branches is evident even if fairly restrictive assumptions are made about the sparsity of the signal and the domain it is sparse in – Mishali’s prototype consists of 4 independent branches. A more recent fully integrated implementation used 8 branches [YBM12]. Scaling these systems to a large number of measurement channels while maintaining CS’s universality property becomes impractical.

Furthermore, these architectures (with the exception of CSMux) require active analog components in each branch that each consume quiescent current. The multiplier can be handled passively [SLD11] but the low pass filters are constructed from high performance, low noise opamps. As the number of channels is scaled to increase the number of independent measurements, the quiescent current increases proportionally. For low power sensing applications, aggregate quiescent current can easily exceed the active power of the ADC, negating the benefits accrued through compressed sensing. The following section describes our CapMux architecture, which uses a single analog processing chain and scales to an arbitrary number of independent channels (of reduced bandwidth) with minimal hardware complexity.

4.3 The CapMux Compressed Sampler

The CapMux architecture is illustrated in Fig. 4.4. CapMux uses time-multiplexing to amortize the quiescent current costs of one high performance integration opamp across the
Figure 4.4: Hardware implementation of Compressed Sensing in CapMux using a single time multiplexed signal processing chain.

$m$ measurement channels. Observe that the signal is multiplied with one random bitstream generated at $mf_c$ instead of the $m$ bitstreams at $f_c$ in the parallel RD architecture shown in Fig. 4.3. The ADC sampling the output of CapMux generates measurements at the rate of $mf_s$, which is equal to the combined outputs of the branches in parallel RD.

The CapMux system hinges on the operation of a multi-channel integrator. For simplicity, assume for now that the multi-channel integrator consists of a bank of $m$ independent integrator blocks, only one of which is active at a time. The role of each hypothetical integrator is equivalent to that of the low pass filter in RD, in that it sums the values output by the random multiplier. The channel synchronization block selects one integrator from this virtual bank in round robin fashion at the same rate as random bitstream generation. That is, each integration only lasts for one random multiplication, for a time slot $\tau = 1/mf_c$ in this case) long and each integrator channel only sees $1/m$-th of the entire bitstream.

One integration round over the $m$ channels is accomplished in $m\tau = 1/f_c$ seconds and if the input signal does not change over this time period (this can be explicitly ensured or simply assumed for low bandwidth signals), the set of $m$ integrations approximates compressed sensing projections over one column of $\Phi$ as shown by the blue downward arrow in Fig. 4.1 or one inner loop iteration in the software CS approach shown in Fig. 4.2. If this process is continued for $T = n/f_c$ seconds, the values accumulated in each hypothetical integrator constitute the $m$ compressed measurements corresponding to $y = \Phi x$. After every $T$ seconds, the integrator channels can be sampled by a low rate ADC and reset to start a new projection.
block. Mathematically, each integration operation can be written as:

$$y_{ji} = \int_{\frac{1}{Ic} + \frac{j}{m/Ic} + \tau}^{\frac{1}{Ic} + \frac{j}{m/Ic}} \Phi_{ji}x(t)dt \quad (4.4)$$

for row $j \in 1...m$ and column $i \in 1...n$ of sensing matrix $\Phi$. For a constant input signal within the integration slot, this is simply $y_{ji} = \Phi_{ji}x_i \tau$ and across $n$ integration rounds output of channel $j$ would be:

$$y_j = \sum_{i=1}^{n} \Phi_{ji}x_i \tau = \tau \Phi_jx \quad (4.5)$$

the $j$-th row of the CS projection matrix with scaling $\tau$.

4.3.1 The Multi-Integrator

The integrator is specially constructed to emulate integration of $m$ channels independently. The design of the multi-integrator is shown in Fig. 4.5. An active integrator is essentially an opamp in the voltage-to-current conversion mode that provides a charging current proportional to the input voltage to a capacitor in the negative feedback path, resulting in signal integration. The output of the integrator is given by $v_j = v_j(0) - \tau/(RC) \cdot x_i$ for channel $j$ and time slot $i$ of the input signal with an initial value $v_j(0) = Q_j(0)/C$, where $Q_j(0)$ is the charge stored in the capacitor of capacitance $C$ before integration.

Bruton and Pederson [BP72] show how one opamp can be time-multiplexed to implement
higher order filters. The multi-integrator uses this idea to share an opamp across \( m \) channels of integration. Since the state of the integrator is stored in the charge on the capacitor, a multi-channel integrator can be formed with a parallel bank of capacitors only one of which is switched in at any time. The channel capacitor switched in is the one actively integrating and by synchronizing capacitor switching to the random multiplication, compressed projections can be achieved. Note that this system is not only low power, because it expends little quiescent current, but is also low energy, because despite the time-multiplexing operation, the duration over which the signal is processed to produce \( m \) compressed measurements is the same as that of the parallel RD design. In contrast to parallel RD, scaling this design to a desired number of measurement channels only entails adding to the switched capacitor bank and modifying the random bitstream generator accordingly. As switches and capacitors occupy little circuit area and power, CapMux can be scaled arbitrarily (but with reduced bandwidth, as described next).

### 4.3.2 Practical Considerations

The performance of CapMux at a given current consumption is determined by the chipping rate, \( f_c \), which must be higher than the Nyquist rate of the input signal. Chipping rate in CapMux depends on the total integration time for one round (one column of the sensing matrix). Channel count and per channel integration time \( \tau \) should, therefore, be kept as small as possible, but the latter is governed by the speed (slew rate) of the opamp, the leakage characteristics of the switches and capacitors, and the overall noise of the circuit. A low \( \tau \) would also mean a reduced integrated charge and a lower signal-to-noise ratio (SNR), unless the capacitors are small too. But, one must ensure that total switch parasitic capacitance (\( \sim 2-5 \text{pF per switch} \)) should be a tiny fraction of integration capacitance. At a given \( \tau \), the maximum allowable signal bandwidth is given by \( 1/(2m\tau) \) for Bernoulli sensing matrices.
Figure 4.6: Empirical recovery performance with $16 \times 64$ sparse binary sampling matrices of varying density compared to traditional Bernoulli sampling matrices.

4.3.3 Sparse Binary Matrices

It has recently been observed that sparse binary matrices [LKR12] also admit recovery guarantees similar to that of Bernoulli matrices while maintaining the universality property. Two properties make sparse binary matrices (SBM) especially attractive. First, SBMs consist of 0/1s instead of ±1s. Second, the number of 1s in each column is a small fixed number called the degree or density of the matrix with $d \ll m$, especially when $m$ is large. The first property implies that an implementation no longer requires signal inversion (multiplication is realized by switching a pass through or inverted signal into the integrator [SLD11]) and the second property implies that only as many capacitors need to be switched in each integration round as the density. Since the density can be much lower than the number of channels, the total integration time is now $d\tau$ instead of $m\tau$ and the maximum allowable bandwidth is now given by $1/(2d\tau)$. Fig. 4.6 shows the empirical recovery performance with SBM matrices of signals sparse in the time domain (the worst case scenario for SBM). For $16 \times 64$ matrices, $d = 8$ performs as well as Bernoulli matrices, which implies an instantaneous doubling of chipping rate and allowable signal bandwidth. It can also be observed that density beyond half the number of channels deteriorates recovery performance due to overlapping of information-bearing elements in $y$. 
4.3.4 Calibrating For Component Variability

Variability and non-ideal behaviour of components causes the output of CapMux to stray from expectations (see top plot in Fig. 4.7). While error can be reduced to a large extent by using high-cost high-precision components, it can not be completely eliminated. Furthermore, better performance in opamps usually comes at the cost of higher quiescent current resulting in an increased energy footprint.

Two factors affect CapMux performance significantly that require to be accounted for through calibration routines: capacitor variations and integrator offset. Integrator offset is caused by opamp input offset voltage that causes small charging currents even when the input is zero. Capacitor variations alter the transfer characteristics of the channels, manifesting as unequal integrator gains and offsets for each. If we express the output of the integrator as a function of these component variations and the input amplitude, $x$, we have

$$V_{out} = V_{os-m} + \beta_m Gx$$

(4.6)

where $V_{os-m}$ is the channel-dependent offset voltage at the output of the integrator, $G$ is the nominal channel gain, and $\beta_m$ is the per-channel gain scalar. By feeding a set of known input amplitudes and measuring the output, we apply linear regression per channel to yield the individual $\beta_m$ scalars and $V_{os-m}$ offsets. The effect that this calibration has on the output of the integrator is illustrated by comparing the error before and after (Fig. 4.7). Note the change in y-axis between the top and bottom plots.

If we take samples across time during the integration of one channel across 64 samples, we see the benefit of having calibrated the system to non-idealities. Fig. 4.8 shows the ideal and measured values of the integrator for the compression of a 3-sparse signal using a sparse binary matrix of density $d = 8$. Despite noise, leakage, and component variation, the integrated signal remains within 5% accuracy of the ideal across all 64 samples.
\textbf{Figure 4.7}: Error in mV between measured and expected values before and after calibration. Different colors indicate values from different channels. Solid black lines indicate inherent error due to 12-bit ADC quantization.

\subsection*{4.3.5 Handling Switch Parasitics}

In order to share access to a single integration path, care must be taken to adequately isolate each channel in time so as to minimize cross-talk. Cross talk occurs because of switch parasitic capacitances $C_p$ that add up when the number of channels is high. $C_p$ appears in parallel with the integrator channel capacitance $C$ and gets charged by the same current. When the channel is switched to a different capacitor in the bank, the charge on $C_p$ gets re-distributed to the new channel capacitance that was switched in. To reduce the effect of cross-talk, the ratio $C_p/C$ must be made very small (by increasing $C$), but this results in a high integration time $\tau$, which reduces the bandwidth performance of CapMux.

Instead, subsequent to each integration, after switching out one channel and before switching in the next one, we first discharge the accumulated charge on the switch parasitic capacitance. This is done by setting the opamp input to zero and closing the negative feedback path. This condition is maintained for $\tau_p = V_{\text{max}}/SR$, where $V_{\text{max}}$ is the voltage rail of the circuit and $SR$ is the nominal slew rate of the amplifier in V/s. The effect of this change is a slight reduction in the chipping rate to $f_c = 1/(m(\tau + \tau_p))$. 

77
Figure 4.8: An example of per-column error for analog matrix multiplication after calibration, with density $d = 8$ and sparsity $s = 3$. Samples where the corresponding element of the sensing matrix is 0 are omitted.

4.4 Evaluation

All results in this section have been measured from a 16 channel CapMux prototype as shown in Fig. 4.9. Digital logic for random number generation and switch synchronization is executed by a microcontroller that also provides synthesized test signals. Fig. 4.10 shows the result of acquiring and recovering a 3-sparse signal in the frequency domain. Overlaid are the actual, ideally-recovered, and experimentally-recovered signals, showing 35.7 dB of measured SNR, while the ideal recovery (with perfect analog components) would have yielded 41 dB. The right side of Fig. 4.10 illustrates the sensing matrix used, the 16 measurements acquired and their error. Fig. 4.10 showcases the power of CS – a seemingly complex signal can be compressively sampled, and recovered with high accuracy given only the sparse domain and a random sensing matrix.
4.4.1 Universality

In order to verify that the CapMux system generalizes to all domains as per the theory that underlies it, we tested across time, frequency, and discrete wavelet domains, input signals of varying sparsity $s$, and sampling matrices of varying density $d$. For each $s$ and $d$ pair, we evaluate both the experimental SNR of the recovered signal from the CapMux hardware and the theoretical SNR under ideal conditions, given the timing and transfer characteristics of the system determined in calibration. The resulting SNRs and error bars bounding the first and third quartiles for all three domains are shown in Figs. 4.11, 4.12, and 4.13, respectively.

As shown earlier in simulation results in Fig. 4.6, the probability of recovery has a maximum value when $d$ is equal to half the number of channels, $d = 8$ in this case. Similarly, for $d < 8$ we observe an increase in the SNR of the recovered signal with an increase in the sensing matrix density for both the time and the wavelet domain. In the frequency domain, however, this does not hold. This is due to the nature of the bases involved – the sampling domain (time) and the recovery domain (frequency) are perfectly incoherent. In other words, a sparse binary matrix of density greater than 1 is superfluous. Additionally, for frequency domain sparsity, the loss in SNR performance is low compared to simulated because the actual integrated values are high in amplitude. We find that the SNR of the recovery is consistently high when the signal being integrated is also large.

In general, the SNR remains high (> 20 dB) for signals of sparsity $s < 4$, barring sensing
Figure 4.10: An example of compressed sensing and recovery of a 3-sparse signal in the frequency domain, with $d = 8$ using the CapMux hardware. The left plot shows the actual, ideally recovered, and experimentally recovered signal. The right side shows (from top to bottom) the sensing matrix (black = 1), the ideal and measured values after matrix projection, and the errors in mV of these projected values. The recovered signal has 35.7 dB SNR.

matrices of very low densities in the time domain, where the sampling and recovery domains are the same. In the time and wavelet domains, recovered signals have on average above 30 dB SNR for $s < 4$. Recovering signals where $s > 4$ with high fidelity would require greater than 16 channels of hardware. As we will see in the following section, the CapMux architecture can sample signals with much higher $s$ values by scaling the number of channels with little or no reduction in signal bandwidth.

4.4.2 Energy Consumption

Energy consumed while performing compressed sensing in software can be divided into two regimes: sampling and digitizing the analog signal with an ADC and projecting the signal onto a lower dimensional basis via row-wise matrix multiplication. CS in hardware as with CapMux adds one additional regime – that of the analog circuitry required to do the equivalent matrix multiplication in hardware – and reduces the current consumption in the other two. The current demands of the hardware are quite low, and 85% of the total hardware
Figure 4.11: Simulated and measured SNR performance (median) for signals sparse in the time domain. Error bars indicate first and third quartiles. Increasing density of sensing matrix $d$ improves the SNR for a given sparsity for both simulated and actual measurements.

Figure 4.12: Simulated and measured SNR performance (median) for signals sparse in the frequency domain. Error bars indicate first and third quartiles. Increasing density of sensing matrix $d$ provides no substantial benefit in simulated results and degrades the SNR somewhat in actual measurements.

current (20 $\mu$A) can be attributed to the amplifier used for the integration itself (17 $\mu$A), thanks to time-multiplexing. There is only a marginal increase in current due to transient loading of decoupling and parasitic capacitance when switching the capacitors for hardware compression. Fig. 4.14 shows the CapMux hardware current for three compression episodes.

Though hardware compression does not require energy for the explicit multiplication and summation of arrays as required in software CS, it does require software control over the hardware switches. If a low power microcontroller is used, this requires waking the processor out of low power mode. This is also the case when performing software CS, though in that case the microcontroller is awake for longer to multiply and sum the $\Phi$ matrix with $x$. In CapMux, current is consumed to wake up an MSP430G2231 microcontroller for a brief period to switch the appropriate hardware lines for CapMux. During the idle time between
Figure 4.13: Simulated and measured SNR performance (median) for signals sparse in the wavelet domain (Daubechies-4). Error bars indicate first and third quartiles. Increasing density of sensing matrix $d$ improves the SNR for a given sparsity for both simulated and actual measurements.

Figure 4.14: Measured current from the CapMux Hardware, with three episodes of compression. The idle current hovers around 16 $\mu$A while compression adds marginal current for transient loads.

switching, the current is dominated by the quiescent current of the MSP430 – around 1.2 $\mu$A – while during switching the processor exits low power mode and consumes above 350 $\mu$A for a short duration. Keeping this time short is paramount, because for each $\Delta t$ decrease there is significant winnings over software CS where the luxury of forgoing the row-wise $\Phi x$ projection does not exist.

As Fig. 4.15 shows, the average current consumed during software-controlled switching is dependent on the density of $\Phi$. Naturally, there is a trade-off between the energy consumed and the accuracy of recovery. This trade-off exists in software CS as well, because a higher SBM density requires more operations per ADC sample.

Finally, CS in hardware has a distinct advantage over CS in software in terms of energy
Figure 4.15: Average current consumption as a function of sampling frequency: ADC only, ADC with the CapMux analog front end (AFE), and ADC with both AFE and software control of switches for densities $d = 2, 4, \text{ and } 8$.

Figure 4.16: Comparison of total current consumption used by the hardware implementation and a software implementation. The curves converge at higher sampling rates due to the dominance of digital logic current.

used for analog-to-digital conversion. For the 16 channel CapMux, only 16 values need be converted to reconstruct a signal that would have taken 64 conversions for an equivalent software implementation. After a certain sampling frequency (around 10 Hz), the energy per ADC conversion for the MSP430 becomes linear. Thus, 64 ADC conversions will take almost four times the amount of energy required to take 16 samples.

Unfortunately, in our implementation of CapMux, the contribution of the microcontroller’s logic to handle channel switching dominates at higher sampling rates. Fig. 4.16 illustrates that the overall current consumption converges between hardware and software implementations (the software version uses optimized code). However, this result is an ar-
Figure 4.17: Potential for improvement for different compression ratios (excluding digital logic). The dots represent the current upper bound on the sampling rate for a given compression ratio.

tifact of our choice of components. Other choices of digital implementation, especially ones that are more integrated and run at lower voltages may constitute a smaller portion of the overall consumption. As a best case depiction, Fig. 4.17 shows the expected relative efficiency of CapMux vis-a-vis software CS when digital logic for switching is excluded.

As the scale of the system increases, the power savings in terms of ADC becomes even more distinct. A 64 channel CapMux CS system will, for example, require only 64 ADC conversions while the equivalent 64 × 256 software CS system will require 256.

4.5 Discussion and Future Work

Scaling and Sparsity

The most natural extension of this work is to implement a system with more channels, allowing for the compression and reconstruction of higher dimensional (s ≫ 1) signals with little to no increase in power consumption. As described earlier, adding more independent channels involves adding more switched capacitors to the multi-integrator and adapting the synchronization routines accordingly.

Similar to the empirical results of using a 16×64 SBM shown in Fig. 4.6, Fig. 4.18 shows results from a Monte Carlo simulation over 64 × 256 SBMs with varying densities. Note that
densities of \( d > 3 \) result in a high probability of recovery, whereas that of the 16×64 matrix was at least \( d = 5 \). Additionally, the 64 channel results show a recovery probability above 90% for signals of up to sparsity \( s = 12 \) while the 16 channel version has 90% recovery for \( s = 2 \). This implies that sparse binary matrices of low density may be sufficient to acquire signals of longer durations and higher information content.

**Bandwidth**

The number of channels and density of the sampling matrix cannot be increased for free – there is a trade-off for each in terms of the sampling rate and thus overall bandwidth of the system. Increasing the number of channels reduces the compression ratio of the system unless the duration \( T \) over which projections are being performed increases proportionally. Fortunately, SBMs of low density may be sufficient at high channel counts so the admissible signal bandwidth of \( 1/(2d\tau) \) remains approximately constant despite the time multiplexed nature of CapMux. Since sparsity and signal duration may be linked (a longer signal may have more information content), one must be cognizant that a higher channel count is only beneficial when sparsity does not increase faster than the signal length.

The bandwidth of the system can be further increased by decreasing the integration time required for each element. This is achieved by reducing the capacitance per channel, at the cost of reducing SNR due to leakage effects and potentially requiring a higher-power integrating opamp to handle faster slew rates.

**Further Considerations**

- The results presented assume a DC signal as would be seen at the output of a sample & hold (S/H) circuit, though no such circuit was used. Preliminary results show that a S/H circuit can be implemented at the additional cost of 2 \( \mu \)A, but the effect that this would have on the bandwidth and compressibility of the input signal is deferred for future research.

- In order to further reduce the power consumption of hardware CS, dedicated logic
units such as an LFSR synthesized in a low power FPGA can be used to control the multiplexing of the integration circuit.

CapMux, achieves high SNR for reconstructed sparse signals across arbitrary domains. Reconstructing the compressed signals comes at little expense in terms of analog hardware, consuming a mere 20 $\mu$A. What’s more, scaling the CapMux architecture to a larger number of channels allows for accurate reconstruction of signals of higher dimensionality with little to no increase in the quiescent current of the analog circuitry, due to time-multiplexed access of a single integration subsystem.

We have evaluated the power required to control the switches of the CapMux front end using a low power microcontroller such as an MSP430, showing how power scales with the density of the sensing matrix, $\Phi$, and sampling rate.

Towards demonstrating the efficacy of the system as a scalable solution, we presented empirical results for a scaled version of the current CapMux circuit up to 64 channels. The 64 channel simulations suggest an almost six-fold increase in the ability to recover sparse signals without decreasing the sampling rate of the compressor.
CHAPTER 5

Concluding Thoughts

The work presented in this thesis suggests three methods for increasing the efficiency of embedded sensing platforms: first, we described a series of operating system abstractions for improving the quality of service for long-lived sensor deployments where sensing is performed periodically and, potentially, infrequently. We further argued that increasing variation in energy consumption for embedded devices necessitates explicit treatment of instance-specific power, demonstrating that doing so can reduce errors in energy expenditure by, in some cases, over 30x; second, we argued that sporadic time-series data streams—those characterized by short and irregular bursts of information followed by long idle periods, like that seen in water monitoring systems—can benefit from an event-triggered sensing architecture coupled with opportunistic energy harvesting. We presented DoubleDip, an event-driven water monitoring system capable of lifetimes in excess of 20 years in some cases; third, we described a new hardware architecture and implementation for compressed sensing, demonstrating a scalable solution for performing hardware compression for signals that are sparse in any domain (not just time). This architecture consumed a mere 20 $\mu$A of current and amortizes energy spent converting analog signals to digital.

Going forward, as device sizes continue to diminish and battery advancements continue to lag, new techniques for saving energy and maximizing quality will be required to meet increasing processing demands. Compressed sensing was introduced as one method for reducing energy requirements with little impact on sensing quality, but, as Thomas Strohmer put forth in [Str12], “maybe the most important legacy of compressive sensing will be that it has forced us to think about information, complexity, hardware, and algorithms in a truly integrated manner.” This rings true particularly for the world of embedded sensing and
embedded systems in general; it is likely that a full-stack approach, from hardware to OS, will be required to manage energy in a way that is in keeping with rising demands and a looming energy crisis.
References


