System Design Trade-Offs in a Next-Generation Embedded Wireless Platform

Michael P Andersen
David E. Culler

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-162
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-162.html

August 25, 2014

System Design Trade-Offs in a Next-Generation Embedded Wireless Platform

Michael P Andersen
Department of Computer Science
University of California, Berkeley
m.andersen@cs.berkeley.edu

David E. Culler
Department of Computer Science
University of California, Berkeley
culler@eecs.berkeley.edu

Abstract

Over the course of the past decade, the evolution of advanced low-energy microcontrollers has raised three questions which this paper outlines and addresses.

The first question is: Can a 32-bit platform be constructed that provides advanced features but fits within the energy constraints of a wireless sensor network? We answer this in the affirmative by presenting the design and preliminary evaluation of Storm – one such system based on an ARM Cortex-M4 that achieves 2.3μA idle current with a 1.5μS wake up time.

The second question we answer is: Can this platform simultaneously meet the very different demands of both monitoring-type applications and cyber-physical systems? We demonstrate that this is indeed possible and present the design trade-offs that must be made to achieve this, yielding a module with a rich set of exported peripherals that fits in a 16mm x 26mm form factor.

The final question explored by this paper is: If such a platform is possible, what new opportunities and challenges would it hold for embedded operating systems? We answer this by showing that the usage of modern 32 bit microcontrollers requires reconsidering system architecture governing power management, clock selection and inter-module dependencies, as well as offering opportunities for supervisory code and the coordination of common tasks without CPU intervention.

Categories and Subject Descriptors

B.0 [**Hardware**]: General; C.3 [**Special-Purpose and Application-Based Systems**]

General Terms

Design, Performance

Keywords

Wireless sensor networks, Energy efficiency

1 Introduction

In the early years of wireless sensor network research, hardware platforms evolved rapidly and operating system structures were transformed by each new generation. From 1999 until 2004, each release of a significant microcontroller or radio advance was incorporated into a new open platform, including WeC [15], Rene [12], Mica [11], MicaZ, iMOTE, BTnode, EYES, iMOTE2 [16], Telos [17]. But since the consolidation around 16 bit microcontrollers (MSP430, ATmega) and IEEE 15.4 radios (CC2410, RF231) a decade ago, new platforms have largely been variations in module form factor, e.g., Epic, Shimmer, despite tremendous growth and advance in essentially all aspects of the industrial ecosystem around embedded networks, wearable technology, and cellular platforms. And, ever since the earliest generations a basic question was whether it was viable to utilize a full 32-bit processor, with adequate storage and the associated widely available tool chains, while meeting a power profile that permitted lifetimes on the scale of battery shelf life and a small part count. The introduction of xSCALE and ARM microcontrollers brought down the part count and improved the active power efficiency, but low power operation remained an elusive challenge. System-on-a-chip options emerged, bringing the part count down further, but they had extremely limited processor architectures and weak tool-chain capability. In this paper, we examine whether this situation has finally changed.

In particular, we address three basic questions.

1. Can we now utilize full-featured, 32-bit microcontrollers with enough memory and flash to support sophisticated applications with the power profile of a mote, i.e., idle power of a few uWs, fast wake up, and efficient active operation?
2. Can the platform serve the distinct needs of the two dominant usage models: *wireless monitoring*, with a few sensors and predictable behavior and *cyber-physical systems* with rich I/O, actuation, and dynamic variation?
3. If so, does such a platform introduce qualitatively new operating system challenges and opportunities?

We show by developing a new platform around specific offerings in the Cortex-M family that the answer to the first two questions is affirmative and by examining aspects of this solution we outline a new suite of important system opportubities and challenges. Indeed, the building blocks are finally of a state where the integration into a system-on-a-chip is likely to produce extremely general, cost-effective solutions.

Section 2 frames the investigation with an enumeration and characterization of the demands placed on a modern wireless embedded platform, forming the criteria for the evaluation of a next-generation mote. Section 3 discusses current trends in microprocessor, transceiver and SoC development, leading to a blueprint for a wireless embedded system that is representative of current trends in industry and meets the demands of current and future wireless embedded networks.

Addressing the first question requires not just an analysis of data sheets; a quantitative, empirical study of the complexities and implications of utilizing next-generation hardware in sensor networks requires the careful design of a physical platform. Section 4 presents one such system – Storm – an example reference platform based upon best-in-class next-generation components. The process of mapping the model of a representative wireless embedded system into a physical instantiation by evaluation of available components and selective design trade-offs is discussed. A physical module design is presented that extends and improves upon the 3Ps [9] to serve the range of usage models from simple sensor networks for monitoring to sophisticated cyber-physical systems.

The Storm platform is then used as a representative for next-generation wireless platforms in general for an exploration of new systems opportunities and challenges in Section 6. We identify five primary factors – modular power management, multiple clock domains, inter-module compatibility, chaining of multiple overlapping transfers and increased supervisory control – which lead to a whole-system optimization framework for real time embedded operating systems, such as TinyOS. Such intricacies naturally pose new problems for the architecture of any embedded operating system aiming to abstract device-specific complexity from users by utilizing layering and modularity.

2 Requirements of a modern wireless platform

As sensor networks have been utilized as solutions in a growing number of fields – such as medicine [5], building management [4] [10], energy usage awareness, security and ecological studies [14] – the demands placed upon individual sensor nodes have become more sharply defined.

2.1 Microprocessor resources

Resource bound applications, such as point-of-origin data analysis or feature extraction, distributed computation, frequency domain techniques, and so on utilize advanced algorithms. However, even simple network stacks generally fill most of available memory on traditional mote-class platforms. For example, Table 1 lists the program memory and RAM space requirements for some configurations of applications that currently ship with TinyOS [1].

Basic applications, such as UDPEcho, barely fit in the 48KB of program space afforded by the MSP430F1611, and some applications such as the PPPRouter do not fit at all[1]. It is clear that even for applications that are not uniquely large or complex, more program memory is required. Computational requirements vary widely, generally with bursts of processing and long idle periods.

Table 1: Requirements for TinyOS applications targeting TelosB with BLIP.

Application	Flash	RAM	Config
Null	3530	1506	-
UDPEcho	39092	6752	static
UDPEcho	42400	6864	dhcp
PPPRouter	FTBFS	FTBFS	-

2.2 Peripheral requirements

In addition to computational and storage resources, many applications place heavy demands on the peripherals of the nodes. This is particularly prevalent in cyber-physical systems where the mote may form the core of a much larger system composed of several sensors and actuators, requiring modules such as PWM controllers, external communication interfaces and high speed analog to digital converters. Some examples of this include mobile medical devices[19] or embedded robotics[8].

An interesting observation is that although many of these cyber-physical systems require increased IO, they simultaneously require a small form factor [18]. This means that a fully generic platform must be able to provide a rich set of peripherals, while also remaining compact.

2.3 Energy budget

Many deeply embedded "deploy and forget" wireless sensor networks utilize battery powered nodes that aim to be low cost and zero maintenance. The primary requirement imposed on the systems is that they must be capable of extremely low idle currents and low duty cycles. In addition, this category of research often focuses on larger deployments of cheaper sensors.

Learning from prior platforms and their evaluation, such as the TelosB [17], two primary characteristics influence the energy efficiency of the system: the current in the processor's lowest useful power saving mode and the wake-up cost. We qualify this with "useful" because many components provide power saving modes that, while impressive, are difficult to use except in specialized applications. The most common are power saving modes where the contents of SRAM are not retained, or where all clocks are stopped and no interrupts can occur. We define the lowest useful power mode as one where there is at least one timer running that is capable of waking the processor up to full running state at some predetermined time in the future. This naturally leads to the second important metric of how long it takes the processor to leave this low power mode and begin executing instructions.

2.4 Adaptability

The requirements placed on the system with respect to resources, peripherals and energy may vary with modes of

[1]This was true at the time of writing: commit ID 14411b7dbe5d5

Table 2: Component minimum operating voltages in various mote platforms.

Platform	MCU	MCU Flash	Flash	Radio
TelosB	1.8	2.7	2.7	1.6
Epic	1.8	2.7	2.5	1.6
MicaZ	2.7	2.7	2.5	1.6
Storm	1.7	1.7	1.8	1.8

operation. For example, a cyber-physical system may have little peripherals activity while it does significant computation in order to implement advanced algorithms to control its actuators, or it may idle for long periods; a deeply embedded system may on occasion require the ability to interface with a large bank of external sensors via GPIO. This common combination of requirements makes a platform such as the Imote2 [16] have limited applicability because while it offers increased computational resources, it also comes with a high idle current(390 μA [16]) and monetary cost.

This adaptability requirement is a moving target – the characteristics of a system may not remain constant, even within a specific application. A example of this is a network of solar powered sensors. Here, when there is no available sunshine, the nodes may be in a very conservative power mode, only acquiring sensor data with a low duty cycle. But, when sun becomes available, the nodes utilize the plentiful energy and perform calculations, transfer data across the network or become routing nodes for other lower power nodes. These applications place a large dynamic range requirement on the capabilities of node hardware.

2.5 Storage

The primary reasons for including a flash chip are that it allows for storage of sensor data while the radio is unavailable, and it enables storing of alternate program images for over-the-air firmware updating. In most platforms, the external flash chip requires a higher voltage than the MCU, as shown in Table 2. This means, for example, that a mote running from two rechargeable AA batteries offering 2.4V will be unable to utilize the flash.

An oft overlooked aspect of storage is the program memory flash within the processor itself. Although this is typically designed for a lower number of erase/write cycles so has limited applicability for data storage, it is important for over-the-air updates. If the internal flash requires a much higher voltage to program than the rest of the system requires to operate, then it will constrain the applications for which the platform can be used.

3 Current technological options

The technology available for use in wireless sensor networks has evolved over the past decade and the options for the constituent modules in a platform have grown. This section reviews the advances that have been made in each category and establishes a blueprint for a platform constructed from best-in-class components.

3.1 Microcontroller

A microcontroller can be gauged on two broad characteristics: its capabilities, and the energy that it consumes to offer those capabilities. While microcontrollers have seen significant development in both directions over the past decade, for the purposes of this paper we opted to keep the energy characteristics comparable to existing ultra low energy sensor platforms, while maximizing available feature set. This is primarily because the idle currents of ultra-low-power MCUs are comparable to the idle currents of other components on the board, so decreasing MCU idle energy consumption further is of little benefit. This choice is also motivated by the observation that it is easier to predict the effects of increased energy efficiency – longer battery life – whereas the effects of a richer set of capabilities provides a more interesting area for research, as explored in Section 6.

To form a baseline energy profile, we opted to use the ultra low power characteristics of the popular TelosB platform as, at the time of writing, it has the most impressive idle currents and wake up times. The reported idle current is just 5.1 μA with a wake up time of 6 μS [17].

Within this energy bracket, the biggest microcontroller design choice is which architecture to use. Is the choice to use a 16 bit processor still valid given the proliferation of low-energy 32-bit processors? If we consider that the algorithms being developed for sensor networks are growing in complexity and we are seeing a proliferation of computationally demanding applications even in small battery powered devices, we can conclude that – assuming it meets the energy demands and other criteria – a 32 bit processor would be useful. Of the available 32 bit architectures, the three that are most common are ARM, MIPS and x86.

Of these, only ARM processors are available in with the required power consumption, and we have seen an explosion of developments, with new microcontrollers based on the ARM Cortex-M family of processors being released every month. This widespread popularity means that there are mature product options available from multiple vendors – making it likely that among the many choices of processor available, there exists a subset that meet the requirements of embedded wireless platforms.

An additional benefit of choosing a well used architecture such as the ARMv7E-M offered by the ARM Cortex-M4 microcontrollers is that porting code written for a given processor to newer processors is likely to be far easier. This is important in the context of academia where research is often done by students who finish and move on, leaving code that must be maintained by those unfamiliar to it.

As noted in Section 2, many of the problems that currently plague researchers are related to the amount of available program space. The 10KB of SRAM offered by the MSP430 is often ample space for a conservative developer, but the 48KB of flash is artificially constraining, especially with a more complex network stack. The move to a 32 bit processor increases the size of instructions, so a given program would correspond to greater flash occupancy, as seen in Section 5.3. Fortunately, however, there are several Cortex-M microprocessors available with well over 256KB of flash.

In addition to plentiful computational resources and memory, several Cortex-M microprocessors introduce a feature that Atmel names "Sleepwalking". The feature is present in offerings from multiple vendors, although it goes by dif-

Table 3: Estimated power consumption across the Cortex-M range for TSMC 90LP fabrication [6]

Processor	μW/Mhz	DMIPS/Mhz
Cortex-M0	16	1.21
Cortex-M0+	9.8	1.31
Cortex-M3	33	1.89
Cortex-M4	33	1.91

ferent names such as "Peripheral Reflex System" in processors from Silabs. This capability allows certain peripheral events to be connected to other peripheral triggers so that rudimentary event chains can occur without any processor intervention. The subsystem also undertakes to enable the clocks that the triggered modules depend on when they are triggered, and disable them afterwards. The implications of this feature are discussed more in Section 6.4.

Although argument for ARM Cortex-M processors appears conclusive, the question of whether to go for the least capable or the most capable processor in the range, being the Cortex-M0+ or the Cortex-M4 respectively, still remains. We advocate the Cortex-M4 for two reasons. The first is a reiteration of the argument made earlier: it is easier to predict the effects of increased energy efficiency, so it is more interesting to study the effects of innovative features. The second is that when whole-system energy costs are accounted for, it is often the case that a faster processor leads to lower total power consumption.

These points aside, the two series of processor are close enough in power characteristics that other factors dominate (the fabrication process, selected peripherals etc). Table 3 shows ARM's characterizations for the standalone Cortex core's power usage but as will be seen in Section 4, the real power consumption of a Cortex-based processor is far more dependent on the vendor specific configuration than on the processor core itself.

For these reasons, the best-in-class microcontroller technology at the moment is likely to be based on an ARM Cortex-M4 core.

3.2 Radio

When radio transceivers are evaluated for energy constrained embedded wireless systems, there are two major factors that are typically considered. The first is the time it takes for the radio to exit its low power sleep mode until it is able to transmit. This is typically considered to be important because other components in the system remain powered up while the radio is starting, and because the radio itself draws higher current during this time.

The second is the current drawn during transmission and reception. The radio often dominates the power budget of a mote especially for nodes in the mesh that need to remain active for long periods of time in order to route traffic for others. While the MCU can go into deep sleep and be woken by the radio interrupt, the radio itself must remain actively listening. As such, low power listening modes are important even though they come at the cost of reduced gain.

802.15.4 radio transceivers have not experienced nearly as explosive a proliferation as microcontrollers. A modern radio, therefore, offers core functionality similar to those used in previous generations of embedded wireless platforms albeit at a lower energy cost and a lower price point (the TI CC2520 currently costs half what the previous generation TI CC2420 costs). In addition, the inclusion of hardware accelerated MAC features such as automatic CSMA/CA, automatic retransmission and automatic acknowledgement has appeared in at least one radio transceiver, as discussed in Section 4.

3.3 Flash

As discussed in Section 2, current generations of wireless platforms often utilize flash chips that are unable to operate at the low end of the system's supply voltage range. There are, however, several flash chips available that are designed to run at different voltage ranges. For example, Micron manufactures serial NOR flash in 2Gb densities that can run from 1.7V to 2.0V.

Unfortunately, this poses a problem. One can choose the higher voltage flash as used in previous generations so that the mote is capable of running from 3.3V, but then the flash cannot be used when running from low voltage power sources. The alternative is that a 1.8V flash chip is used and then the whole mote is run at 1.8V, but then care must be taken to regulate input battery voltage such that the voltage never exceeds 2V. The latter is not a bad choice, as advances in switched mode power supply regulators have led to compact, high-frequency buck converters capable of achieving excellent efficiency with small inductors [3]. This would mean that a mote could last longer off the same power supply.

There is one critical drawback to the mandatory low voltage option that prevented us from choosing it. If the system were to run at 1.8V, then all IO would have to be at 1.8V. There are many sensors and components available that are unable to operate at such low voltages and precluding their use would limit the generality of the platform.

Fortunately, recent advances have yielded flash chips capable of full functionality over a 1.8V to 3.6V range, albeit at a comparatively low density. This development allows for the platform to truly run at low voltages while still retaining the ability to operate at higher voltages, a hitherto unreachable goal.

3.4 Trends in System-on-Chip design

We are beginning to see developments in System on Chip technology where a powerful microprocessor is combined with a radio transceiver. While the combination of a microcontroller and radio in a single package is not new, it is only recently that chip miniaturization has allowed for this combination to utilize processors such as the Cortex-M4 and to be paired with radio transceivers that are themselves impressive.

A prime example of this is the Freescale MKW2xDx series that combines a 50Mhz Cortex-M4, 802.15.4 radio transceiver, 512KB of flash and 64KB of SRAM in an 8x8mm land grid array package [2]. This is essentially a first-class mote in a single package. This particular chip is, at the time of writing, still a brand new product that has not

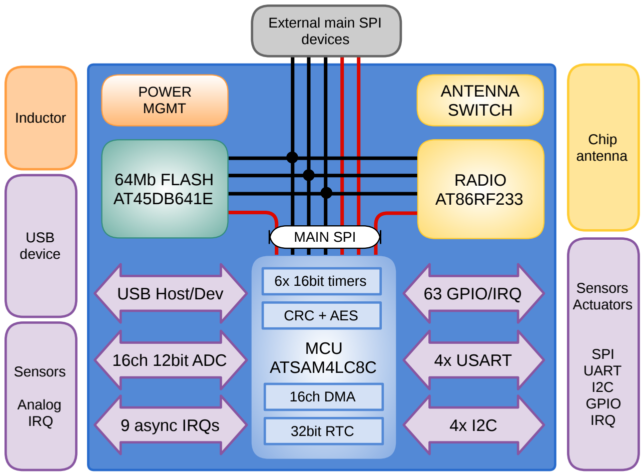

Figure 1: The Storm platform architecture

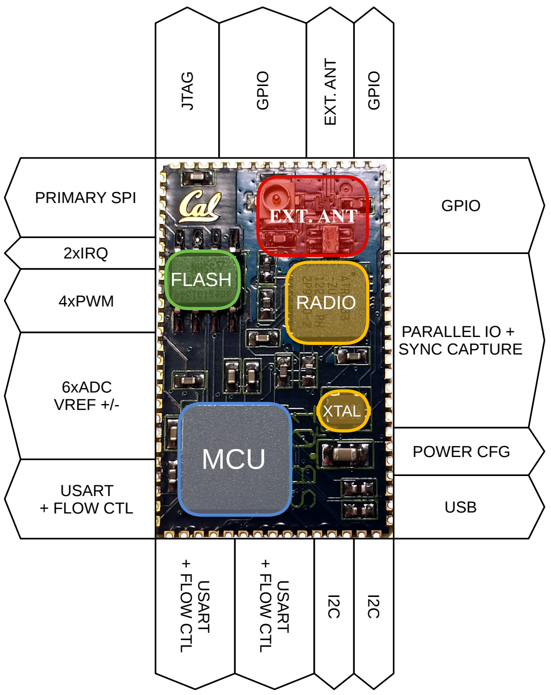

Figure 2: The Storm module

reached general availability but represents a trend in industry. We predict that there will soon be a proliferation of such SoCs at competitive prices.

The next step from integration of the radio into the SoC is the integration of an energy source into the package, a concept which although not currently prevalent in mass produced products, has been proven possible with stacked dice [13].

4 Can a 32-bit processor be low power enough?

With the context of available technology established, in this section we address the first question posed by Section 1: Is it possible for a system based on a fully-featured 32 bit microprocessor to perform within a tight energy budget? We answer in the affirmative by way of example with Storm, a reference platform based on a SAM4L 48 Mhz Cortex-M4 microcontroller, AT86RF233 802.15.4 radio and AT45DB081E 8 Mbit flash that serves as both a set of design guidelines for constructing 32 bit platforms with "mote-class" energy budgets and as a means for empirical evaluation of such a system in Section 5.3.

4.1 Overview

We begin with a brief overview of the platform before diving into the components that constitute it. Figure 1 shows the architecture of the Storm module in the main block, with components on the carrier indicated outside the block. The individual parts and peripheral signals of the module are illustrated in Figure 2. Note that most of these signals are multiplexed, and only one of their possible functionalities is indicated.

4.2 Microcontroller

Although Section 3 concludes that an ARM Cortex-M4 is the current best-in-class embedded processor, this does not narrow down the available choices much. There are, at the time of writing, 196 licensees of the Cortex-M family of intellectual property [7]. Each of these licensees represents one vendor who in turn combines this processor with various peripherals, memory and flash. Even after the all the functionality is fixed, different vendors use different fabrication processes, leading to different costs and energy usage. Table 4 shows a few of the hundreds of Cortex-M4 processors from a handful of vendors. These selections are all the most capable of their respective families, but not necessarily the flagship family from the vendor.

After evaluating several offerings from nearly a dozen vendors, a few guidelines emerged to narrow it down to the finally selected chip – the ATSAM4L. These discriminators were, in the order of efficacy:

- The available flash and RAM on the chip
- The available packaging and pin count
- The availability of the product – many very promising chips are advertised as being available but are in fact several months from production.
- The comprehensiveness of the documentation – if the necessary figures of merit are not described in the datasheet, they are evidently not important to the manufacturer.
- The current consumption of a useful low-power mode and its wake up time
- The granularity of its low power modes

The Atmel ATSAM4LC8CA, indicated in bold in Table 4, was chosen as it had best-in-class energy characteristics, sufficient flash, RAM and IO, as well as a comprehensive power management system and significantly better documentation

Table 4: A small sample of available Cortex-M4 processors

Vendor	Device	f_{max}(Mhz)	SRAM(KB)	Flash(KB)	Sleep(μA)	Wake(μS)
NXP	LPC408x	120	96	512	550	240
STMicro	STM32F372xx	72	32	256	1.32	42.7
Silabs	EFM32WG990	48	32	256	0.95	2
Freescale	K20Dx	50	16	128	1.3	130
Atmel	**SAM4L**	**48**	**64**	**512**	**3**	**1.5**

than other vendors.

A noteworthy feature of the processor is that it has 16 independent DMA channels. While the MSP430 family had DMA, the limited number of channels and the multiplexing of communication peripherals contained it's implications. The SAM4L, however, has several more independent communication mechanisms, so the impact of the DMA is greater.

4.3 Radio

The radio is an important component to select, as it is inescapably responsible for a significant share of the energy budget. Fortunately, as discussed in Section 3, there are only a handful of choices available. These are laid out in Table 5. The Freescale offering is a (currently) unreleased SoC that includes a Cortex-M4 core on-die.

As we are targeting compatibility with existing infrastructure, it made sense to use 2.4GHz IEEE 802.15.4, making the two obvious choices for the radio chip the second generation TI CC2520, and the Atmel AT86RF233. The CC2520 would be the easier chip to develop support for, as it is the newer version of the ubiquitous CC2420. The chip is, however, already more than five years old and lacks several features that the newer Atmel chip offers. Both chips are the flagship 802.15.4 transceivers from their respective vendors at the time of writing.

One key difference between the Atmel and the TI chip is that the AT86RF233 is capable of automatic retransmission of packets that require acknowledgement but do not receive it, enabling automatic CSMA/CA. The TI CC2520 does not automatically perform CSMA/CA activities, although it does export the CCA line to the microcontroller directly, allowing for faster manual response to an assessment than the in-band CCA signal from the RF233.

When combined with the peripheral event systems found in modern microcontrollers, the automatic CSMA/CA with retransmission can offer significant advantages over the software CSMA/CA methods that are currently employed. The SAM4L, for example, can be configured so that it sends a wake up command to the radio, prepares the packet in memory and goes to sleep. When the radio IRQ line is asserted to indicate it is ready for transmission, the clock for the SPI clock domain is automatically started, the packet is copied via DMA from MCU memory to the radio and transmission is triggered. As soon as this process completes, the SPI clock is automatically stopped again all utilizing the peripheral event system. This means that the entire process of sending a packet, from the radio wake up command to when the ACK is received, can be performed without waking up the processor. With the CC2520, the MCU would need to wake up, check for acknowledgement, wait some back-off time if the acknowledgement did not arrive, and then trigger retransmission.

For reception, both chips feature automatic acknowledgement. The CC2520 has the feature to specify which addresses should receive ACKs with the frame pending bit automatically set - this feature is not present on the RF233. This is quite useful if motes are utilizing the frame pending signal as, in the absence of this feature, the MCU would have to wake up after frame reception, parse the address, determine if the bit should be set or not and configure the radio accordingly before the ACK is sent. It was decided that this did not constitute a big enough problem to outweigh the automatic retransmission for two reasons, the first is that it is likely that the MCU will want to wake up and receive the packet from the radio as soon as it is received anyway, so the burden of setting the frame pending bit may prove to be insignificant. The second is that, at least in our use cases, full duplex communication tends to be short lived and the frame pending bit is not of paramount importance.

In the past, with the AT86RF230, the automatic acknowledgments were tied to automatic address filtering - rendering them useless if the mote needs to snoop on traffic addressed to other motes. The RF233 changes this by adding support for 802.15.4-2006 promiscuous mode separately from the automatic acknowledgement address match. This allows for the chip to receive all packets, even if it only acknowledges packets that are addressed to it directly.

Weighing the features, the possibility of automatic CSMA/CA was deemed more useful than selective automatic acknowledgments. This combined with the better power and signal strength characteristics was enough to convince us that the Atmel radio chip was the better choice.

The radio subsystem extends beyond just the transceiver chip itself, however, and includes some thought into how the antenna will be connected to the radio. The Telos and Epic motes utilized a lumped element balun with the final capacitor having two possible locations. This serves as an assembly-time switch between an external antenna connector and a PCB trace antenna connector. To reduce size, BOM cost and variability during manufacture, we decided to use a single integrated balun/filter that was designed specifically for matching the RF233 to a 50 Ω single-ended antenna.

The final major change in the radio subsystem is the use of an antenna diversity switch instead of using an assembly-time capacitor location choice. This allows a single universal module to utilize both an external antenna mounted via U.FL

Table 5: Key metrics for select radio transceivers

Year	Vendor	Device	TX (dBm)	RX (dBm)	Wake (μs)	Sleep (μA)	TX (mA)	RX (mA)	CCA	AES	Auto ACK	Auto RE-TX
2013	Atmel	RF233	+4	-101	450	0.02	13.8	11.8	Y	Y	Y	Y
2007	Ti	CC2520	+5	-98	500	1	33.6	24.8	Y	Y	Y	N
2013	Freescale	MKW24D512V	+8	-102	-	-	18	19.5	Y	Y	-	-

and a chip or PCB trace antenna connected via the signal exported on the edge of the module. This decision was made after observing the usage of Telos motes in an educational environment, it was realized that a given mote is often retasked. Having the ability to swap between a larger, more powerful whip antenna and a more compact chip antenna without altering the hardware would increase the generality of the mote.

4.4 Flash

As discussed in Section 2, it is important to have a flash chip that offers sufficient storage for program images and data, but also that operates over the full voltage range of the device. These requirements lead to a very small pool of available options

At the time of design, Adesto Technologies was releasing a new 64 Mbit flash chip - the AT45DB641E - that operates over a supply range of 1.7V to 3.6V. This would allow the entire system to operate over 1.8V to 3.6V. The downside is that this flash is not yet readily available. The prototype versions of the platform used for evaluation in this paper utilize the smaller 8 Mbit density edition, although the production configuration will use the 64 Mbit version that the vendor assures us will be available by Q3 2014.

The AT45DB641E flash offers comparable power characteristics to the chips selected for previous motes. The lowest power mode utilises 1 μA, which is significantly less than the 15 μA consumed by the AT45DB161D used by the Epic or the 10 μA used by the Telos' M25P80. Programming consumes 14 mA, which although slightly higher than the flash used in the Epic, is still reasonable.

The radio chip and the flash chip are placed on the same SPI bus, which is also connected externally. The reasons for this are threefold. Firstly, although there are five possible SPI channels on the microprocessor, the primary SPI bus is more energy efficient than the USART SPI channels. Secondly, the primary SPI bus has four logical channels, with independent automatic chip select capabilities, baud rates, programmable inter-byte delays and polarities so there is no need to adjust configurations between flash and radio access. Finally, the primary SPI peripheral encodes the channel and chip select into each word that is written to the transmit register. This allows for a single DMA transfer to encode a sequence of multiple commands, even to different devices on the bus. This final point is of import when one wishes to transmit a packet to the radio as it allows for queuing multiple commands requiring the CS line to be de-asserted between them.

5 Diverse design points

We turn our attention to addressing the second question posed in Section 1: can a platform meet the requirements of both monitoring systems with strict power requirements and cyber-physical systems with diverse peripheral and IO?

This too, is answered in the affirmative by consideration of two aspects in the platform design: how the module presents the IO and how core system requirements are handled. We then present a set of microbenchmarks to evaluate the system and show how the design has enabled an example cyber-physical system.

5.1 Form factor and assembly

Cyber-physical systems require both large numbers of IO and small form factors. In the design of the Storm module this requirement required careful design to meet. While previous module designs have emphasized exporting as many internal signals as possible in order to increase generality and facilitate easy debugging, the sheer number of signals present in the Storm platform makes that an untenable proposition. The Epic core, for example, utilized castellated edges in an LCC-68 form factor to export its IO [9], which was enough to cover the signals of interest but would not cover even just the GPIO on the Storm.

As a brief summary, the microprocessor utilized by Storm has 78 IO connections excluding pins reserved for power, voltage references, and programming. In addition to this there are 8 pins that could be exported by the radio and the flash. To export every useful signal while still providing sound power connections would require either a high density connector, a ball grid array connection, or an artificially increased module size to allow for all the connections to be made on the edge.

All of the available form factors have problems associated with them. A high density connector would increase the height of the complete assembly when mated with a carrier, limiting usefulness in space constrained mobile sensing applications. In addition connectors have a tendency to become unavailable and are often a significant portion of the BOM cost (at the time of writing, the cost of the pair of mating 51-pin connectors used in the MICAz and MICA2 motes adds up to over \$6 - greater than the cost of the \$5 radio transceiver). Even if a cheap high density connector with guaranteed availability was found, motes deployed in the field often experience mechanical failures with connectors.

They second option of a soldered connection on the underside of the module, such as a ball grid array or land grid array would not add to the BOM cost, and would not increase the height of the final assembly. The key downside to such a connection is that it would then require specialized equipment to assemble onto the carrier and clean - exactly the problem we are attempting to avoid.

Table 6: Storm external peripheral and IO capabilities

Peripheral	Count
GPIO / sync IRQ	63
USB Host	1
PWM Channels	12
USART[2]	4
Primary SPI channels	2
ADC channels	14
I^2C channels	2 master + 2 master/slave
Async IRQ[3]	9
16bit PD DAC[4]	2
10bit DAC	1
8bit sync capture	1

The third solution - to artificially increase the PCB size to allow for all connections to appear on the edge of the PCB - would allow for cheap, mechanically sound and easy to mount modules at the cost of increased PCB area. This is still not ideal, as we wish to create a platform for use in space constrained environments as well.

The compromise is a variation of the third – to export the most useful subset of signals as castellated connections on the board such that the total board size is not increased beyond a useful size. In addition, an observation was made that for many applications, the enclosure may be longer in one dimension with less penalty than if it were uniformly increased in size over two dimensions. This is often because the battery may already be rectangular, or because the sensor is being worn on a finger or a wrist - both allowing longer thinner objects.

As such, a rectangular PCB measuring 26 mm by 16 mm was chosen with castellated edges having a 1mm pitch. This yields 80 connections as the corners cannot have routed vias for manufacturability reasons. Of these, 9 are related to power and voltage references, 3 for the antenna and the RF shield and 5 for programming. The remaining 63 are available as GPIO. Table 6 lists a subset of the exported functionality of the module.

There are only 6 internal GPIO connections that are not exported, as they would have shadowed more useful signals. These are the radio IRQ line, the radio sleep/transmit line, the radio reset, the radio clock, the radio chip select and the flash chip select.

As the flash and radio are both on the primary SPI bus that is exported outside the module, it is still easy to do low level inspection of communication between the devices on that bus. Figure 3 for example shows the start of a RPL DODAG Information Object packet being received by the platform as part of the verification of functionality in the code size test discussed in Section 5.3. The absence of the radio chip select line is not critical, as the command boundaries can be inferred by the inter-frame delay and known start sequences.

As well motivated by [9], separating concerns between a *core* module and *carrier* boards allows for continued innovation in the applications of sensor networks with application specific peripheral circuity while still isolating researchers from the task of designing the core functionality which requires *deep expertise*. A solder on module allows the platform to cater to the "3 P's": prototype, pilot and production [9] as it is easy to probe and debug during prototyping, but remains cost effective during pilot and production.

In addition to these arguments, there is an additional reason to separate the core module from the application specific circuitry - manufacturability. High density microprocessors and compact radio transceivers are now only available in packages that are not possible to work with without specialized equipment, whereas the castellated edges form factor leads to a compact and essentially free method of connection with sensor boards that remains easy to hand solder without any specialized equipment. An example carrier board can be seen in Section 5.4.

5.2 Power

Sensing platforms have diverse power supplies, ranging from coin cell batteries to solar panels to energy harvesting circuits. To remain useful for all these applications the module is capable of operating with full functionality over 1.8V to 3.6V.

For space constrained systems, the processor can be operated in linear mode, where it derives its internal core voltage by linearly regulating the power supply down to the 1.65V used to drive the internal logic and clocks. For operation from power supplies that regularly drop to below 2.1V or are externally regulated to 1.8V this is the optimal configuration.

For use cases where the carrier can fit a small 22 μH inductor and the power supply mostly stays above 2.1V, the processor can be operated in buck mode. When dealing with voltage supplies of 3.6V as common with Lithium Polymer or Lithium Ion batteries, this results in a reduction of processor power consumption of more than 50%, even with the inefficiencies of the buck converter. As the current requirements of the processor are low, the inductor can be very small. The recommended inductor measures 3mm x 3mm x 1mm and costs \$0.3, but smaller alternatives are available for slightly higher cost.

If all of the sensors that the mote is interfacing with are capable of operating at 1.8V, the ideal power configuration is to run the processor in linear mode and externally step down the power supply so that the entire system runs at 1.8V. The leakage current of all components decreases as the supply voltage decreases, and the active current of the radio and flash also decrease. Such a configuration would significantly prolong the lifetime of the mote running from higher voltage battery packs. For example, the radio transmission current would appear to be only 7.6 mA to a Lithium Polymer battery running at 3.7V assuming a conversion efficiency of 90%, well within the capabilities of modern high frequency buck converters ([3] for example).

Experience with carrier boards for the Epic has led us to believe that while exporting separate power rails theoretically allows for external filtering and isolation between power domains, in reality routing constraints or other factors lead many engineers designing carrier boards to deviate from ideal power trace layout. As an example, a low impedance ground connection should ideally be implemented using via-free traces from the various power domain ground pins to

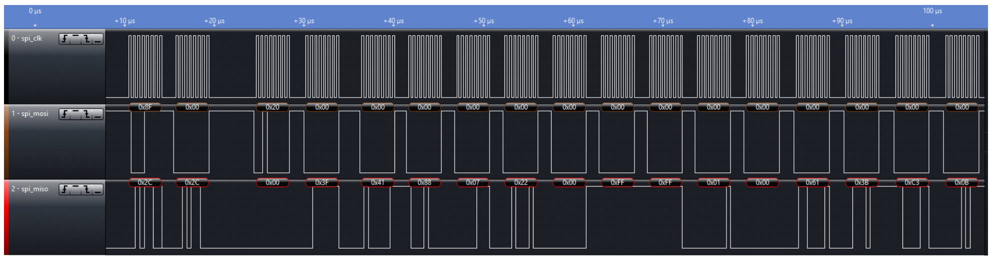
0 µs
+10 µs
+20 µs
+30 µs
+40 µs
+50 µs
+60 µs
+70 µs
+80 µs
+90 µs
100 µs
0 - spi_clk
1 - spi_mosi
0x8F 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
2 - spi_miso
0x2C 0x2C 0x00 0x3F 0x41 0x88 0x07 0x22 0x00 0xFF 0xFF 0x01 0x00 0x61 0x3B 0xC3 0x0B

Figure 3: The initial bytes of a RPL DIO being received as observed via the external SPI bus

a central star-connection. In reality, implementing such a connection severely restricts the routing of other signals of interest on the board, many of them with equally demanding routing constraints - such as USB or analogue traces.

To attempt to mitigate this, we do not provide separate power rails for flash, radio and MCU externally, but rather use the module as the center of the ground star, with the connection between domains handled internally. Ground traces on the carrier board should run from the module to the various other connections on the board directly - preventing ground loops. As this leaves the PCB space under the module free for other routing and does not require traces that connect to multiple pins on the module - which tends to naturally form loops around the module - we reduce the difficulty of meeting routing constraints for other signals.

In addition, we found that with carrier boards designed for the Epic, the analogue supply was sometimes directly connected to the digital supply externally[5] or that the filtering circuitry for the analogue supply increased the complexity of the carrier. To avoid placing an expertise requirement on users of the platform while also ensuring a clean signal, the analog rail is derived from the single supply and filtered on the module. It is not exported externally, preventing noise from coupling onto the analog domain.

Traces to the ADC pins should be routed differentially and utilize the differential ADC capabilities of the MCU thereby providing accurate analogue readings without requiring careful external routing. For more advanced carriers that have an external analogue power domain, the module exports positive and negative analogue voltage reference inputs so that the MCU can be configured for single-ended analogue-to-digital conversion.

5.3 Micro benchmarks

While the platform is new, benchmarking is ongoing, and the TinyOS support is nascent, we are still in a position to address one of the primary questions that the platform is designed to answer: is it possible to gain all the features of a modern 32 bit microprocessor while fitting within the energy constraints associated with a sensor network node? We present a preliminary answer to this question in the form of two characterization experiments: idle current consumption and active current consumption. We also provide one data point towards exploring the effect of a 32 bit architecture on code size.

[5]such as on the Irene, the Epic Interface A, or the Common Sense badge

Table 7: Idle power comparison between a TelosB and a Storm

Voltage	TelosB μA	Storm μA
3.300	8.8	21.0
3.000	7.1	13.8
2.700	5.7	7.2
2.400	-	3.8
2.100	-	2.6
1.800	-	2.3

To characterize the first metric, we measure the current draw of the mote while running the TinyOS Null application. As the Storm platform is designed to run at low voltages (despite being capable of running at 3.3V) the experiment is performed at multiple points across its operating range. Note that the TelosB is not measured below 2.7V as the device is no longer fully functional below that point – the MSP430 cannot self-write and the external flash cannot be read. In contrast, the Storm platform retains full functionality down to 1.8V. The results are presented in Table 7.

To measure active power characteristics, we measure the power consumption of the device while it is running a computation of the sum of squares over a set of samples. This calculation is useful because it represents a realistic use case: if the energy of aggregating samples at the sensor is less than that of transmitting the full set of samples over the network for analysis, then aggregation should be performed. To further expand the test to cover energy per unit computation instead of just energy per unit time, the runtime of the task was measured externally.

In the interests of comparison with platforms running at 4 Mhz, such as the TelosB, both 4Mhz configurations and 48Mhz configurations were tested.

The code for the task used can be found in Listing 1, with the results of the experiment presented in Table 8. It can be seen that the answer to the question posed is a resounding yes: it is possible for a modern 32 bit processor to be more energy efficient than the currently used ultra-low-power 16 bit processors. In fact, at the configuration that is likely to be most common (3.3V supply, 48Mhz with external inductor) the energy consumed for the operation is just 17% of that consumed by the TelosB, the current best-in-class low energy research platform.

As a preliminary validation of functionality and measure-

Table 8: Benchmark power comparison results. The bold line indicates the anticipated common configuration

Device	Supply (V)	CFG	Freq (Mhz)	Run time (μS)	Current (mA)	Energy (μJ)	Percentage
TelosB	3.3	-	4	712.7	2.29	5.386	100%
Storm	3.3	LDO	4	393.0	1.049	1.360	25%
Storm	3.3	BUCK	4	393.5	0.501	0.651	12%
Storm	1.8	LDO	4	393.7	0.896	0.634	11%
Storm	3.3	LDO	48	32.8	13.625	1.479	27%
Storm	**3.3**	**BUCK**	**48**	**32.9**	**8.602**	**0.934**	**17%**
Storm	1.8	LDO	48	32.9	13.124	0.777	14%

Listing 1: Active power consumption and code efficiency test listing

```
int16_t buffer [256];
uint64_t acc;
task void workload()
{
    uint16_t i;
    acc = 0;
    for (i = 0; i < 256; i++)
        acc += ((int32_t)buffer[i]) * ((int32_t)buffer[i]);
}
```

ment of code size, a modified UDPEcho application was deployed to both TelosB and Storm platforms. Both were adjusted to periodically transmit broadcast packets in addition to their standard functionality. This application is a good test for code size as it includes RPL and BLIP, forming a comprehensive network stack. The application as compiled for the MSP430 requires 39280 bytes whereas the application compiled for Storm requires 87032 byes, more than double. We verified that the packets broadcast were received by both platforms – so the code sizes represent working images that were not broken by optimization. As the SAM4L includes more than 10x the flash of the MSP430, the implications of this increase in code size are not currently significant, but may influence over-the-air updates. We defer a full discussion of this problem and possible solutions to future work.

5.4 Example use in a cyber-physical system

One of the first consumers of the Storm platform, and a prime example of a cyber-physical system crossing multiple design points is the Personal Environmental Control System (PECS) that we are currently developing for increasing energy efficiency and occupant comfort in buildings.

The system consists of smart furniture augmented with sensors, heating capabilities and cooling capabilities, along with 802.15.4 and Bluetooth Low Energy radio connectivity. Here, the embedded wireless platform must be able to capture readings from temperature, relative humidity, occupancy, battery voltage and CO2 sensors while also performing actuation in the form of pulse width modulation on six channels. In addition to this, depending on user interaction the device may be called upon to communicate via USB or interact with a user via a color touch screen. In total, the system requires no less than 58 GPIO pins comprised of SPI, I2C, multiple UARTs, PWM and external interrupts.

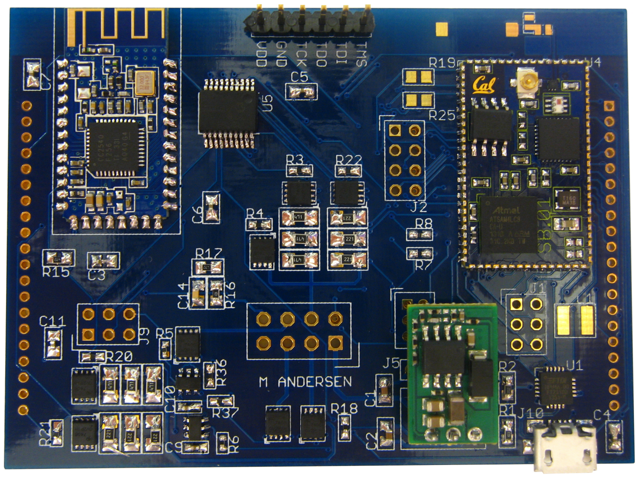
Figure 4: The PECS control board

The difficulty in this device is that it has a very wide dynamic range of requirements: it must be capable of exceptionally low idle currents while the device is not in use, but it must also be capable of the computation associated with a graphics screen. In addition, there are several states in between while the user has indicated their preferences and the system is acting upon them.

6 Opportunities and Challenges

With the first two questions posed by this paper answered, we turn to the final question: does this new category of platform introduce qualitatively new operating system challenges and opportunities? This too is answered in the affirmative, and we explore five examples of these.

6.1 Modular power management

While the SAM4L and similar microcontrollers offer features that allow for more optimal solutions to new and existing problems - such as several DMA channels, Sleepwalking and independent communication modules - these all come at a cost of increased power consumption. This energy can largely be broken into two categories - static leakage and dynamic power dissipation. While reducing static leakage largely does not fall into the domain of user concerns, it does result in processors becoming more energy efficient when operated at lower voltages. This is one of the reasons why the Storm platform was designed to have full functionality at 1.8V. The latter category of dynamic power consumption, however, requires cooperation from the user of the chip, with implications for the design of the operating system.

The approach taken by most advanced, low energy microcontrollers is to separate the functionality they provide into modules that can be individually enabled and disabled. Unfortunately this modularity at the hardware level no longer mirrors the modularity of the software system.

Although the first layer of dependencies matches that of previous generations of microcontrollers – a USART module has a set of registers corresponding to its configuration for example – every module has intricate and dynamic links to modules which have no counterpart in the high level user view of the operating system yet require information from the user. A concrete example is that of a module such as the Power Manager which is depended upon by several other modules in the system but only if the user application intends on reconfiguring the modules at a later state. If the configuration is fixed, then the module can be disabled, saving energy. To revert this decision requires restarting the chip.

In addition to userspace dependencies on device-specific “hidden” components, peripherals have dependencies on parts of the chip being serviced by different components. For instance the peripheral event control system that enables the Sleepwalking functionality interacts with almost every peripheral on the device.

While the methods in the past have avoided obtrusive inter-module dependencies by observing the underlying configuration registers, this method is no longer tractable: the time and energy spent by the system observing such configuration is $E \propto M \times S$ where E is the total energy, M is the number of modules, and S is the typical state space that needs to be observed to configure a given module. Unfortunately, due to energy saving techniques and the clock distribution structure discussed in Section 6.2, S is not constant, but rather proportional to M. This renders $E \precsim M^2$. As microprocessors become more complex, this register observation technique will not scale. Considering that the architecture of TinyOS and nesC lends itself to static analysis at compile time, there is an opportunity for resolving this problem by increasing the intelligence of the tools.

6.2 Multiple clock domains

The fine-grained module enable flags, while effective, are not sufficient for ultra-low power operation as although a module that is not logically performing any action - such as incrementing a counter or driving an IO pin - consumes less energy, there are still losses in the module as it reads the configuration flag that is keeping it disabled on every clock cycle. To prevent this, the clock source for the module is also gated, which as a side effect reduces the energy lost to parasitic capacitances in the clock distribution circuitry between the gate and the module. Here, the chip designers must make a trade-off: the higher up in the clock distribution tree the clock is gated, the greater the energy savings, but also the coarser the power management controls. To achieve the best of both worlds, some chips - like the SAM4L - utilize several layers of clock gating and module enable/disable flags.

Most modules do not need to operate at the same speed as the CPU, however, so they often have input clock divider allowing the internal logic in the module to operate at a different speed to the input clock. This is somewhat wasteful, however, as dividing a fast clock to produce a slow one uses more energy than simply using a slow clock.

To mitigate this, modules are separated into clock domains that can have different clock sources. A clock domain running at, say, 12 Mhz with the modules operating with a divisor of 1 consumes less energy than a clock domain running at 48 Mhz and the modules within operating with a clock divisor of 4, despite the apparent frequency being the same. To further expand the range of choices, there are several clock sources that are identical in frequency, but have differing accuracies, start-up times and power consumptions.

This arrangement poses two challenges for clock management in an embedded operating system. The first is that the per-module divisors are typically quite limited in their range so, depending on which modules are enabled, the clock domain frequency can only be adjusted within specific ranges while keeping the behavior of all the peripherals in that domain constant. The second is that the high level concept of a clock source now needs to encapsulate factors such as start-up time and accuracy. To further complicate matters, some clocks configurations are “soft” constraints: a synchronous SPI transfer for instance can change frequency arbitrarily without effect as long as the clock remains within the range tolerable by the devices on the bus.

The net effect of these changes to clock management is that, given more than one way of achieving an application level goal, the most energy efficient way of achieving that goal is dependent on *other unrelated parts of the application*. A simple example is that of completing an SPI DMA transfer. If no fast clock is currently required by other parts of the application, the transfer will use the least amount of energy if fed from the 115 Khz system RC clock that is always operational. If, however, an unrelated part of the application is using something that requires the DFLL or the PLL to be active, then the static costs of that clock source have already been paid and the net energy cost of the SPI transfer will be lower if fed from that clock source.

It is important to realize that the module requesting the transfer cannot simply observe the clock configuration registers to determine which clock to use. Over and above the energy usage being proportionate to a growing state space problem mentioned earlier, a consumer of a clock, it needs to know for how long that clock will be available, or obtain a strong reference to the clock that will prevent it from being disabled. The former is not observable from the registers so forms an inter-module dependency. The latter could lead to a priority inversion problem, where an unimportant task opportunistically locks a high power clock with transient dependents, preventing it from being disabled.

While this phenomenon has been present to some extent in the microprocessors typically used for sensor networks they have always posed a tractable problem because the module dependencies have been fairly contained. As the number of peripherals on a chip grow, however, and the total state space of the microcontroller expands, there arises a need for new methods of tacking power management and policy. While this paper does not propose a definitive solution, we

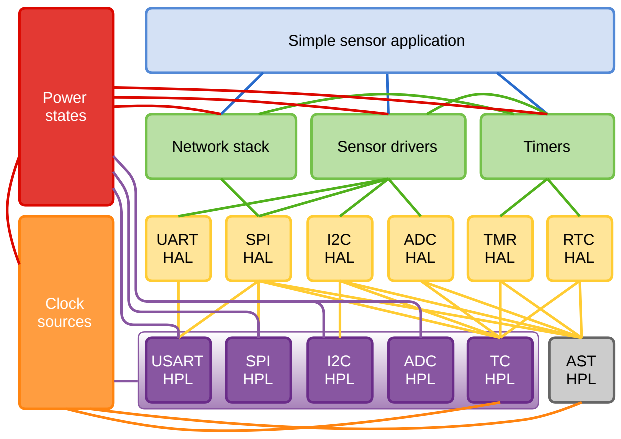

Figure 5: Complex inter-module dependencies introduced by power management and shared clock domains

will briefly explore the problem.

6.3 Inter-module compatibility

Code design for embedded systems often emphasizes modularity and separation of concerns. TinyOS hardware abstraction layering best practices, as laid out in TEP 2, suggest that the peripherals and modules present in a platform be separated with as fine a granularity as possible at the lowest level - the Hardware Presentation Layer (HPL). This implies, for instance, that the HPL code for a UART would be independent from the HPL code for a Timer or an ADC module. The various parts of the system should remain isolated and may even be implemented by different authors.

On top of the HPL there is the Hardware Adaptation Layer and the Hardware Interface Layer. It is even more true at these layers that dependencies between modules should not bleed through. A typical HIL component provides a clean, user friendly interface to a piece of device functionality, even going as far as to software emulate missing functionality in order to make the user's experience better.

The clear benefit of such an architecture is that it makes it possible to develop very complex platform-independent services and applications that are not only capable of targeting the multitude of currently available platforms, but also future platforms – as evidenced by the intricate RPL and BLIP stack that Storm supports even its infancy.

Unfortunately, this modularity is at odds with the global view required to reach optimal energy states and maintaining this isolation between modules while concurrently fully utilizing the available low-energy features is going to become more difficult as processors evolve.

As an example, Figure 5 illustrates the TinyOS stack for an application that connects the network, several sensors and some Timers on a Cortex-M4 based platform. The first thing to note is the high degree of cross-dependencies between the HAL layer and the HPL layer. In the absence of a complete overhaul to the way dependencies are managed in TinyOS, the only way for a HAL layer to guarantee that the dependencies of its module are met is to interact with the HPL components of everything that the module depends on. While these dependencies do not leak through to the user, they are still considered harmful as it means that HAL code is not portable, even between processors in the same family – the addition of a single new timer requires altering all modules that could potentially utilize that timer as a clock source.

The second point to note is that, apart from the ASynchronous Timer (AST), the other peripherals are in the same clock domain. This means that the configuration of the clock domain source must be calculated as a group, using the constraints imposed by all the modules and their consumers.

The third point to note is that the power management requires high-level knowledge from system services in order to calculate an efficient power state. To see why this is so, consider the current method for ensuring low-energy operation, which is to compute the lowest power state achievable whenever the result is deemed likely to have changed. This is the approach suggested in TEP 112. In order to approach optimality, the power state calculation must be aware of not just the current configuration of modules in the system but also of the intent of components using those modules, something not derivable from configuration registers.

As an example of this, significant energy could be saved by switching an ongoing SPI DMA operation to the 115Khz system clock as the processor core can reach its full idle state and maintain the DMA using Sleepwalking - essentially reducing the energy consumption to the bare minimum and allowing the SPI clock domain to automatically shut down upon completion of the DMA operation without processor intervention. Such a useful optimization can only occur if the power policy calculation is aware of application level information such as if the DMA transfer had a deadline or not. The option to relegate the decision to the user is also not easily selected, as the slower clock SPI DMA is only more energy efficient if there are no other modules demanding that a higher speed oscillator be kept running. This would mean that the isolated SPI consuming component would need global view about the constraints of other components in the system in order to make an effective decision.

6.4 Chaining multiple background transfers

The Sleepwalking feature discussed in Section 3 has significant implications for the way in which common tasks are implemented. Several of the tasks that previously required coordination from the processor can now happen while the processor is in a very low power mode – the same power mode that was characterized at 2.5 μA idle current in Section 5.3.

Apart from introducing resource dependency and power management complexities similar to those already discussed, this has consequences for how embedded operating systems export functionality to users. One such implication is that there are now several more ways of accomplishing the same goals, and the trade-offs between them are dynamic, changing at run-time based on factors that are outside the visibility of a given component. In fact, it is common for the most energy efficient means of performing a task to change while the task is still in operation. For example, if no other tasks are pending and the current executing task can be implemented

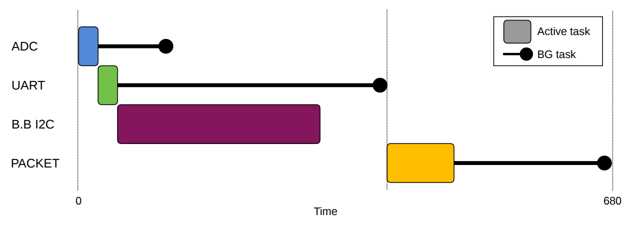

Figure 6: A task schedule with packet transmission following data acquisition

as a peripheral event chain, then spending the energy to move peripherals to low power clocks and going to sleep becomes the most energy efficient option.

One possible way of abstracting this complexity from users is to become more declarative rather than imperative. This would allow the underlying system to choose the best method of actualizing a given application intent. While declarative languages in an embedded space have been explored [20], these have not addressed such paradigms in the context of a fully generic language on hardware that intrinsically lends itself to runtime resolution of intent into implementation.

A subset of the opportunities and challenges posed by background transfers can be seen in the difference in total energy consumption due to task scheduling – the existence of 16 independent DMA channels and comprehensive peripheral DMA support means that many of the common tasks done by a sensing platform can be done in parallel or in the background.

As an example, consider a typical sensor that periodically wakes up, reads sensor values and sends the sensor data over the radio. There are sufficient DMA channels that the UART, ADC and radio can be done primarily in the background, requiring only a few cycles to begin the transfer. There may also be a sensor that requires bit banging (say a standards non-compliant I2C temperature sensor) which utilizes busy waits so can not be read over DMA.

The first pass at a system might be to post tasks that read the three sensors, and then pack and send the packet with the sensor data as illustrated in Figure 6. This is suboptimal because the UART takes long enough to complete that the processor needs to go to sleep and re-awaken in order to process the packet.

One possible optimization would be to recognize that the packet transmission could occur at the start of the next wake up period, thereby cutting out a sleep and wakeup. This is illustrated in Figure 7. Note that, depending on the scheduler, tasks may be randomly dispatched making it possible for the bit-bang task to be scheduled before the other two sensors, as shown.

This ordering is not guaranteed to yield a total energy consumption less than in Figure 6, though, as although the UART transmission occurs while the processor is in a low power state, the communication peripheral clock domain must now be powered over a longer period of time that it was in the first scenario.

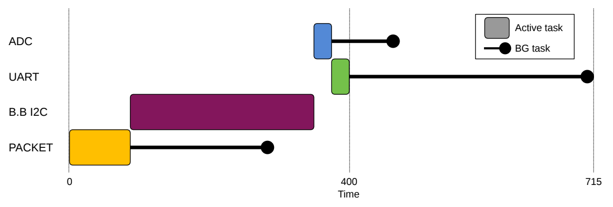

Figure 7: A naïve task schedule with packet transmission preceding data acquisition

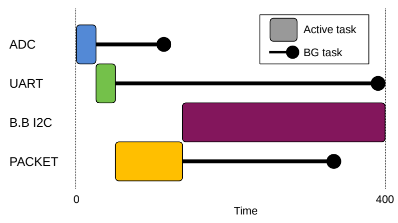

Figure 8: A carefully ordered schedule that minimizes time awake

Inspection of the tasks, however, will see that they are sensitive to dispatch order. Merely rearranging the tasks, as in Figure 8, so that the DMA operations are begun before the foreground bit-bang IO yields a total active time identical to Figure 7 but with no active DMA operations or clock domains at the end.

While task scheduling is certainly not a new problem, discovering ways of hiding the intricacies of the solution from the user will certainly play a bigger part in embedded operating systems to come. To understand why this is an interesting problem, consider how the user should ensure that task scheduling occurs in an optimal order. Without providing the scheduler with additional information about task deadlines, resource usage and estimation completion times, the user is left with a solution that would require modifying each individual component that is posting tasks and adding a manual dispatch of the next task upon its completion. This dependency between unrelated components contradicts the principles of modularity.

The challenges posed here can be summarized as an open question: how can global-view-requiring energy-optimal operation be achieved in modern microcontrollers without introducing inter-component dependencies that compromise the layering and modularity principles of embedded operating systems?

6.5 Supervisory control

Many Cortex-M4 microcontrollers feature Memory Protection Units. This allows the processor to restrict access to up to eight individual variable-sized blocks of memory so that they are only modifiable by the processor running in supervisory mode.

This has interesting implications for motes attempting to implement reliable over-the-air programming. While there are several options for program image dissemination that ensure that program images are not corrupt, the actual problem of ensuring that the program image is *logically correct* remains difficult. This is largely because an incorrect program can enter an infinite loop, preventing any future over-the-air updates and potentially preventing the code that is supposed to load a golden image from running.

The ability to protect arbitrary memory ranges, however, allows supervisory code to create a software-defined grenade timer that triggers a non-maskable-interrupt into a privileged subroutine that could, for example, ensure that the payload program image was maintaining radio contact with the mesh.

In addition to the MPU, the Flash Patch and Breakpoint unit allows the supervisory code to intercept the read access of a given flash location and replace the result with one stored in RAM. This facilitates the dynamic redirection of function calls (among other things), allowing for more flexible instrumentation of payload code. For example the grenade timer could be reset upon receipt of a certain type of packet or a certain code path in the payload.

The key change here is that a malicious or logically incorrect payload is unable to bypass the grenade timer by direct manipulation of control registers – something that was possible with watchdog timers in previous generations of embedded wireless platforms.

7 Conclusion

This paper outlines and addresses three questions posed by the technological advancements of the past decade, showing that:

1. It is indeed possible to construct a powerful embedded wireless platform based on a 32-bit microprocessor while meeting the energy profile of the current best-in-class ultra-low-power "mote". This is shown by the presentation and preliminary evaluation of Storm – one such system with a rich feature set, and best-in-class energy characteristics.

2. It is possible for the platform to simultaneously cater to the needs of both *wireless monitoring* and *cyber-physical systems*. We demonstrate the design principles enabling this development, showing that a solder-on-module with selectively exported IO is sufficiently versatile to meet a diversity of design points.

3. The use of feature-rich 32-bit processors brings with it new challenges and opportunities for operating system design. Principally we explored the issues of power management, clock distribution, modularity, background operations and supervisory control.

8 Acknowledgments

This material is based upon work supported by the National Science Foundation under grants CPS-1239552, and the NSF Graduate Research Fellowship Program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

9 References

[1] TinyOS main code repository. `https://github.com/tinyos/tinyos-main`.

[2] Freescale Kinetic KW2x MCUs. `http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=KW2x`, 2014.

[3] Linear technology LTC3388. `http://www.linear.com/product/LTC3388`, 2014.

[4] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2Nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pages 1–6, New York, NY, USA, 2010. ACM.

[5] H. Alemdar and C. Ersoy. Wireless sensor networks for healthcare: A survey. *Computer Networks*, 54(15):2688 – 2710, 2010.

[6] ARM Holdings. Cortex-M Series. `http://www.arm.com/products/processors/cortex-m/`.

[7] ARM Holdings. Arm processor licensees. `http://www.arm.com/products/processors/licensees.php`, 4 2014.

[8] M. Bordignon, E. Pagello, and A. Saffiotti. An inexpensive, off-the-shelf platform for networked embedded robotics. In *Proceedings of the 1st International Conference on Robot Communication and Coordination*, RoboComm '07, pages 45:1–45:4, Piscataway, NJ, USA, 2007. IEEE Press.

[9] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler. A building block approach to sensornet systems. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 267–280, New York, NY, USA, 2008. ACM.

[10] G. Fortino, A. Guerrieri, G. O'Hare, and A. Ruzzelli. A flexible building management framework based on wireless sensor and actuator networks. *Journal of Network and Computer Applications*, 35(6):1934 – 1952, 2012.

[11] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12–24, Nov 2002.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, pages 93–104, New York, NY, USA, 2000. ACM.

[13] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1mm3 die-stacked sensing platform with optical communication and multi-modal energy harvesting. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 402–404, Feb 2012.

[14] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[15] J. D. McLurkin. *Algorithms for distributed sensor networks*. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California, 1999.

[16] L. Nachman, J. Huang, J. Shahabdeen, R. Adler, and R. Kling. Imote2: Serious computation at the edge. In *Wireless Communications and Mobile Computing Conference, 2008. IWCMC '08. International*, pages 1118–1123, Aug 2008.

[17] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369, April 2005.

[18] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA, 2010. ACM.

[19] V. Shnayder, B.-r. Chen, K. Lorincz, T. R. F. Jones, and M. Welsh. Sensor networks for medical care. In *SenSys*, volume 5, pages 314–314, 2005.

[20] Z. Wan, W. Taha, and P. Hudak. Event-driven frp. In S. Krishnamurthi and C. Ramakrishnan, editors, *Practical Aspects of Declarative Languages*, volume 2257 of *Lecture Notes in Computer Science*, pages 155–172. Springer Berlin Heidelberg, 2002.