Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone)

JeongGil Ko[1], Kevin Klues[2], Christian Richter[3], Wanja Hofer[4], Branislav Kusy[3], Michael Bruenig[3], Thomas Schmid[5], Qiang Wang[6], Prabal Dutta[7], and Andreas Terzis[1]

[1] Department of Computer Science, Johns Hopkins University
[2] Computer Science Division, University of California, Berkeley
[3] Australian Commonwealth Scientific and Research Organization (CSIRO)
[4] Department of Computer Science, Friedrich–Alexander University Erlangen–Nuremberg
[5] Department Computer Science, University of Utah
[6] Department of Control Science and Engineering, Harbin Institute of Technology
[7] Division of Computer Science and Engineering, University of Michigan, Ann Arbor

Abstract. Some have argued that the dichotomy between high-performance operation and low resource utilization is false – an artifact that will soon succumb to Moore's Law and careful engineering. If such claims prove to be true, then the traditional 8/16- vs. 32-bit power-performance tradeoffs become irrelevant, at least for some low-power embedded systems. We explore the veracity of this thesis using the 32-bit ARM Cortex-M3 microprocessor and find quite substantial progress but not deliverance. The Cortex-M3, compared to 8/16-bit microcontrollers, reduces latency and energy consumption for computationally intensive tasks as well as achieves near parity on code density. However, it still incurs a $\sim 2\times$ overhead in power draw for "traditional" *sense-store-send-sleep* applications. These results suggest that while 32-bit processors are not yet ready for applications with very tight power requirements, they are poised for adoption everywhere else. Moore's Law may yet prevail.

1 Introduction

The desire to operate unattended for long periods of time has been a driving force in designing wireless sensor networks (WSNs) over the past decade. By focusing primarily on this requirement, platform designers have usually sought out those hardware components with the lowest sleep and active power draws [10,26]. Modern platforms, for example, use low-power microcontrollers (MCUs) like the Atmel ATmega128 [5] and TI MSP430 [37]. These devices draw mW of power while active and μW when sleeping, trading off low power for limited memory and slower clock frequencies. In turn, the limited processing and memory resources of these platforms restrict the applications they can support. Typical applications follow a *sense-store-send-sleep* archetype where on-board sensors are infrequently sampled and measurements are delivered to gateways over single- or multi-hop paths. Unfortunately, computationally intensive and higher-data-rate applications are not well supported on these platforms.

Applications that involve high-performance or high-resolution signal processing, such as structural monitoring [39], habitat monitoring [2], and motion analysis [20],

G.P. Picco and W. Heinzelman (Eds.): EWSN 2012, LNCS 7158, pp. 98–114, 2012.

typically have to make compromises to work around platform limitations. These applications tend to be characterized by alternating intervals of intense activity followed by periods of relative inactivity. In traditional deployments, designers are forced to make compromises related to the type of processing that nodes can perform, the responsiveness of the overall system, and in the placement of communications and computation elements [25] within the system. The only alternative is to use platforms that sacrifice power for processing speed [32], or platforms that combine low-power MCUs with either high-performance but energy-hungry CPUs [23] or analog processing [28].

In this paper, we examine whether modern 32-bit processors can rival their 8/16-bit counterparts in terms of both power and performance. We explore this question in the context of today's leading 32-bit embedded architecture – the ARM Cortex-M3 MCU [29] – and find quite substantial progress when compared with a leading 16-bit embedded architecture – the TI MSP430 MCU [37]. We ground our study in two new WSN platforms based on the ARM Cortex-M3: *Egs* and *Opal*. Both platforms use the Atmel SAM3U variant of the Cortex-M3 because of its low-power states, fast wakeup support, and integrated memory protection [6]. Egs targets wireless health applications while Opal targets robust environmental monitoring. Both target applications demand higher performance than motes based on 16-bit MCU provide but also require lower energy consumption than conventional 32-bit systems.

Our results show that Cortex-M3-based platforms can offer significantly better performance and power profiles for next-generation WSN applications with a reasonable increase in power draw for traditional *sense-store-and-sleep* applications. Applications with heavy workload requirements benefit from an order-of-magnitude faster processing, with the potential for further speedups when frequency scaling is used. Additionally, the Cortex-M3 provides a serial interface that does not limit the end-to-end throughput of the network. These platforms also achieve good code density due to the efficient ARM Thumb-2 instruction set, but require between 1.1 and 3 times greater RAM space, which in turn slightly increases current draw during sleep. In terms of power, the higher-powered sleep mode on the Cortex-M3 only translates to a $\sim 2\times$ whole-system power draw while running a typical workload for *sense-store-send-sleep*-style applications under TinyOS.

These results show that the dichotomy of low power or high performance is, for many WSN applications, now a nearly false one. Freed from the limitations of 16-bit operation, system developers can now focus on higher-performance signal processing, sensor-hosted databases, and spec-compatible JVMs instead of how to squeeze the last byte or cycle out of their 16-bit program.

The rest of this paper is structured as follows. In Section 2 we introduce the target applications for the two platforms presented in Section 3. Section 4 discusses about our efforts to provide TinyOS support to the Cortex-M3 and Section 5 presents our evaluations using various applications. Finally, we conclude the paper in Section 6.

2 Application-Driven Platform Requirements

As application developers venture into new computation-intensive domains, they find existing 8/16-bit mote platforms inadequate. We present applications from two different

domains that require high performance during some periods of time while maintaining overall low-power operation and derive a set of requirements that platforms supporting this new class of applications should provide.

2.1 Emerging Wireless Sensing Applications

(1) Wireless Healthcare: Wireless sensing platforms are being increasingly employed for healthcare applications [17,22,38]. Such applications are a sub-category of the emerging class of *mobiscopes*, which federate distributed mobile sensors into taskable systems [1]. In this respect, they inherit the requirements for mobility support and low-power operation, but combine them with network quality-of-service requirements.

(2) Persistent Environmental Monitoring: Government agencies are increasingly adopting wireless sensors for persistent monitoring of complex environmental systems. As an example, water catchments collect water from streams and rivers for later refinement and prevention of excessive flooding and droughts. Catchments encompass a number of inter-dependent systems: water eco-systems with aquatic life, adjoining marsh lands with terrestrial life, and agricultural pastures with cattle. Therefore, mote platforms need to support a variety of tasks, which include low-power sensing of water column temperatures, both low- and high-power micro-climate sensing, mobile wildlife and cattle monitoring and control, and processing of audio/video streams that track the biodiversity of the entire ecosystem.

2.2 Platform Requirements

(1) Sufficient Computing Resources: Adequate computing resources are required to process physiological data in near real-time. Similarly, local processing of A/V streams can reduce high-bandwidth multimedia data to a set of well selected classifiers [21,34].

(2) Low Energy Consumption: Mobiscopes used in medical applications should consume little energy to allow uninterrupted operation for multiple days [17]. Aggressive energy management and advanced battery charging circuitry are needed to assure uninterrupted operation using solar-harvested power in outdoor deployments.

(3) Sensor Support and Extendability: Given the diversity of healthcare, environmental, and climate sensors, platforms should be able to provide developers with the freedom to extend their sensing capabilities with various sensors.

(4) Storage: When wireless connectivity is intermittent [17] or bandwidth is inadequate to deliver all measurements [20], the platform should have enough non-volatile storage to preserve the collected data.

(5) Security: According to U.S. and international laws, physiological data must be secured to protect patients' personal information [17].

(6) User Interface: Medical staff are responsible for a large number of patients and are chronically overworked [8]. Previous deployment experiences suggest that wireless monitors should include LCDs that display measurements to reduce staff workload [17].

(7) Network Connectivity and Reliability: Multiple radio interfaces simplify user access to the collected data – for example, through Bluetooth-enabled mobile phones. Robustness and reliability of radio communication between sensor nodes can be improved if the radios exhibit diverse propagation characteristics.

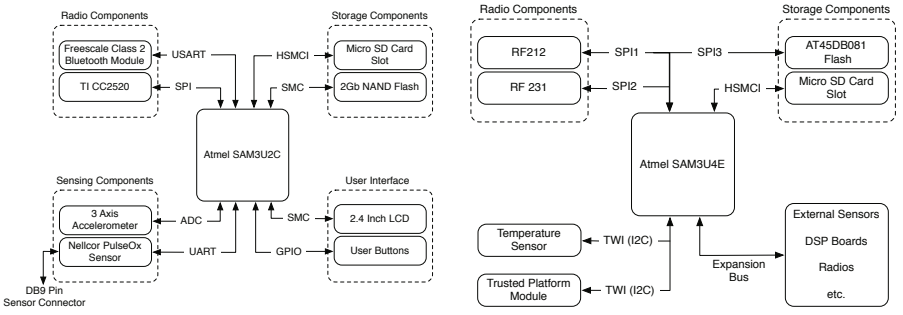
3 ARM Cortex-M3 Platforms: *Egs* and *Opal*

In the last decade, low-power mote platforms were mostly based on 8- or 16-bit MCUs. The common assumption was that such MCUs were adequate and that moving to 32-bit MCUs would be too costly in terms of energy consumption or device cost. Introducing the Telos family of motes [26] was a big leap in technology advances. Features like direct memory access (DMA) allowed high-speed sampling or long MCU sleep times, as the DMA controller could handle data sampling in the background while the core was kept asleep.

At the same time, several WSN platforms with 32-bit MCUs have been proposed to support applications that require higher processing power [9,27,35]. None of those, however, can match the current draw of 16-bit MCUs in sleep or suspend mode. A common solution is to use an 8- or 16-bit MCU as a power manager or pre-processor [23]. In that model, the higher-power 32-bit MCU performs high-speed calculations and is turned off by the low-power MCU when not needed. While this duty cycling achieves reasonable energy performance, the multi-MCU nature of these devices makes them more complex, expensive, and leads to additional leakage currents.

With the introduction of the ARMv7 architecture, ARM and its silicon partners claim that their 32-bit MCUs can compete with 8- and 16-bit microcontrollers not just in performance, but also in *energy efficiency*. The ARM Cortex-M series consists of the Cortex-M0/M1 (ARMv6-M architectures), Cortex-M3, and, more recently, the Cortex-M4. The Cortex-M3 is currently the most widely used core of the Cortex-M family by silicon vendors. It provides an excellent balance of energy efficiency (12.25 DMIPS/mW on TSMC 90 nm G [3]) and calculation performance. It also supports single-cycle 32x32 multiplications and has a flexible interrupt controller supporting up to 256 priority levels. Additionally, silicon vendors can add a memory protection unit (MPU), whose functionality and interface are defined by ARM.

To address the challenges introduced by the applications outlined in Section 2, we now present two platforms designed around the Cortex-M3-based Atmel SAM3U MCU. Among the various Cortex-M3 MCUs, the SAM3U strikes a good balance between performance (clock speed up to 96 MHz), storage capabilities (NAND flash interface), peripherals (USB, High-Speed MCI, USARTs, etc.), and power efficiency. Its largest variant (SAM3U4E) provides 256 KB of flash, 52 KB of SRAM, and can wake up from its lowest power state in $< 10\ \mu$s, while drawing as low as 8.9 μA. Two types of ADCs, a high-speed 12-bit, and a low-power 10-bit ADC provide the functionality necessary for a modern sensing system and the USB interface provides a high-speed serial link to a computer. Finally, the SAM3U's MPU rounds up the complete SAM3U package and was one of the key deciding factors when choosing an MCU for the platforms presented below.



(a) Egs (b) Opal

Fig. 1. Functional diagram of the Egs and Opal platforms

3.1 Egs

Egs (Figure 1(a)) is our attempt to develop a wireless sensing platform for wireless healthcare applications that meets the requirements outlined in Section 2. To satisfy these requirements, Egs uses an Atmel SAM3U2C MCU as its core. The SAM3U2C, with its compact 100-pin layout, 36 KB of SRAM, and 128 KB of flash memory, provides a balance between the small form factor required for healthcare mobiscopes and the need for ample computing and memory resources. The device can be (re)programmed using a mini USB connection, which is also used to recharge its 1,200 mAh Li-ion battery.

For persistent data storage, Egs provides an on-board NAND flash memory, connected using the static memory controller (SMC). For removable memory, Egs also exposes a microSD card slot through the High-Speed Multimedia Card Interface (HSMCI). Egs also incorporates several sensors widely used in healthcare applications. Specifically, a Nellcor OEM controller provides heart rate and pulse oximetry (SpO_2) measurements, and a 3-axis accelerometer is provided for applications such as fall detection [12] and activity monitoring [38].

These sensors, however, are only a small subset of the sensors that various healthcare applications may require. Given that many physiological sensors provide Bluetooth connectivity [40], Egs incorporates a Bluetooth module that operates in master or slave mode [11]. When configured as a master, Egs can aggregate measurements from multiple sensor slaves. Configured as a slave, Egs can upload collected measurements to cell phones or laptops. To further support low-power wireless mesh connectivity, Egs also includes the TI CC2520 802.15.4 radio [36] as well.

Feedback from medical staff has consistently ranked a readable display as the most required feature for physiological monitors [17]. To satisfy this requirement, Egs includes a 240×320 color LCD screen. It displays text and graphics, including vital signs, and combined with the two user buttons, is the device's primary user interface.

3.2 Opal

To effectively support the requirements for the persistent environmental monitoring applications introduced in Section 2, we designed the Opal platform. Like Egs, Opal is built around the SAM3U4E MCU, but provides a different set of peripherals to fulfill its varying set of requirements. Figure 1(b) provides a functional diagram of the components contained in Opal.

In order to achieve high link reliability and connectivity, Opal nodes provide Multiple Input / Multiple Output (MIMO) capabilities in the form of dual 802.15.4 radios. Each radio supports switching between two separate antennae and runs at bitrates between 250 Kbps and 2 Mbps. Incorporating two radios into the system allows us to take advantage of benefits from radio diversity, as recently introduced by Kusy et al. [18].

For security, the inclusion of a Trusted Platform Module (TPM) allows Opal to support efficient secure actuation, remote attestation, and improved cryptographic computations over insecure links. This module allows for fast calculations of asymmetric cryptographic operations and Public Key Infrastructures (PKI) due to improved handling of data and keys in memory.

Additionally, Opal includes a TWI-based temperature sensor, standard flash, and microSD storage for data logging and storing program images. Opal also exposes a 120-pin expansion bus of the 144 pins on the SAM3U4E to connect various peripherals. Finally, the Opal platform enables extreme longevity of unattended sensing deployments by including a proprietary energy harvesting input that feeds into a hardware-based Li-ion battery charging circuit. This mechanism allows the MCU to be put to deep sleep while the batteries are recharged with solar or other ambient energy sources.

4 Porting TinyOS

Considering the large number of developers and resources available for the Cortex-M3, a variety of software platforms could have been used for application development. However, given the availability of high-quality software components for power management and networking in TinyOS, we decided to port TinyOS 2.x to the SAM3U MCU. The paragraphs below describe the highlights of this effort.

4.1 Memory Protection in TinyOS

As mentioned previously, one of the deciding factors to use the SAM3U was its memory protection unit (MPU). An MPU is used to isolate software components from one another by performing memory access checks in hardware. To this end, the MPU is configured with memory ranges where each component is contained, along with the appropriate access rights (e.g., read, write, execute) to those memory ranges. Unlike a memory management unit (MMU) (used in most PC-like architectures), MPUs are not able to perform address space virtualization and swapping; in other words, all data and code reside in a single physical address space.

Leveraging the TOSThreads [15] package in TinyOS, we added support for *thread isolation* on the SAM3U by configuring and re-configuring the MPU at run time to

Table 1. SAM3U current draw at different clock speeds. Measurements are taken for active and sleep modes. In wait mode, the MCU is in deep sleep.

MCU State	Active Mode (mA)	Sleep Mode (mA)
4 MHz	4.98	0.99
12 MHz	10.01	2.84
48 MHz	30.19	11.72
84 MHz	44.01	20.02
96 MHz	48.00	21.99
Wait Mode	—	0.02

constrain all threads to the minimal access permissions they need to run. This includes read and write permissions to their thread-local data, read and execute permissions to their thread code, and read and execute permissions to a common code section for system call wrappers. Since TOSThreads runs TinyOS inside a special kernel thread, and the MPU protects all threads from accessing each other's data, these system call wrappers provide a well-defined interface for user threads to invoke functions on the kernel thread. Internally, the system call switches the processor from user to supervisor mode, enabling access to kernel data and code via the kernel system call handler. Memory protection thereby isolates faults caused by buggy or malicious code by containing them in the thread that they appeared in. The benefits of containing faults in TinyOS have recently been explored by Chen et al. by using software-based isolation with a safety-aware compiler inserting run time checks for fault detection [7]. Our memory-protected TinyOS does not need a specialized compiler and can be leaner in terms of binary size, since the hardware implicitly performs run time checks.

In addition to TOSThreads, the Deluge [13] over-the-air reprogramming subsystem can also benefit greatly from the use of an MPU. Deluge utilizes a boot loader to read stored program images from flash and load them at startup. If either Deluge or its boot loader is corrupted by the mismanagement of memory, the entire network may become unusable. An MPU can be used to ensure proper isolation of Deluge and its boot loader from buggy or even malicious code.

4.2 Power Management

The power consumption of the Cortex-M3 depends on its clock speed. Different clock frequencies not only have different active power draws, but also limit which low-power states the MCU can enter.

The MCU's main clock (MCK) is driven by a different oscillator depending on the frequency used. In our TinyOS implementation, MCK frequencies $<$12 MHz use the Fast RC oscillator, while higher frequencies use the 3–20 MHz oscillator. Due to its quick response and fast stabilization, the Fast RC oscillator is the default clock source upon startup. The 3–20 MHz oscillator takes longer to stabilize but acts as a more accurate clock source.

The Atmel SAM3U defines three low-power modes [6]. The lowest is the "backup mode", which consumes as little as 2.5 μA. In this mode, the core memory peripherals

Table 2. Latency associated with entering and exiting wait and sleep modes with different main clock (MCK) sources. The latency for wait mode includes checks on the current MCK source and reconfiguring the clock source (if required). The sleep mode does not require such checks.

Start Mode	End Mode	Latency (μs)
Any Clock Source	Sleep	1
Fast RC Oscillator	Wait	4
3–20 MHz Oscillator	Wait	114
Sleep	Any Clock Source	1
Wait	Fast RC Oscillator	4
Wait	3–20 MHz Oscillator	3867

are not powered, the voltage regulator is turned off, and the core resets on wakeup; therefore, the practical use of this mode is limited. The MCU can also be put in "wait mode", where we were able to measure a power consumption of $\sim$19 μA with a wakeup time of $\sim$4 μs. In this mode, I/O states are kept unchanged and the core is restored to the previous state on wakeup. The third low-power mode is the "sleep mode", which is used when a very short response time ($\sim 1\mu$s) is required and which only puts the core to sleep. In sleep mode, the power consumption depends on the MCK speed (Table 1).

Note that the SAM3U can only enter the wait mode when the MCK is driven by the Fast RC oscillator. Thus, when running at high frequencies, the MCK source needs to first switch to the Fast RC oscillator. Conversely, when exiting wait mode, the MCK source needs to switch back from the Fast RC oscillator to the 3–20 MHz oscillator. Table 2 presents these latencies, measured using our port of TinyOS to the SAM3U. Entering the wait mode while operating on a frequency that already uses the Fast RC oscillator introduces only a minimal amount of latency. On the other hand, when the 3–20 MHz oscillator sources the MCK in active mode, the latency to enter and exit the wait mode increases. While the latency required to enter the wait mode is still minimal (i.e., 114 μs to stabilize the Fast RC oscillator), the latency introduced when exiting the wait mode is relatively higher (i.e., 3867 μs). While this value is tolerable for most wireless sensing applications, some may desire a quicker response. These systems can potentially wake up quickly using the fast RC oscillator and move to a higher frequency by starting the 3–20 MHz oscillator in parallel for faster processing speeds.

5 Evaluation

We present several sample applications using our Egs and Opal platforms and compare their performance to the TelosB platform using the TinyOS software stack. Latency and per-task energy consumption are the primary metrics for evaluation.

5.1 Compression

High-frequency sampling applications (e.g., ECG data monitoring [17], structural monitoring [39]) can use compression to reduce the number of bytes and amount of energy

Table 3. 100-byte payload compression/decompression latency and energy consumption for Egs with five different MCK speeds and TelosB with S-LZW

Configuration	Compression Latency	Energy Consumption	Decompression Latency	Energy Consumption
TelosB (4 MHz)	10.405 ms	62.87 μJ	5.265 ms	31.81 μJ
Egs (4 MHz)	3.472 ms	160.29 μJ	1.323 ms	61.08 μJ
Egs (12 MHz)	1.125 ms	76.44 μJ	0.430 ms	29.22 μJ
Egs (48 MHz)	0.302 ms	41.95 μJ	0.106 ms	14.72 μJ
Egs (84 MHz)	0.167 ms	33.80 μJ	0.068 ms	13.69 μJ
Egs (96 MHz)	0.147 ms	32.64 μJ	0.053 ms	11.76 μJ

they expend for data delivery [30]. Unfortunately, compression introduces latency that can lead to longer duty cycles. Moreover, memory buffering during compression can act as a bottleneck for the whole system.

To quantify the benefits of using a Cortex-M3, we compare TelosB to Egs running the S-LZW compression algorithm, proposed specifically for resource-constrained motes [30]. Table 3 presents the latency and the respective energy consumption to compress and decompress 100 bytes. Specifically, we measured Egs running at 4, 12, 48, 84, and 96 MHz. Our results indicate that Egs' latencies are significantly shorter than TelosB's, even at the same frequency (4 MHz). The explanation for this speedup is the reduced number of memory accesses and operations of the 32-bit processor compared to the 16-bit MSP430 MCU. As an example, while the SAM3U and MSP430 MCUs execute the exact same compression task, the number of `MOV` instructions in the assembly trace was 295 and 481 for the SAM3U and MSP430, respectively.

Since a device stays active during both the compression and decompression tasks, latency directly relates to energy consumption. While Egs consumes more energy than TelosB when active, Table 3 shows that the lower latency leads to comparable or lower energy consumption. By accomplishing the task sooner, Egs can transition to a low-power mode more quickly. Furthermore, the results in Table 3 demonstrate that for the Cortex-M3-based platforms, compression/decompression time is lower than the time necessary to transmit the same packet over an IEEE 802.15.4 wireless link ($\sim$ 3 ms for a 100-Byte packet using a 250Kbps radio), thereby removing a potential bottleneck.

5.2 Error Correction Codes

It is important to protect packets from external interference that is prevalent in the 2.4 GHz frequency range. For example, in applications operating in clinical environments, it is important to employ a scheme that helps the system overcome the irregular channel noise introduced in such environments [17]. Liang et al. recently proposed using Reed-Solomon (RS) codes to protect 802.15.4 packets from interference caused by nearby WiFi networks [19]. RS codes are block-based, linear error correction codes that can recover data corruptions and erasures by dividing a message into multiple blocks of user-defined size and computing extra parity blocks attached to the original payload.

Table 4. Processing speed and energy consumption measurements for Reed-Solomon coding on Egs at 96 MHz. The results are normalized to the corresponding values from the TelosB.

Operation	Processing Speed	Norm. Speedup	Norm. Energy Consumption
Encode	973 pkts/sec	36 x	106.44%
Decode (No Error)	567 pkts/sec	63 x	61.25%
Decode (5B Error)	399 pkts/sec	57 x	70.23%
Decode (15B Error)	265 pkts/sec	53 x	78.83%
Decode (30B Erasure)	217 pkts/sec	54 x	87.10%

Liang et al. developed a light-weight version of RS encoding, termed *TinyRS*, which can operate on resource-constrained motes such as TelosB. However, since the RS encoding process happens directly prior to a packet transmission and the decoding process takes place directly after a packet is received, it is desirable to minimize the respective latencies. To examine how a Cortex-M3 core can improve these operations, we compute the number of packets that can be RS-encoded or -decoded in one second. We performed experiments with typical bit errors expected on a wireless channel using payloads of 65 bytes followed by 30 parity bytes.

Table 4 presents the results measured for Egs operating at 96 MHz and TelosB operating at 4 MHz (TinyOS default). Note that Egs supports RS encoding of packets at full transmission speeds of the radio, since the processing speed of Egs is faster than the speed at which the radio can transmit 100-byte packets (i.e., $\sim$3 ms), while this is not true for TelosB. For example, consider the number of packets that can be encoded per second. The Egs platform can encode 973 packets/sec, while the TelosB can encode only 27 packets/sec (i.e., Egs is 36 times faster). Similarly, when all payloads are received with 5 byte errors, the Egs platform decodes 399 packets per second, compared to only 7 packets on the TelosB. These results indicate that by using a faster MCU, RS codes can be practically used in various WSN environments. Furthermore, while encoding requires similar energy on the two platforms, thanks to the shorter latency, decoding on the Cortex-M3 is more energy-efficient than on the TelosB.

5.3 High-Speed Serial Communications

Most existing mote platforms contain radios with a maximum bandwidth of 250 Kbps. Newer radios also support speeds of up to 1 Mbps (RF212) or 2 Mbps (RF231). With these radios, platforms should be able to run high-throughput streaming applications with near real-time performance. The issue, however, is that the end-to-end throughput on current mote platforms is limited significantly by two parts: (1) the serial connection speed with the PC, and (2) intra-node packet processing delay. The ARM Cortex-M3 contains features that allow us to partially overcome these bottlenecks. First, the Cortex-M3 contains an integrated High-Speed USB 2.0 controller for fast data communication with a PC. This controller supports speeds of up to 480 Mbps (with DMA), in contrast to 115.2 Kbps as seen on many existing mote platforms. Our preliminary USB driver that does not use DMA is capable of streaming data at speeds of up to 1.8 Mbps, thereby

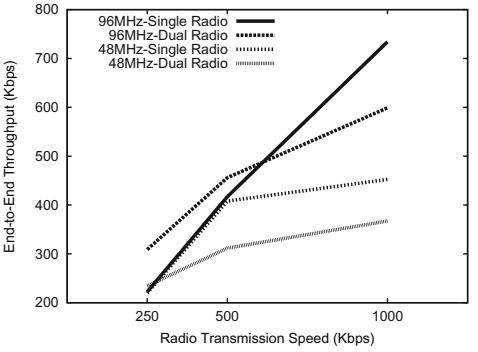


Fig. 2. Throughput of a data streaming application for different radio and MCU speeds. The application consists of a single node transmitting back-to-back packets through a basestation node connected to a PC via a USB link, using either one (RF231) or two radios (RF212, RF231) with bitrates between 0.25 and 1 Mbps on each radio. The MCU speed on both the basestation and the streaming node varies between 48 and 96 MHz.

removing the speed barrier between the mote and the PC. Second, since the Cortex-M3 can run at speeds of up to 96 MHz, intra-node processing delays are also reduced.

To demonstrate how these features help increase the maximum end-to-end throughput, we devised an experiment in which we vary the clock frequency and radio bitrates on the Opal platform. Our test environment consists of a single-hop network including a transmitting node that broadcasts packets as fast as possible through a basestation connected via USB to a PC. We vary the MCU speed on both the basestation and the transmitting node between 48 and 96 MHz and send packets using either one (RF231) or two radios (RF212 and RF231) with varying transmission bitrates between 250 Kbps and 1 Mbps; in the dual radio case, both radios are set to run with the same bitrate.

Figure 2 shows our results. For a single radio transmitting at a bitrate of 250 Kbps, we are able to sustain USB transfer rates of 222 Kbps at 96 MHz and 220 Kbps at 48 MHz. For 500 Kbps, we see transfer rates of 417 Kbps at 96 MHz and 408 Kbps at 48 MHz, indicating that the packet processing latency is no longer a bottleneck when using low bitrates with high MCU clock speeds. Instead, the throughput decrease can be attributed to (1) the delay incurred when transferring each packet over SPI from the MCU to the radio, and (2) the radio transition time as it switches from transmit mode to receive mode and back again between each packet transmission. However, as we increase the bitrate to 1 Mbps, we start to see the effects of packet processing overhead when operating at 48 MHz – at 96 MHz we achieve a throughput of 734 Kbps, while only achieving 452 Kbps at 48 MHz. For the dual-radio case, we see a similar phenomenon, except that even at 96 MHz we start to see a non-linear decrease in performance before reaching 1 Mbps. With two radios transmitting at the same bitrate, we should, ideally, be able to sustain an end-to-end throughput equal to twice the bitrate of a single radio (i.e., a USB transfer rate of 500 Kbps when both radios are configured at 250 Kbps). However, the intrinsic overheads are amplified in this case due to arbitration on the shared SPI bus between the two radios. This overhead could be alleviated in future platform revisions by giving each radio its own dedicated SPI bus.

Table 5. Memory footprints for a Java Virtual Machine and implementations of the RPL and 6LoWPAN standards in TinyOS. These components require relatively large amount of ROM and RAM compared to traditional sensor network components.

Component	ROM (B)	RAM (B)
TakaTuka JVM - Null App on Egs/Opal	24380	2527
TakaTuka JVM - Null App on TelosB	22620	902
TakaTuka JVM - Simple Periodic Radio Tx on Egs/Opal	36084	3038
TakaTuka JVM - Simple Periodic Radio Tx on TelosB	36018	1242
6LoWPAN Header Compression on Egs/Opal	7826	1960
6LoWPAN Header Compression on TelosB	7222	1581
RPL (Route Storing Edge Node) on Egs/Opal	9024	8370
RPL (Route Storing Edge Node) on TelosB	8428	7454

Overall, these results suggest that end-to-end throughput on Cortex-M3-based platforms is no longer limited by the low transfer rates between a basestation and a PC. Instead, we are now limited only by delays incurred in system software and platform architecture decisions. With further work, Cortex-M3 mote platforms may soon be able to drive end-to-end throughput to its maximum potential.

5.4 JVMs and IETF Standards in Sensor Networks

Most existing software components for WSNs are compact in size in order to operate on resource-constrained mote platforms. However, several software components for newer applications require larger amounts of RAM and ROM space.

First, we examine the case of using Java Virtual Machines (JVMs) on mote platforms. Table 5 reports the memory footprint of the open-source TakaTuka JVM [4]. The results from this table suggest that when using JVMs on resource-constrained motes, only a small amount of memory space will be left for the application. For example, if the JVM were installed on a TelosB platform, it would take up approximately 47.13% of the ROM space and with a simple radio application, the ROM consumption would increase to 75.04% of the 48 KB limit, leaving only a small amount of space for different application components (e.g., sensing modules). Another mote platform that includes a JVM is the Sun SPOT [35], operating with the Squawk JVM as its default software platform [33]. The Sun SPOT uses a 32-bit ARM920T core, can operate at frequencies of up to 180 MHz, and is equipped with various sensors and actuators. However, while Sun SPOTs can effectively support JVM-related applications, data sheet comparisons with the SAM3U indicate that the Sun SPOT's processor board's power consumption is more than twice the one of the SAM3U in the lowest-power setting.

The implementation of IETF standards for WSNs can also require a large memory space. In Table 5 we present the memory footprints for the RPL and 6LoWPAN header compression implementations in TinyOS [16]. For RPL, we configured the edge routers to maintain routes for up to 200 children. The memory footprints indicate that the 6LoWPAN header compression combined with RPL is too large to fit in a TelosB platform in its current form due to the limited RAM size (i.e., 10 KB). While one

Table 6. Energy consumption of TelosB and Opal when performing tasks of a low-duty-cycle application. Numbers for TelosB and RF231 are from [14] and the data sheet, respectively. The numbers for SAM3U4E were measured on the SAM3U-EK development board.

	ADC Read+ Flash Log	**CCA Check**	**Sleep Current**
Opal	343 (μAs)	157 (μAs)	*(SAM3U+RF231)* 40 (μA)
TelosB	22.58 (μAs)	94.3 (μAs)	9 (μA)
Ratio	15.19 x	1.66 x	4.44 x

could reduce memory usage by reducing the number of children that an edge router can support, doing so would be undesirable for large-scale deployments. Schmid et al. [31] recently proposed a network architecture in which *leaf* nodes minimize idle listening by only turning their radios on to transmit their data and in which those nodes do not listen on the wireless channel for potential incoming packets. In such cases, router nodes should buffer the packets destined to the leaf nodes until they wake up. Such message buffers would likely be larger than the available resources on current mote platforms. On the other hand, using a resource-rich Cortex-M3-based platform will allow WSNs to fully experience the benefits of using a standards-compliant networking stack.

Finally, a comparison of the memory footprints of a Cortex-M3 platform and TelosB in Table 5 suggests that the Cortex-M3 offers good code density, close to that of a 16-bit MCU, by leveraging the compact yet expressive Thumb-2 instruction set. However, as a side note, the increased RAM usage on the Cortex-M3 platform implies that its static leakage can affect the platform's current draw in sleep mode, slightly increasing its overall energy consumption.

5.5 Low-Duty-Cycle Applications

The applications presented up to now require large amounts of data processing or consume large amounts of memory. One remaining question, however, is how the Cortex-M3 stands up to low-power 8/16-bit MCUs when running *traditional*, low-duty-cycle applications with large idle times. These applications typically perform three types of operations: (1) periodic sampling and logging of sensor data, (2) periodic radio wakeups for clear channel assessment (CCA), and (3) sleeping when idle.

To evaluate the energy consumption of such an application, we designed an experiment that measures the energy consumed by each of these operations on Opal. Specifically, we measured three different values: (1) the amount of energy required to sense and log an ADC sample to flash, (2) the amount of energy needed to perform a single CCA check on the RF231 radio, and (3) the total current draw of the platform when idle (i.e., "wait mode"). Values for the idle current of the SAM3U4E were measured on the SAM3U-EK development board and values for the RF231 radio are from its data sheet. Table 3 presents all of our gathered measurements alongside similar measurements for TelosB from [14]. We chose to lower the clock frequency of SAM3U to match the one of TelosB (i.e., 4 MHz) for a fair comparison. Given power consumption at this frequency, we argue that running at this speed would be appropriate for applications focused on preserving energy. If the application required complex tasks, it could temporarily switch the MCK speed, and return to 4 MHz once the task completes.

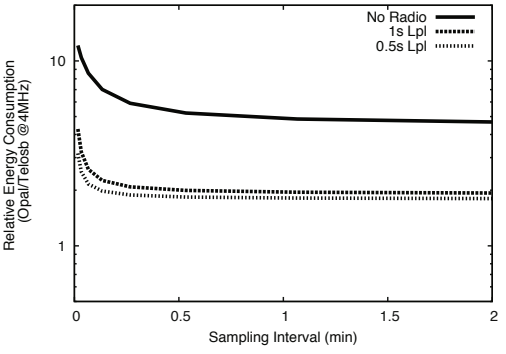


Fig. 3. Relative energy consumption of Opal vs. TelosB running a low-duty-cycle, periodic sensing application at different sampling intervals with low-power listening

Using Table 6, we extrapolate the energy consumption of both platforms as a function of sampling frequency ($f_{Sampling}$) and CCA frequency (f_{LPL}). We use equations (1) and (2) to derive the relative energy consumption of Opal vs. TelosB. $E_{Sampling}$ is the amount of energy spent for ADC sampling and E_{LPL} is the energy needed for a CCA check in LPL.

$$I_{Platform} = E_{Sampling} \times f_{Sampling} + E_{LPL} \times f_{LPL} + I_{Idle} \quad (1)$$

$$\frac{E_{Opal}}{E_{TelosB}} = \frac{I_{Opal}}{I_{TelosB}} (for\ a\ constant\ V_{CC}) \quad (2)$$

Figure 3 plots the results of applying these equations to sampling frequencies between 1 s and 2 min, with LPL CCA check intervals of 0.5 s, 1 s, and the radio disabled. Lines on the graph represent $\frac{E_{Opal}}{E_{TelosB}}$.

Given these plots, it is obvious that TelosB outperforms Opal in all cases. Specifically, when no LPL is employed, the relative energy consumed by Opal is dominated at low sampling frequencies by the ratio of energy consumed while sleeping (i.e., 4.44x), and at high sampling frequencies by the power consumed while logging (i.e., 15.19x). The more often you sample, the less often you sleep, making the logging term more dominant when sampling fast. However, with LPL, the extra energy consumed by CCA checks begins to counteract the savings gained by low sleep currents. The shorter the LPL interval, the lower the overall ratio. At low sampling frequencies and short LPL intervals, the ratios asymptotically approaches 1.66×. Specifically, by choosing a reasonable CCA check interval of 0.5 s [24] and a sampling frequency $<$ 1 sample/min, we see a relative energy consumption of only around 1.8×. While not ideal, these results indicate that it is possible for Cortex-M3 platforms to come within a factor of 2 of the energy efficiency of mote platforms designed specifically for low power.

We further attribute some overheads to our current implementation, which is at an early development stage compared to TelosB. Additional features such as DMA and well synchronized timing/clock configurations will reduce the energy consumption further while allowing the MCU to utilize its fast wakeup functionality.

6 Conclusion

We started this paper with a simple question – can modern 32-bit MCU rival their 16-bit brethren on power while beating them on performance? We end this paper with our answer – soon. While we illustrate the initial trends in this work, we anticipate many developments and improvements to take place in our software stack, reducing the energy consumption and operational latency measurements even more over time. Furthermore, from an entire system's perspective, these new MCUs largely free system developers from the performance limitations of 16-bit architectures without completely abandoning their attractive power characteristics. It is clear that this tradeoff will soon be dominated by other costs (e.g., battery leakage and wireless synchronization). With ARM committed to supplying 32-bit processors at sub-dollar price points, 32-bit architectures are poised to assume the processing mantle for future sensor network platforms.

Acknowledgments. We would like to give special thanks to Dr. Philip Levis for supporting Wanja Hofer's visit to Stanford University where he first started working on this project. Andreas Terzis and JeongGil Ko were partially supported by the National Science Foundation under grant #0855191.

References

1. Abdelzaher, T., Anokwa, Y., Boda, P., Burke, J., Estrin, D., Guibas, L., Kansal, A., Madden, S., Reich, J.: Mobiscopes for human spaces. IEEE Pervasive Computing 6, 20–29 (2007)
2. Ali, A.M., Yao, K., Collier, T.C., Taylor, C.E., Blumstein, D.T., Girod, L.: An empirical study of collaborative acoustic source localization. In: IPSN (2007)
3. ARM Holdings. Cortex-M3 Processor, `http://www.arm.com/products/processors/cortex-m/cortex-m3.php`
4. Aslam, F., Schindelhauer, C., Ernst, G., Spyra, D., Meyer, J., Zalloom, M.: Introducing takatuka: a java virtualmachine for motes. In: SenSys (2008)
5. Atmel Corporation. Atmel ATMega 128 Microcontroller Datasheet, `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`
6. Atmel Corporation. AT91 ARM Cortex-M3 based MCUs: SAM3U Specifications (2009)
7. Chen, Y., Gnawali, O., Kazandjieva, M., Levis, P., Regehr, J.: Surviving Sensor Network Software Faults. In: SOSP (2009)
8. Coalition for American Trauma Care. Action Needed to Bolster Nation's Emergency Care System (June 2006)
9. Crossbow Inc. Imote2: High-Performance Wireless Sensor Network Node (2007), `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf`
10. Crossbow Technology Inc. MICAz wireless measurement system (June 2004), `http://www.xbow.com`
11. Free2move AB. Low power Audio BluetoothTM Module with antenna F2M03ALA Datasheet (2007), `http://www.free2move.se`
12. Freescale Semiconductor. Three axis low-g micromachined accelerometer, `http://www.freescale.com`
13. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: SenSys (November 2004)

14. Klues, K., Handziski, V., Lu, C., Wolisz, A., Culler, D., Gay, D., Levis, P.: Integrating Concurrency Control and Energy Management in Device Drivers. In: SOSP (2007)
15. Klues, K., Liang, C.J., Paek, J., Musaloiu-E, R., Govindan, R., Terzis, A., Levis, P.: TOSThreads: Safe and Non-invasive Preemption in TinyOS. In: SenSys (November 2009)
16. Ko, J., Dawson-Haggerty, S., Gnawali, O., Culler, D., Terzis, A.: Evaluating the Performance of RPL and 6LoWPAN in TinyOS. In: Proceedings of the Workshop on Extending the Internet to Low Power and Lossy Networks (IP+SN 2011) (April 2011)
17. Ko, J., Lim, J., Chen, Y., Musaloiu-E., R., Terzis, A., Masson, G., Gao, T., Destler, W., Selavo, L., Dutton, R.: MEDiSN: Medical Emergency Detection in Sensor Networks. ACM Transactions on Embedded Computing Systems, TECS (2010)
18. Kusy, B., Richter, C., Hu, W., Afanasyev, M., Jurdak, R., Brunig, M., Abbott, D., Huynh, C., Ostry, D.: Radio diversity for reliable communication in wsns. In: IPSN (April 2011)
19. Liang, C.-J.M., Priyantha, N.B., Liu, J., Terzis, A.: Surviving wi-fi interference in low power zigbee networks. In: SenSys (2010)
20. Lorincz, K., Chen, B.R., Challen, G.W., Chowdhury, A.R., Patel, S., Bonato, P., Welsh, M.: Mercury: A Wearable Sensor Network Platform for High-Fidelity Motion Analysis. In: SenSys (2009)
21. Luo, L., Cao, Q., Huang, C., Abdelzaher, T., Stankovic, J.A., Ward, M.: Enviromic: Towards cooperative storage and retrieval in audio sensor networks. In: ICDCS (2007)
22. Malan, D., Fulford-Jones, T., Welsh, M., Moulton, S.: CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In: MobiSys 2004 Workshop on Applications of Mobile Embedded Systems (June 2004)
23. McIntire, D., Ho, K., Yip, B., Singh, A., Wu, W., Kaiser, W.: The low power energy aware processing (LEAP) embedded networked sensor system. In: IPSN/SPOTS (2006)
24. Moss, D., Hui, J., Klues, K.: TEP 105: Low Power Listening (2008), `http://www.tinyos.net/tinyos-2.x/doc/pdf/tep105.pdf`
25. Paek, J., Greenstein, B., Gnawali, O., Jang, K.-Y., Joki, A., Vieira, M., Hicks, J., Estrin, D., Govindan, R., Kohler, E.: The tenet architecture for tiered sensor networks. ACM Transactions on Sensor Networks (TOSN) 6(4) (2010)
26. Polastre, J., Szewczyk, R., Culler, D.: Telos: enabling ultra-low power wireless research. In: IPSN (2005)
27. Reddy, P.G., Sridhar, N.: Lakon: a middle-ground approach to high-frequency data acquisition and in-network processing in sensor networks. In: IPSN. ACM (2010)
28. Rumberg, B., Graham, D.W., Kulathumani, V.: Hibernets: energy-efficient sensor networks using analog signal processing. In: IPSN (2010)
29. Sadasivan, S.: An Introduction to the ARM Cortex-M3 Processor. Technical report (2006)
30. Sadler, C., Martonosi, M.: Data compression algorithms for energy-constrained devices in delay tolerant networks. In: SenSys (November 2006)
31. Schmid, T., Shea, R., Srivastava, M.B., Dutta, P.: Disentangling wireless sensing from mesh networking. In: HotEmNets (2010)
32. Schott, B., Bajura, M., Czarnaski, J., Flidr, J., Tho, T., Wang, L.: A modular power-aware microsensor with $>$1000x dynamic power range. In: IPSN (2005)
33. Simon, D., Cifuentes, C., Cleal, D., Daniels, J., White, D.: Java on the bare metal of wireless sensor devices: the squawk java virtual machine. In: International Conference on Virtual Execution Environments (2006)
34. Skraba, P., Guibas, L.: Energy Efficient Intrusion Detection in Camera Sensor Networks. In: Aspnes, J., Scheideler, C., Arora, A., Madden, S. (eds.) DCOSS 2007. LNCS, vol. 4549, pp. 309–323. Springer, Heidelberg (2007)
35. Smith, R.B.: Spotworld and the sun spot. In: IPSN (April 2007)
36. Texas Instruments. CC2520: Second generation 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver (2007), `http://www.ti.com/lit/gpn/cc2520`

37. Texas Instruments Incorporated. MSP430 Datasheet
38. Wood, A., Stankovic, J., Virone, G., Selavo, L., He, Z., Cao, Q., Doan, T., Wu, Y., Fang, L., Stoleru, R.: Context-Aware Wireless Sensor Networks for Assisted Living and Residential Monitoring. IEEE Network (2008)
39. Xu, N., Rangwala, S., Chintalapudi, K.K., Ganesan, D., Broad, A., Govindan, R., Estrin, D.: A Wireless Sensor Network for Structural Monitoring. In: SenSys (November 2004)
40. Zephyr Technology. BioHarness BT (2010), http://www.zephyr-technology.com