A Modular Power-Aware Microsensor with >1000X Dynamic Power Range

Brian Schott, Michael Bajura, Joe Czarnaski, Jaroslav Flidr, Tam Tho, and Li Wang
University of Southern California
Information Sciences Institute
Arlington, Virginia

Abstract—We introduce a power-aware microsensor architecture supporting a wide operational power range (from <1mW to >10W). The platform consists of a family of modules that follow a common set of design principles. Each module includes a local power microcontroller, power switches, and isolation switches to enable independent power-down control of modules and module subsystems. Processing resources are scaled appropriately on each module for their role in the collective system. Hard real-time functions are migrated to the sensor and radio modules for improved power efficiency. The optional Linux-based processor module supports high duty cycling and advanced sleep modes. Our reference hardware implementation is described in detail in this paper. Seven different modules have been developed. We utilize an acoustic vehicle tracking application to demonstrate how the architecture operates and report on results from field tests on tracked and wheeled vehicles.

Keywords-component; low power embedded systems; power aware computing; microsensors

I. INTRODUCTION

Unattended ground sensors (UGS) or microsensors are specialized embedded systems designed for personnel and vehicle detection most often used in minefield replacement, force protection, and border monitoring applications. A typical microsensor system is composed of a processor, a radio, and several low-bandwidth sensors such as microphones, geophones, and magnetometers. Microsensors may also have high-bandwidth sensors such as infrared or visible spectrum imagers, micro-radar, or wide-band RF receivers [1]. Since microsensors are intended to operate in hostile or inaccessible environments, battery lifetime is the dominant design consideration. However, application performance metrics such as probability of detection, false alarm rate, response time, classification accuracy, and identification confidence are equally important metrics. Balancing power and performance is the main challenge in microsensor design.

Microsensor power and performance requirements are highly application dependent and vary according to sensor modality, target type, and terrain. Indeed, performance needs can change dramatically within the same mission. For example, a microsensor passively monitoring a quiet border has very different processing and communication patterns once targets begin to arrive. Multiple sensors may be used as the system adapts from detection, to classification, to identification and tracking. Power and performance at each stage scales with increases in sensor bandwidth and algorithm complexity.

Developing a power aware microsensor platform capable of managing the broad dynamic processing and power range of these applications is the focus of our research. We have established a baseline acoustic tracking scenario for tracked and wheeled vehicles with the goal of operating over a 1000X dynamic power range. At the lowest level of performance, our goal is to monitor acoustic activity at an average power of <1mW. The next stage is vehicle detection at 10mW. Once a vehicle is detected, our system should be able to compute a line-of-bearing to the target at 100mW. Finally, tracking of the vehicle, imaging, and sending spot reports can operate at around 1W. This four-stage scenario is a good approximation of many classes of applications for microsensors.

This paper describes our modular power aware microsensor developed under the DARPA-funded Power Aware Sensing Tracking and Analysis (PASTA) project. Section II introduces the distributed microsensor concept and design motivations. In Section III, we describe our reference implementation, hardware modules, and software environment. Section IV then reports power and performance results against our baseline vehicle tracking application. The paper concludes in Section V with lessons learned and future work.

II. ARCHITECTURE

We realized early in the design that traditional processor-centric architectures didn’t have the dynamic range necessary to achieve our goals. Even when sleeping, platforms built around 32-bit embedded processors have difficulty meeting our lowest power goals. Our early analysis using the WINS platform found that the rapid duty cycles of sensors and wireless network protocols tended to keep the processor awake and idling most of the time [2]. Microsensors built with 8-bit microcontrollers such as ”motes” [3] can meet the low-power goals, but have difficulty meeting the processing and memory requirements of complex algorithms. Since neither alternative spanned our dynamic range, a hybrid approach was chosen to adopt the best aspects of both classes of microsensors.

Our concept is a modular power aware microsensor consisting of a family of interchangeable processor, sensor, and communication modules that can be mixed and matched according to mission requirements. As shown in Figure 1, the platform could be configured in a variety of ways, such as: (a) simple radio relays, (b) low-power

Figure 1. Power Aware Microsensor Concept
sensor tripwires, or (c) high-performance trackers and imagers - all without having to learn and port to a new platform each time. Specific modules operate at power ranges appropriate to their role in the overall system. Furthermore, modules must consume minimal amounts of power when "off". The overhead of module power control and quiescent leakage power must be significantly less than our 1mW budget. In other words, even when the high performance processor is physically present in the system, the stack must electrically scale down to our lowest power mode (<1mW).

We define a power aware microsensor consisting of a system area network of sensor, radio, and processing modules, each with enough of local intelligence to manage its own resources. A central processor is not required in this architecture. Modules must exhibit a high degree of interoperability and support mechanisms for automatic discovery, channel allocation, and event or schedule-driven collaboration. The key observation for this distributed architecture approach is that sensor and radio modules are event driven entities operating on independent schedules. Sensor module duty cycles are tied to the physical sensor sampling frequencies. Some amount of low-level processing operates in real-time, which is able to trigger more advanced processing on detection events. Radio modules can have similar schedule and event-driven behavior defined by their media access control and routing protocols. In our distributed architecture, each module is allowed to efficiently operate on its own schedule and power off when not in use.

III. IMPLEMENTATION

We identified a number of design goals when developing our distributed power aware microsensor system. First, the power overhead needed to manage the module should be relatively small—essentially free compared to what is consumed during normal operation. Second, inter-module communication must be relatively power efficient at rates ranging from kilobits to megabytes. We found that natively supported protocols generally achieve the best energy-bit efficiency in embedded devices. Third, communications channels need to be commonly supported and flexible. Custom point-to-point connections should be avoided. Channel setup and power management should be software-driven and support module discovery. Finally, configuration of the stack should be electrically and mechanically universal. We wished to avoid stack-order dependencies found in other platforms. Ideally, multiple copies of the same module may coexist within the same microsensor stack.

Other goals were driven by practical experience from using other platforms in the field. Support for rapid prototyping and data collection were important objectives. Data collection is often overlooked, but extremely important during application development. We desired the ability to create testbeds using COTS peripherals similar to how PC/104 systems are used [4]. Availability of Linux device drivers and friendly programming tools were strong motivators.

Our reference implementation of this power-aware architecture concept is shown in Figure 2. We selected a 180-pin connector for our microsensor stack, which is logically partitioned into two busses. The core module bus contains power, mandatory control signals (reset, clock, control I2C, JTAG), and six 8-bit "serial" channels. The 8-bit channels have recommended pin assignments for standard serial interfaces (2xUART, 2xSPI, 2xI2C) with unused pins as general purpose I/O. The remainder of the connector (120 pins) is reserved as a processor expansion bus. In this bus, we've allocated a full 32-bit memory bus and common peripheral interfaces such as: compact flash (x2), USB master, USB slave, LCD, MMC, keyboard, mouse, and AC97. This arrangement allows for the most flexibility. Low power modules only use the core module bus, but pass all 180 pins through the stack. Changing the processor won't require updating other modules that make up the microsensor system.

Every module in the stack contains a microcontroller (µC) that controls a power switch and a set of bus isolation switches. These microcontrollers are always on, but consume very little power because they are clocked very slowly (32kHz) when not actually shutdown. The microcontrollers form a network over I2C for module discovery, power management, and channel allocation. They can power off the entire module with the power switch. The six bus-isolation switches have two functions. First, they prevent power leakage of other modules when the module power switch is off. Second, they allow the equivalent of an electronic breadboard for connecting serial channels between modules. The microcontrollers can negotiate for one of the six channels, enable the switches on each module, and allow the modules to communicate directly using their most efficient serial interface (SPI, UART, or I2C).

A. Hardware Modules

We have produced seven different modules in this research project. Schematics and source code are available from the PASTA web site (http://pasta.east.isi.edu). Each module is a small printed circuit board approximately 6.5cm (2.5") by 4.5cm (1.75"). For comparison, this is just slightly larger than a MICA2 mote [5]. This footprint was selected to support our 180-pin connector and a compact flash socket. Each board has a plug connector on top and socket on bottom, so that modules can stack together. The plug connector comes in a variety of heights from the manufacturer. The height is set according to the tallest component on the board. An example stack including IOB, ADC, and PXA is shown in Figure 3.

1) PXA Module

The PXA module in Figure 4 contains our main embedded processor, an Intel XScale™ PXA255, with 64MB of SDRAM, and 32MB of flash. The processor supports clock rates from 100-400MHz, with a 33MHz idle. The core voltage supply can be scaled from 0.85V to 1.3V during operation and shutoff when asleep. This module includes the Intel SA-1111 coprocessor, which provides support for USB master and two compact flash card interfaces. Serial interfaces, such as SPI, UART, and I2C are wired to the core module bus. The 100MHz memory bus and other parallel peripheral interfaces such as LCD and compact flash are wired to the processor expansion bus.

2) IOB Module

IOB regulates the battery supply (6-12V) to generate the primary 3.3V supply used in our stack. Regulating on the IOB was a tradeoff to save board space on other modules. IOB contains most of the digital I/O connectors. USB master and USB slave are visible in Figure 5.
Three serial connectors (visible in Figure 3) can be routed to any of the serial channels on the core module bus using bus isolation switches. The leftmost connector has RS232C transceivers to enable direct interfacing to up to two standard serial ports using a Y-adapter.

3) CF Module

CF contains a standard Compact Flash socket shown in Figure 6. Up to two CF modules can be used together. Since the Compact Flash interface standard has its own mechanisms for card discovery and power shutdown built into the SA1111 on the PXA module, CF does not have its own microcontroller.

4) ADC Module

ADC is shown in Figure 7. It was specifically developed to support acoustic beamforming for tracking vehicles. However, it is general purpose enough to be used in other applications. ADC has a 12-bit four-channel 100ksps analog to digital converter. Each of the four channels has programmable cutoff filter and gain stages. The 8051 can operate in store-and-forward mode, where data is sampled continuously then forwarded over one of the SPI channels. This allows the main processor to sleep periodically without missing any data.

5) TRIPWIRE Module

TRIPWIRE is a single-channel 12-bit analog-to-digital converter with optimized for 1 kHz operation. The gain is limited to two settings to save power. It was designed for performing acoustic vehicle detection under 10mW.

6) FPGA Module

The FPGA module shown in Figure 9 was built to emulate a DSP being developed at MIT for acoustic vehicle detection under 1mW [6]. The module contains a Xilinx Virtex-II 3000 FPGA, 2MB SSRAM, and 32MB of SDRAM. FPGA operates as a coprocessor on the 100MHz PXA memory bus.
7) MOTE Module

MOTE is an adapter module that allows mote microsensors using the standard 51-pin connector [7] to operate within our stack. The module is shown in Figure 10. In normal operation, our adapter fits between the mote processor and sensor boards. We can use the mote stand-alone as low power radio module. It can still use standard 51-pin sensor boards, but the adapter must be at the end of our stack in this configuration. Figure 11 depicts this configuration where adapter fits between the MICA2 processor and sensor boards. Finally, we can use mote sensor boards without the mote processor. In this configuration, our onboard 8051 is used in place of the mote processor.

B. Software

On the PXA processor module, we use the standard Linux 2.6.9 kernel with ARM and PXA patches in addition to our board support package [8]. A Linux 2.4.19 kernel is also supported. Our software distribution is derived from Familiar (http://www.handhelds.org), which is the most common Linux distribution found on personal digital assistants. Consequently, there is a wide range of device drivers, system daemons, utilities, and application programs available for our platform. We use the iPKG package management system for software updates over the Internet.

The software development and debugging environment for our embedded Linux system is very robust. We use the standard GCC cross compiler tool chain for ARM processors. Applications can be natively compiled on our platform. The ability to use USB and Compact Flash peripherals such as 1GB microdrives and portable hard drives allows a complete Linux software distribution to be installed on our microsensor. Similarly, wired and wireless Ethernet adapters allow us to log into the nodes with ssh and mount networked file systems such as NSF. Our system also provides a serial console through the IOB module for low-level debugging.

The 8051 microcontrollers programs on our system are compiled using the small device C compiler (SDCC). Our microcontrollers share a common messaging protocol over the I2C network. Basic module functionality is captured in a set of commands supported by common library routines. Each module extends the base commands with new capabilities. We don't use an RTOS. The task scheduler and event handlers called from the main event loop are very simple. A useful feature of our software infrastructure is the ability to flash new code images over the I2C network without having to resort to JTAG programming. Our microcontroller software distribution also takes advantage of revision control provided by iPKG.

IV. RESULTS

We have established a baseline acoustic tracking scenario for tracked and wheeled vehicles with the goal of operating over a 1000X dynamic power range. At the lowest stage, our goal is to monitor acoustic activity at less than 1mW. The next stage is vehicle detection at 10mW. Once a vehicle is detected, our system should be able to compute a line-of-bearing to the target under 100mW. Finally, tracking of the vehicle, imaging, and sending spot reports can operate around 1W.

Figure 12 diagrams the five modules used for our experiments: IOB, PXA, ADC, TRIPWIRE, and MOTE. The IOB is connected directly to a lab power supply. A current sense resistor on the IOB is enabled in series before the DC/DC converter. A national instruments ADC records voltage drops across the current sense resistor at 10kHz to compute power consumed by the PASTA system. This rate is sufficient to detect and sample individual cycles of the DC-DC converter.

A. Low Power Modes

The lowest power mode of 0.182 mW is achieved when the IOB cuts power to the entire stack and the main IOB 8051 is in its "halt" state. The IOB watchdog counts down to decide when to resume operation. This mode is used when the system is down for relatively long periods. When the counter resets, it restarts the stack by turning on the main power switch. The next lowest power mode of interest is 0.780 mW, when all modules in the stack are being powered, but in their "halt" state. This represents the minimum overhead needed to keep all other modules at their lowest power when a single module is active. For example, a MOTE would need to stay active to maintain network connectivity, while the rest of the PASTA system could be in "halt". This overhead can't be avoided because all microcontrollers share a common power supply.

We encountered a number of discrepancies from our datasheet analysis. For example, even though the UART interface was disabled there was a noticeable 0.02 mW drain until the serial cable was disconnected. On a different computer, the serial cable actually appeared to power the system. There were also differences between our measurements and the documented power for "halt" and "idle" states of the 8051. The cause could be estimation errors for 32kHz operation or undocumented power leakage from interfaces. Low power modes are summarized in Table I.

<table>
<thead>
<tr>
<th>Power (mW)</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.182</td>
<td>IOB in halt/standby mode, power cut to rest of PASTA &quot;stack&quot;</td>
</tr>
<tr>
<td>0.780</td>
<td>All modules in stack powered and operating in halt/standby mode</td>
</tr>
<tr>
<td>2.250</td>
<td>All modules in stack powered, operating in Idle mode</td>
</tr>
</tbody>
</table>

![Figure 12. PASTA Stack Configuration](image-url)
B. PXA Power Modes

The measured power of PXA in its "sleep" state is 7.3 mW, not including the overhead of other modules in the stack. This is the lowest power mode that retains Linux state. With all other modules in the stack powered and idle, the total stack power when the PXA is sleeping is 9.55 mW. If the processor core voltage is not shut down during sleep, the total stack power increases to 25mW. When the processor is running, it tends to dominate total power. With the processor running at 200MHz and memory bus at 100MHz, PXA power is 286 mW while executing the idle instruction and 460 mW in a busy while (1) loop. Beyond this, real application power is highly dependent on memory access rates and peripherals. Although we haven't reached this even with our FPGA module running, our power supply can support up to 15W.

Again, we encountered a number of discrepancies that highlight the danger of relying on datasheets for system power prediction. The design estimate for the best sleep mode was about 50% lower, and most of the budget was SDRAM refresh. This difference is under investigation. Additional system tuning may get us closer to our goal.

Another area of investigation is the sleep/wake core power sequence. The data sheet recommends the default 10ms of pre-charge time before the processor resumes operation after the core was disabled. We are able to disable this default on our hardware to shorten the time. We reduced the pre-charge time to 5ms in our design. However, we have measured large power drains (over 500mW) after turning on the core supply but before the processor resumes operation. Unless we can eliminate this effect, it may be beneficial to leave the core supply on when the processor sleeps for relatively short periods, and only disable it for longer periods. The sleep time overhead of the PXA processor and Linux 2.4 kernel for the sleep/wake cycle is 4ms.

The low-power savings of the modular pasta design are illustrated here. A processor-centric microsensor design using the PXA255 processor would be limited to 7.3mW as it's lowest-power operating mode because the processor could not shut itself off completely during extended sleep periods. Our modular design achieves a lowest system power mode of 0.182 mW, for a 40x reduction in low-end power. PXA power modes are summarized in Table II.

### Table II. PXA Power Modes

<table>
<thead>
<tr>
<th>Power (mW)</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>Additional power of processor in sleep with 64 MB SDRAM memory refresh</td>
</tr>
<tr>
<td>9.55</td>
<td>All modules in stack powered, in idle mode, except PXA in sleep mode with memory refresh, core power OFF</td>
</tr>
<tr>
<td>25</td>
<td>All modules in stack powered in idle mode, PXA processor in sleep mode with memory refresh, core power on</td>
</tr>
<tr>
<td>286</td>
<td>Processor Idle at 200MHz, console USB disabled</td>
</tr>
<tr>
<td>460</td>
<td>Processor running while (1) loop at 200MHz, console and USB disabled</td>
</tr>
</tbody>
</table>

C. Baseline Application Performance

This section describes the theoretical and achieved performance of our modular system when mapping our acoustic beamforming application. Our system power goal for this mode is 100mW. Our beamformer uses 1kHz sampling at 12-bit resolution on four microphone channels. In a processor-centric architecture, the processor would have to run continuously because the best PXA sleep cycle is 4ms—too long to sample at 1ms. However, our modular system is able to separate real-time data collection from near real-time signal processing. Our ADC buffers data and periodically transfers it to PXA for processing. Measuring theoretical performance of our system involves studying PXA and ADC module interactions. Our algorithm executes at about 300x real-time on the processor, so the PXA runtime is dominated by data transfer. The theoretical transfer time for one second of data is 34ms over a maximum rate 1.8mbps SPI channel. This implies that the PXA can sleep >95% of the time. The ADC sustains sampling using a 3MHz clock, but needs to speed up to 24MHz to sustain the 1.8mbps SPI transfer. The ADC can therefore operate at 3MHz for 95% of the time and 24MHz the remaining 5%. The estimate for total system power is under 50mW. The predicted savings of modular design is 5x the processor-centric approach.

Our measured performance presently falls short of our goal, but still illustrates a savings for modular design. We measured stack power at 178mW when running the beamformer. This is still 30% less than the PXA processor running in idle mode performing no computation (286mW). Our bottleneck lies with the ADC. It operates at 24MHz for 100% of the time instead of our 5% goal. The ADC at 24MHz consumes 49 mW, half of our total budget. Furthermore, SPI transfer takes four times longer than expected (160ms). Consequently, the ADC is using much more power than predicted and the PXA is "on" longer because of the data transfer time. Increasing the ADC to 49MHz reduces transfer time to 66ms (still twice as long as expected), but ADC power increases to 84 mW. Total power at this rate is 197mW. The shorter processor "on" time offsets some ADC increase.

The strength of the modular approach is how we are able to proceed with module-level system optimization to dramatically improve overall system power. Focusing just on the ADC store-and-forward functions will enable us to achieve our goal. We believe we can eliminate the SPI performance issues on the ADC at 24MHz by careful optimization of the SPI interrupt service routine on the 8051. Several of the sensitive data structures and offset counters can be moved into faster internal memory (IDATA). There is also some unnecessary function call overhead for the sake of code sharing. Dynamically adjusting the 8051’s system clock is presently limited by the fact that both the sample clock and the filter clock are derived from the system clock. Changing this clock now will glitch the filter and sampling. We can address this issue by either running the sample and filter clock from the external 32kHz clock, adjusting the clock frequency within the sampling ISR to prevent counter glitches, or introducing two distinct operating modes in the ADC (sample and transfer). In the third option, there would be a 38ms gap between each second of data. These options are currently being explored.

V. Conclusion

We have introduced a modular power-aware microsensor architecture that distributes intelligence out to the individual modules in the system. The processor resources on each module are appropriately scaled for their role in the collective system. Intelligent modules can take the hard real-time burden away from the main processor for better power management. We have shown that this architecture can be built with the addition of a low-power microcontroller, a power switch, and bus isolation switches on each module. Tangible benefits in terms of power, performance, and flexibility can be achieved on real applications using this approach.

Figure 13 shows graphical displays from our acoustic beamforming field test at Fort AP Hill in July 2004. In the map display, the six nodes (blue dots) are spaced approximately 100m apart. The yellow lines indicate computed bearing on the node. The brightness of the bearing line increases with a signal confidence measure. The red dot is the estimated position from our tracker algorithm. A blue circle (partially obscured by our target estimation) indicates GPS ground truth for the vehicle. The bottom-left inset shows our acoustic debugging displays for both time and frequency domains from all four channels. The data is plotted upside down because the window origin is in the top-left corner. The second window in this inset shows a polar plot of our beamformer response. Finally, the top-left inset shows our field prototype. Batteries are...
packaged in the same box with the hardware. The vertical post holds an antenna for our 802.11b instrumentation network.

Future work on this system includes additional tuning of our ADC module for better beamforming results and vehicle detection on our TRIPWIRE module. Additional field tests are planned in 2005 to demonstrate the full end-to-end system and hybrid integration with mote sensor networks. PASTA is free for Government use and the software is open source.

ACKNOWLEDGEMENTS

The authors would like to thank George Ruda, Tony Anderson, Dowell Black, Kevin Boyle, Phil Lundy, David Randall, and Adam Terio for their contributions to this program. This work would not have been possible without the invaluable support of the people of U.S. Army CERDEC at Ft. Monmouth, Ft. Belvoir, and Ft. A.P. Hill.

This research is sponsored by the Defense Advanced Research Projects Agency and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F33615-02-2-4005. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force Research Laboratory, or the U.S. Government.

REFERENCES


