On-demand Linux for Power-aware Embedded Sensors

Carl Worth, Mike Bajura, Jaroslav Flidr, and Brian Schott
USC/Information Sciences Institute
{cworth,mbajura,jflidr,bschott}@east.isi.edu

Abstract

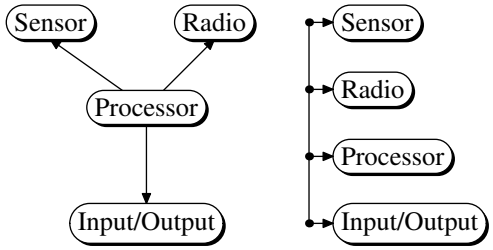
We introduce a distributed sensor architecture which enables high-performance 32-bit Linux capabilities to be embedded in a sensor which operates at the average power overhead of a small microcontroller. Adapting Linux to this architecture places increased emphasis on the performance of the Linux power-up/shutdown and suspend/resume cycles.

Our reference hardware implementation is described in detail. An acoustic beamforming application demonstrates a 4X power improvement over a centralized architecture.

1 Introduction

Traditional sensor platform architectures are based on a hub-and-spoke model with peripherals clustered around a central processor as shown in Figure 1(a). In this model, the lower-bound of total system power is set by the lowest active mode of the central processor which must be continually active to broker peripheral operations.

System power is typically reduced by using less-capable processors or microcontrollers in place of the central processor. Although sensor activity is mostly infrequent and bursty with low average computational requirements, peak processing requirements can still be quite high. Within a system design, this creates tension between the desire for the high-performance processing capability of a larger processor and the low-power operation of a smaller one.



(a) Hub-and-spoke (b) Distributed

Figure 1: Alternate sensor node architectures

This tension is further complicated by the observation that while many large processors require significantly more power than small ones when inactive, they also often provide significantly more power-efficient computation when active. Another tradeoff in this design space weighs the strength of development and debugging tools, such as Linux, available for larger processors versus the constrained programming environments available for small ones.

Replacing the hub-and-spoke architecture with a distributed model as shown in Figure 1(b) can decouple processing from peripheral operation and create a system that combines the strengths of both large and small processors.

In this model, processor and peripherals become autonomous modules that are each powered independently. High-performance processing can be made available when needed, but without increasing the lower-bound of total system power. Low average system power can be achieved by operating for a majority of the time in extremely low-power modes with only essential modules active.

This distributed architecture places Linux in an unconventional role as a peer module rather than as a central processor. This emphasizes the performance of the power-up/shutdown and suspend/resume cycles as keys for achieving low average system power.

The remainder of this paper is organized as follows. Section 2 describes several popular research and commercial sensor platforms. Section 3 recounts the design challenges we faced as we built a reference sensor node with autonomous modules. As usual, real-world issues forced difficult engineering decisions. Section 4 details the modules we have built so far.

Our hardware is in a more complete state than our software. We have identified some aspects of the behavior of Linux that we need to investigate more fully. These issues are discussed in Section 5. Section 6 contains power results we have obtained with a vehicle tracking algorithm. Finally, Section 7 draws conclusion and Section 8 describes areas for future work.

2 Related Work

Applied research in wireless sensor networks has made use of a variety of platforms with varying processing capabilities and power requirements, but almost always with a hub-and-spoke model. Several platforms are described below in order from more-capable, higher-power platforms to less-capable, lower-power platforms.

2.1 PC/104

PC/104 [3] is a well-supported specification for PC-compatible, embedded systems consisting of stacking modules. Cerpa et al [2] chose PC/104 systems for their "high end" sensors in a tiered deployment for habitat monitoring. Their cited reasons for choosing PC/104 include the ability to run PC-compatible software (ie. Linux) and the wide spectrum of available PC/104 modules.

PC/104 provides great flexibility and the power requirements are lower than that of desktop PCs. However, at 1-2 Watts per module[3], and with most sensors requiring at least two modules, this platform requires too much power for many sensor applications.

2.2 Embedded StrongARM devices/PDAs

Off-the-shelf devices based on low-power, embedded processors such as the Intel SA-1110 or PXA25x offer another convenient platform for sensor network research. Compared to x86-class processors, these processors offer a significant power savings along with a reduction in maximum clock rate and the absence of a hardware floating-point unit.

Representative devices of this class include the HP iPAQ, the CerfCube[4], and the Crossbow Stargate[14]. These devices are extensible via Compact Flash, Bluetooth, etc. but have less flexibility than PC/104. And while the power requirements of these systems can be less than a comparable PC/104 stack, Mainwaring et al[9] found that at 2.5W active power, the power usage of the CerfCube was excessive for long-term use in a sensor network.

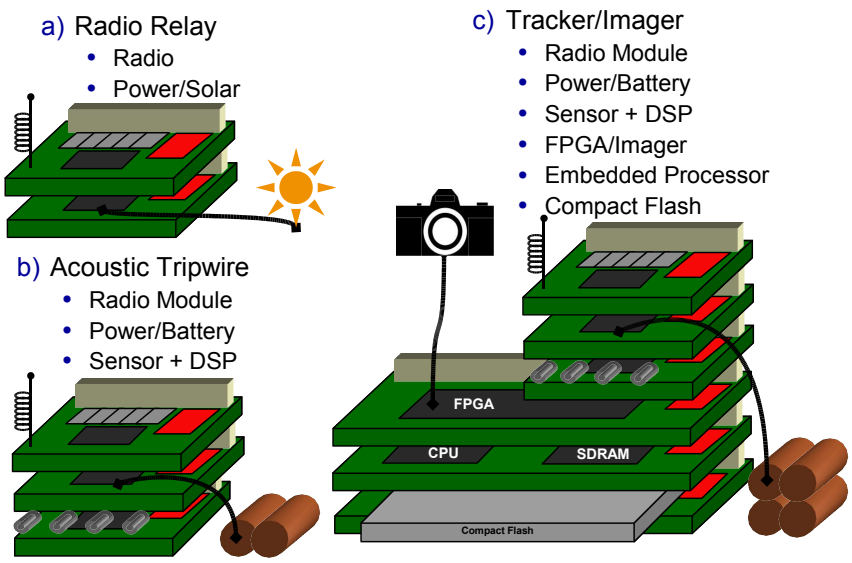


Figure 2: Power-aware sensor concept

Several research sensors have been developed with architectures similar to these off-the-shelf platforms. These include the μAMPS[10] and WINS[1] nodes. For example, the WINS node has a central 133MHz StrongARM SA-1100 processor along with radio and sensor peripherals. As measured by Raghunathan et al[12] the WINS node operates in the range of 360-1080 mW and can also be placed into a 64 mW sleep mode.

2.3 Motes

An example of a very low-power sensor architecture is that of the Berkeley Motes[6, 7, 8]. Current Motes are based around a central microcontroller (MCU) such as the ATmega 90LS8535, an 8-bit MCU with 128KB Flash and 8KB SRAM. The Mote includes a radio and has serial connections and 10-bit analog ADC ports to various sensors on expansion modules. Typical power consumption for this sensor when active is in the 10-100 mW range. Sleep power is about 60 μW.

Mote-class sensors demonstrate that a wide variety of low-bandwidth sensing applications can be accomplished with very small processors and with very little memory. The limitation of these systems is encountered when an application doesn't fit within the memory and processing footprint of the MCU. High bandwidth sensor processing is beyond the capabilities of these small-scale sensors and there is little room for expansion.

3 Implementation

Our primary system design goal was to construct a family of interchangeable processor, sensor, and communication modules that can be mixed and matched according to the application requirements. Ideally, our architecture would be able scale from simple sensors as shown in Figure 2(a) and Figure 2(b), to complex sensors as shown in Figure 2(c) without having to learn and port to a new platform at every scale.

Other goals were driven by practical experiences of using other platforms in the field. Rapid prototyping is an important concern. The ability to create testbeds using COTS peripherals is a strength of platforms such as PC/104. Availability of Linux device drivers and a friendly programming environment were strong motivators in our implementation decisions. Data collection is an important step in sensor network algorithm development. We wanted lots of data storage and data networking options in our new platform.

Primary constraints on the system design include size and power. We targeted the size of the Berkeley Mote, while still supporting a Linux-capable processor in the stack. In the end, the size was dictated by the minimum footprint of a Compact Flash socket and our chosen stack connector. Power in our system needs to be able to scale from 1 mW to a few Watts. The low power target limited many of our implementation choices.

An early design decision was how the autonomous modules would communicate. Interfaces such as ethernet were quickly dismissed due to power requirements. In small embedded devices, the most power-efficient communication is available with hardware-supported interfaces such as Serial Peripheral Interface (SPI), Controller Area Network (CAN), Universal Asynchronous Receiver Transmitter (UART), and Inter-Integrated Circuit (IIC or I2C). Each of these interfaces has strengths and weaknesses. I2C supports multi-master operation, but has a limit of 100kbps on most devices. SPI has faster transfers, (up to a few megabits per second), but has limited multi-master support in most devices and little flow control. UART is widely supported, but requires clock agreement on both interfaces. CAN is multi-master, but the bus drivers in the supporting devices are relatively high power, (since CAN is designed for long cables).

We decided to support the three major interface standards (I2C, SPI, and UART). We allocated six 8-bit channels on our connector and specified two preferred channels for each standard interface. In order to prevent bus contention and power leakage when modules are off, all modules have bus isolation switches between themselves and the connector. We also introduced a separate I2C control network for module discovery and coordination of the switches.

A small microcontroller (MCU) is standard on most modules to control the bus isolation switches, a power switch, and any module-specific functions. The MCUs are intended to be always on when the node is operating, (with a power overhead as low as .05 mW). They network with each other over I2C to facilitate module discovery and coordinate access to the channels using a common messaging protocol.

Figure 3 contains a diagram of the features common to each module, as well as an optional processor expansion bus so that high-speed, high-power peripherals, (USB, Compact Flash, LCD, and AC97 audio), can be used in the stack or removed for low-power operation.

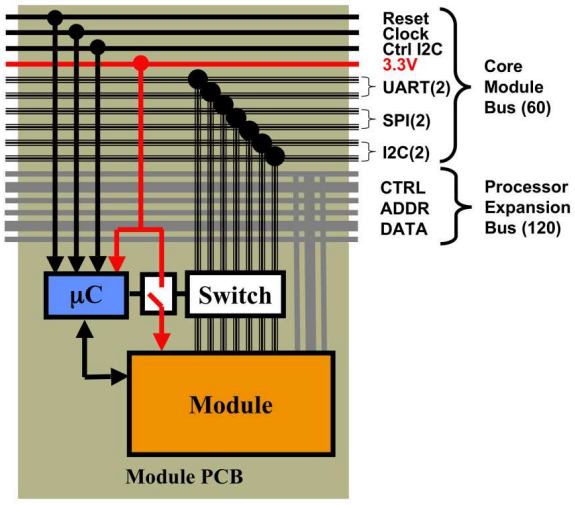


Figure 3: Power-aware module diagram

4 Available Modules

Our hardware modules are small boards approximately 6.5×4.5cm (2.5×1.75"), with a 180-pin connector on either side. So far, we have designed, built, and tested 4 modules, 3 of which are shown in Figures 4 and 5.

PXA A module including an Intel PXA255 XScale processor, 64MB of SDRAM and 32MB of Flash. This board supports dynamic voltage scaling, an active clock rate range of 100-400MHz, and a 33MHz idle mode. An SA-1111 coprocessor provides support for USB master and two Compact Flash cards. All interface lines are routed to the stack connector.

ADC A four-channel, 12-bit analog-to-digital converter module. The MCU has sufficient memory for a dedicated 7kB sample buffer. Basic signal processing can be performed by the local MCU or samples can be efficiently transmitted over an SPI interface to the PXA module for advanced processing. An important point is that the PXA255 processor can be off or suspended during data sampling.

IOB The power & I/O board is the only required module in the stack. It provides the primary power supply and contains most of the digital I/O connectors, (USB master and slave, SPI, I2C, and UART).

FPGA This module was developed as an emulation board for a low-power DSP being developed at MIT[15], but is interesting for other high-performance applications. It contains a Xilinx Virtex-II 3000 FPGA, 2MB of synchronous SRAM, and 32MB of SDRAM. This module operates as a coprocessor on the memory bus of the PXA255 and supports the two SPI channels on the stack.

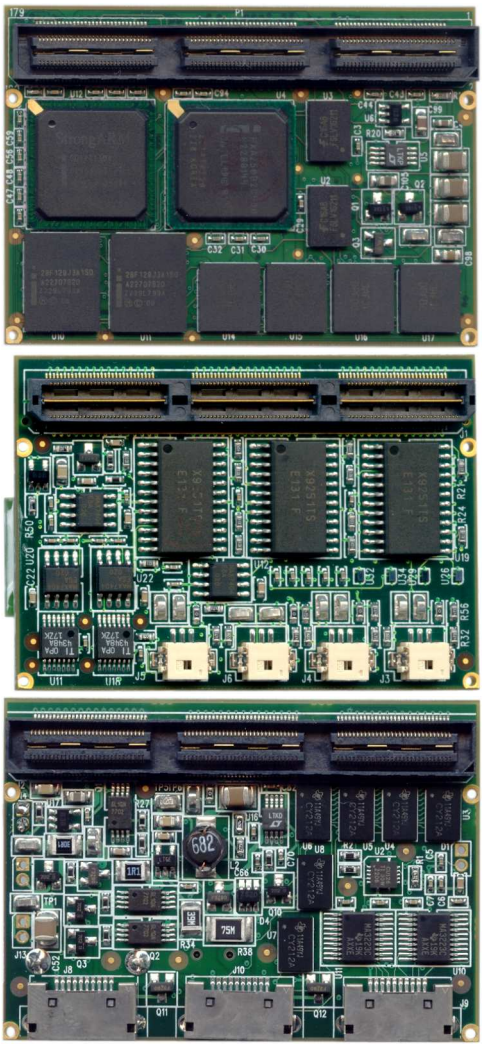


Figure 4: PXA, ADC, and IOB modules at actual size

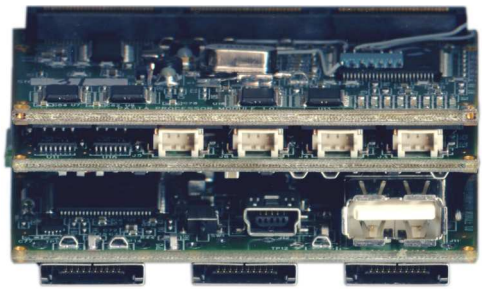


Figure 5: Stacked modules

We also have a Compact Flash (CF) board, two of which can be placed into the stack. This board connects to the processor expansion bus so that it acts as a daughter-board of the PXA module rather than an independent module. Figure 6 shows a stack consisting of IOB and PXA modules along with a CF board populated with a CF ethernet adapter.

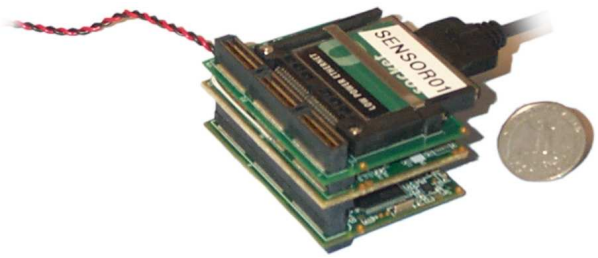


Figure 6: Stack including CF board

Table 1 shows design-time estimates of the power consumed by each module for various operational modes. In the "off" mode everything on the module is powered off except for the power-control MCU. The PXA module has the widest operational power range due to the the range of processor clock rates and various possibilities in processor and memory utilization. Figure 7 provides more details of the how the PXA module power can scale from 0.05 mW to 1.5 W. The innermost portion of this diagram includes figures for the overhead of power conversion and the 32kHz MCU on the IOB module.

Module	Mode	Power
PXA	off	0.05 mW
PXA	suspended	2.5 - 7.5 mW
PXA	active	150 - 1530 mW
IOB	active	0.1 mW
ADC	off	0.05 mW
ADC	active	40 mW

Table 1: Power modes for various modules

5 Impact on Linux

In a conventional hub-and-spoke model, Linux runs on a central processor and manages some number of peripheral devices. In contrast, our distributed architecture places Linux on an autonomous module which is a peer to other modules. Any other module might request a power transition of the Linux module, from off to powered, from suspended to active, etc.

The efficiency of these power transitions is a critical component of the average system power. The distributed platform is designed to achieve low average system power through aggressive duty-cycling of high-powered components. The time and energy spent during power-mode transitions is overhead that must be amortized, imposing limits on practical duty cycles that can be used.

We are currently using Linux version 2.4.21 with the standard ARM and PXA patches as well as customizations for our PXA module. The user-level software distribution is derived primarily from the handhelds.org[11] Familiar distribution. Our reference sensor application (see Section 6) does not turn the PXA module off, but does suspend/resume the processor aggressively to achieve active operation for a few milliseconds once per second. The time spent during suspend/resume is divided between time spent in driver callbacks and time spent in the kernel proper. Table 2 shows the times we have measured for these transitions on our PXA module.

Transition	Time
Suspend drivers	507 ms
Suspend kernel	13 ms
Resume kernel	78 μs
Resume drivers	25 ms

Table 2: Linux transitions with driver callbacks

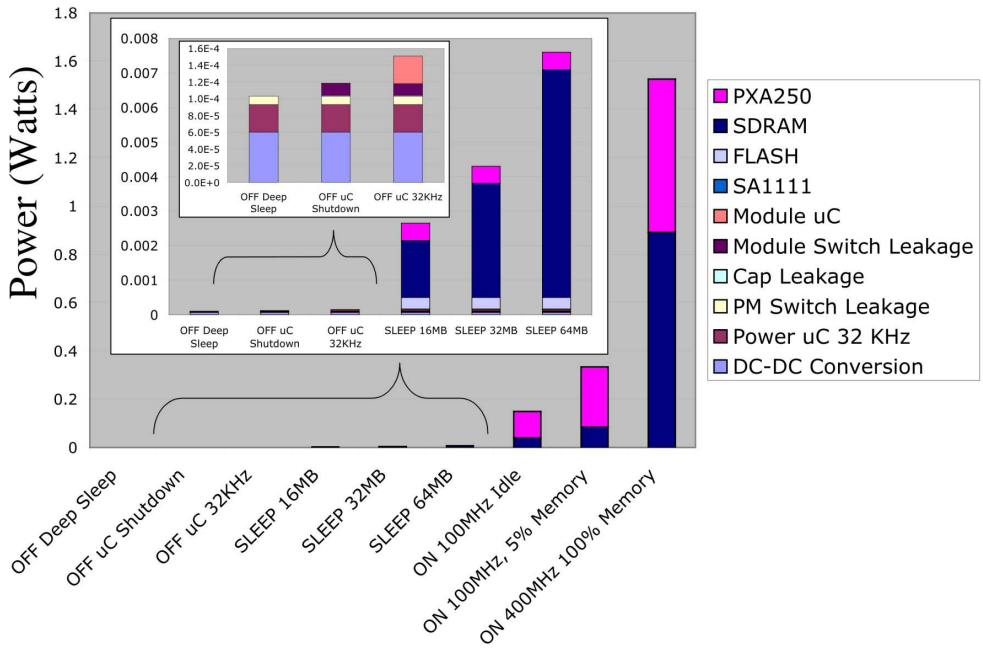


Figure 7: PXA module power states

We expect the suspend/resume transitions of the Linux kernel itself to behave in a symmetric fashion. However, we measured a very responsive resume time of 78 μs and a much slower suspend time of 13 ms. We do not yet have a complete explanation for why the suspend process is so much slower, although we have accounted for a 1 ms delay that is caused by the MCU on our PXA board, and therefore not a feature of the standard Linux kernel.

A much more significant problem is the time spent in the power management callbacks of various subsystems and drivers. A suspend time of 520ms spells disaster for an application such as the one described in Section 6.

We quickly tracked down the source of this long suspend time to the USB OHCI driver (hcd). An ill-fated decision in our design was the choice of the SA-1111 coprocessor. One difficulty we encountered was that we had to provide our own suspend/resume callbacks as they do not exist for the SA-1111-based hcd in the standard kernel. To simplify the task, we have ported the PCI-based OHCI code which contains a call to `mdelay (500)` along with the comment, "Suspend chip and let things settle down a bit". This single 500ms delay accounts for over 98% of the time required to suspend drivers. We suspect that this constant can be safely reduced so that much of the time lost during suspend can be recovered. Even so, USB-related timeouts, etc. are on the order of milliseconds—orders of magnitude more than the time required by the kernel.

Clearly, this poses a serious problem for applications with a high duty cycling requirement. We are currently working around the long driver suspend times by simply removing drivers for non-essential devices and subsystems, (such as SA-1111), prior to running an application with a restricted power budget. This allows the convenience of things such as using a USB 802.11 adapter during development and debugging without the long suspend times of the USB drivers during execution.

6 Results

We implemented a vehicle tracking algorithm using 4-channel acoustic beamforming. In this application, data is continually sampled at a rate of 1kHz, but signal processing only needs to be performed at a maximum rate of 1Hz.

In previous work[13], this algorithm was implemented on a successor to the WINS node, (a hub-and-spoke platform with an Intel SA-1110 processor). Although efficient signal processing software was developed, system power savings were modest since the processor had to remain active (yet mostly idle) at all times simply to drive data collection.

We have ported the algorithm to the distributed platform described in this paper. On this platform, the signal processing for one second's worth of data can be completed in 3 ms by the PXA255 processor running at 100MHz. Since this is a newer processor than the SA-1110 of the WINS platform, direct comparison of power numbers between the two platforms would be unfair. Instead, we estimate the power needed for two implementations of the algorithm on the distributed node.

The first version is intended to behave as if in a hub-and-spoke system. The processor remains active at all times to store samples into main memory. The second version takes advantage of the distributed nature of the platform. Linux on the PXA module is suspended as much as possible while the ADC module continues to sample and buffer data. This approach adds the overhead needed to suspend/resume Linux and to transfer data from the ADC module to the PXA module over the SPI channel. The amount of data to be transferred is 8192 bytes, (4 channels $*$ 1024 samples/s $*$ 2 bytes/sample $*$ 1 s). The SPI transfer rate is 1.8 Mbps yielding a total transfer time of 36.4 ms.

We measured the power consumed by the PXA module at two different stages in the algorithm. During active computation the PXA module consumes 528 mW. When mostly idle, (eg. when transferring 1kHz data from the ADC module), it consumes 370 mW. We have not yet measured the average power consumed during the suspend or resume transitions, but we use an estimate of 370 mW. This estimate should be conservative as the actual power usage should ramp down to less than 10 mW during the transition.

Combining these measurements with estimates from Table 1 and the time measurements from Table 2, we compute the total energy spent to compute one result per second. From this we can determine the average power necessary for the complete algorithm. These results are given for both versions of the algorithm in Tables 3 and 4.

Module/Mode	Power	Time	Energy
IOB active	0.1 mW	1 s	0.1 mJ
ADC active	40 mW	1 s	40.0 mJ
PXA active	528 mW	3 ms	1.6 mJ
PXA idle	370 mW	997 ms	368.9 mJ
Estimated energy per second			410.6 mJ
Estimated system power: 411 mW			

Table 3: Hub-and-spoke power requirements for beamforming

Module/Mode	Power	Time	Energy
IOB active	0.1 mW	1.0 s	0.1 mJ
ADC active	40 mW	1.0 s	40.0 mJ
PXA suspended	7.5 mW	948 ms	7.1 mJ
PXA resuming	370 mW	78 μs	28.9 μJ
PXA transferring	370 mW	36.4 ms	13.5 mJ
PXA processing	528 mW	3 ms	1.6 mJ
PXA suspending	370 mW	13 ms	4.8 mJ
Estimated energy per second			95.9 mJ
Estimated system power: 96 mW			

Table 4: Distributed power requirements for beamforming

7 Conclusion

The 96 mW beamforming result marks a success for the distributed sensor platform—a 4X power reduction over the 411 mW required for the hub-and-spoke platform. This shows that it is possible to take advantage of 32-bit, Linux processing without average power exceeding the 10-100 mW range of a less-capable sensor based on a 8-bit microcontroller, (ie. a Mote).

8 Future Work

The field of power-aware sensing is rich, and we have only just begun to explore the possibilities, even within our own platform. Many applications require a much smaller power budget than the 96 mW result we have demonstrated. Our long-term goal is to design sensors capable of operating entirely from scavenged energy, (eg. solar), which requires operation in the range of 1 mW[5].

We are currently building a low-power "tripwire" module which will implement single-channel acoustic vehicle detection. This will allow the PXA module to be completely off when a vehicle is not present. We anticipate that this will allow beamforming within a power budget as low as 10 mW.

As mentioned in Section 5, there remains a fair amount of engineering and research with regards to the role of Linux within a distributed sensor. This includes reducing the time required in the power management callbacks of all relevant drivers as much as possible. An additional task is to move from Linux version 2.4 to 2.6. The dynamic nature of the new unified device model in 2.6 should make a natural fit with a platform consisting of autonomous modules that can be powered on and off at any point.

Additionally, new power-scheduling research could better take advantage of the wide range of power modes in this platform. For example, Linux could monitor the frequency and duty cycles of the power-up/shutdown and suspend/resume cycles. It would then be possible to provide intelligent feedback into these cycles based on the relative overhead of each transition. This work would complement current efforts that dynamically adjust voltage and clock rate based on system load.

9 Availability

This work has been developed as part of the *Power-Aware Sensing Tracking and Analysis (PASTA)* project, but we hope that it will be useful to researchers and hobbyists with a wide range of applications. To that end we are working toward making the hardware modules available at cost. Further details will be made available at the PASTA website, `http://pasta.east.isi.edu`. All of the software developed under PASTA is also available there under the terms of the GNU General Public License (GPL).

10 Acknowledgements

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

[1] Jonathan R. Agre, Loren P. Clare, Gregory J. Pottie, and Nikolai P. Romanov. Development platform for self-organizing wireless sensor networks. In *Proceedings of Aerosense*, pages 257–268. International Society of Optical Engineering, 1999.

[2] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[3] PC/104 Consortium. PC/104 Embedded-PC Modules, 2004. `http://www.pc104.org/`.

[4] Intrinsic Corporation. CerfCube embedded strongarm system. `http://www.intrinsyc.com/products/cerfcube/`.

[5] L. Doherty, B. A. Warneke, B. Boser, and K. S. J. Pister. Energy and performance considerations for smart dust. In *International Journal of Parallel and Distributed Sensor Networks*, 2001.

[6] Jessica Feng, Farinaz Koushanfar, and Miodrag Potkonjak. System-architectures for sensor networks issues, alternatives, and directions. In *International Conference on Computer Design: VLSI in Computers and Processors*, page 226. IEEE, 2002.

[7] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[8] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for smart dust. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278. ACM Press, 1999.

[9] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM Press, 2002.

[10] R. Min, M. Bhardwaj, S. Cho, A. Sinha, E. Shih, A. Wang, and A. Chandrakasan. An architecture for a power-aware distributed microsensor node, 2000.

[11] The Handhelds.org Project. Handhelds.org (web document). `http://www.handhelds.org`.

[12] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks, 2002.

[13] R. Riley, S. Thakkar, J. Czarnaski, and B. Schott. Power-aware acoustic beamforming. In *Proceedings of the Fifth International Military Sensing Symposium*, 2002.

[14] Crossbow Technology. Stargate xscale processor platform. `http://www.xbow.com`.

[15] A. Wang and A. Chandrakasan. Energy-efficient dsps for wireless sensor networks, 2002.