OCTOBER 2000

WRL
Research Report 2000/6

The Itsy Pocket Computer

Joel F. Bartlett
Lawrence S. Brakmo
Keith I. Farkas
William R. Hamburgen
Timothy Mann
Marc A. Viredaz
Carl A. Waldspurger
Deborah A. Wallach

COMPAQ **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Itsy Pocket Computer

Joel F. Bartlett, Lawrence S. Brakmo, Keith I. Farkas, William R. Hamburgen,
Timothy Mann,[*] Marc A. Viredaz, Carl A. Waldspurger,[†] Deborah A. Wallach

Abstract

The Itsy pocket computer is a powerful information device small enough to be comfortably worn or carried. It was created to support research in user interfaces and applications. A 32-bit, 200-MHz microprocessor and 32 MB each of flash memory and DRAM made it the first small handheld able to run traditional desktop applications such as continuous speech recognition, speech synthesis, multimedia, and video games. Itsy offers fine-grain control of the hardware to support flexible power management and monitoring. Comprehensive expansion capability via daughtercards makes it easy to extend the system. The Linux operating system, with extensions for a flash file system, resource sharing, and power management, eased porting of several advanced application environments (X-Windows, Java and Squeak) and many applications. Our user interface experiments confirm that traditional desktop methods do not extend well to interactive applications on small devices, but that gesture- and speech-based approaches have great potential. Our power measurements and analysis show that 300 – 900 mW, which our battery can easily supply, is sufficient to run a variety of the most demanding applications.

1 Introduction

Many of us in computer architecture cheered as mainframes morphed into minicomputers and then into workstations on our desks. These same evolutionary forces are driving computation off the desktop and onto our bodies. In this new world, questions like *does it fit comfortably in your pocket* can outweigh such classic measures as TPC or SPEC. At the same time, the functionality that we seek—continuous speech recognition, video input and output, encryption, and continuous wireless connectivity—suggests that these classic performance measures will still be relevant, and that the new portable devices will need to be be measured against another even harsher standard: computation per unit of energy.

We believe that it is not sufficient to investigate off-the-desktop computing with thought experiments or with mockups based on notebook computers. While some useful things can be learned from such activities, the physical dimensions and mass of portable devices cannot be faked. No amount

[*] Compaq Computer Corporation Systems Research Center
[†] Formerly at Compaq Computer Corporation Systems Research Center, now at VMware, Inc.

Figure 1: The Itsy pocket computer version 2.

of weight training, creative tailoring, and positive thinking will turn a 1 kg sub-notebook into a 100 g pocket computer that can be controlled with one hand.

To that end, we have designed and constructed the *Itsy* pocket computer, an experimental, low-power, high-performance platform for wearable or off-the-desktop computing. Our design has focused on two goals: first, to maximize the performance packed into a unit that can be comfortably worn or carried all day in one's pocket or purse, and second, to minimize the constraints on experimenters who wish to customize and extend the system's hardware and software.

To the best of our knowledge, Itsy was the first pocket computer with sufficient processing power and memory capacity to run traditional desktop software, including continuous speech recognition, a full-fledged Java virtual machine, real-time MPEG-1 movie decoding, and the Linux operating system. In this paper, we describe key hardware and software attributes of Itsy and the rationale behind our design decisions. In addition, we quantify its performance and power consumption.

2 Itsy overview

The Itsy pocket computer, shown in Figure 1, was designed as a research prototype. Our primary hardware goals were to attain high performance, low power consumption and weight, and small size. We also wanted a base system with a rich feature set to support user interface and applications research, as well as the flexibility to easily add new capabilities. Criteria such as cost or suitability for volume manufacturing, which are of utmost importance for commercial products, played no significant role.

We chose a StrongARM processor for Itsy because it met our performance requirements with a superior MIPS per Watt ratio, and offered power-saving features such as *sleep* and *idle* modes (see Section 3.5). From among the StrongARM family, the SA-1100 [1] was chosen for its on-chip controllers that made it easy to build a complete system with few additional components.

The minimum size and weight of a small system is generally bounded by its battery and display. This trend is certainly true for Itsy, which is just 70% larger than these two components alone. Our Li-ion cell is just large enough to plausibly allow a full day of intermittent use, and the display was chosen mainly for its small size. The motherboard and an auxiliary board fill the remaining space. The amount of flash memory and DRAM was determined by the maximum number of parts that would fit without making the system still bigger.

A key attribute of our design is a flexible daughtercard interface. Unlike common extension standards (such as PCMCIA and CompactFlash), all the available functionality that could be used by a daughtercard is accessible, including the full memory bus. For example, we have not included a radio transceiver on the base system because there was no obvious best choice. However, the flexible daughtercard interface gives us the freedom to experiment: several projects, beyond the scope of this paper, are adding radios to the Itsy. The technologies being explored include metropolitan-area radio modems, campus-area networks, proximity networks, and even software radios.

Another aspect of the flexibility is that no assumptions were made about how the system is typically used. For example, no peripherals are automatically disabled when sleep mode is entered. Instead, each unit can individually be turned on and off. This strategy leaves the option of disabling any of these units while running, or conversely, of keeping any of them active while in sleep mode.

The same quest for a flexible environment guided the decisions made regarding the software. An open-source operating system was selected over a proprietary one, because of the ease of experimenting with new concepts (such as power management) and of sharing the resulting code with partners. Also, a general-purpose OS was preferred to a real-time OS, because of the much larger pool of applications usually available for the former. The Linux OS meets these requirements and is well suited to exploit the performance characteristics of the hardware.

3 Hardware

This paper focuses on Itsy v2, the second of two major versions we designed.[1] Building and using 75 Itsy v1 systems [2] yielded many lessons that influenced design decisions on Itsy v2. For example, power studies done on Itsy v1 [3] validated features of the Itsy v2 power system, and user experience suggested a need for much more than the 4 MB of flash memory provided on Itsy v1.

Figure 2 shows the general architecture of Itsy, while Table 1 provides its major specifications. Itsy is designed around the StrongARM SA-1100 [1, 4], a low-power 32-bit microprocessor implementing the ARM instruction set (without floating-point hardware) and providing most of the control and

[1] The actual system described here is version 2.3.

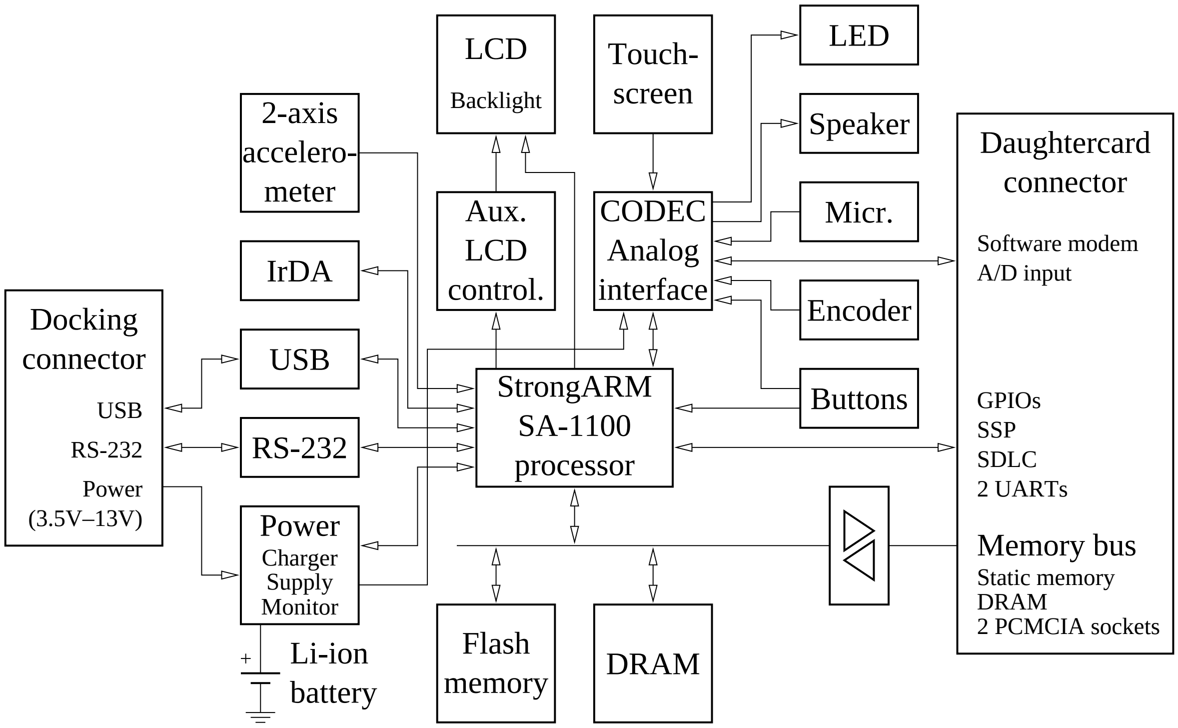

Figure 2: Itsy v2 architecture.

interface logic likely to be used on a handheld system. We used a small programmable logic device to implement some additional decode logic as well as an auxiliary LCD controller (see Section 3.2).

Flash memory is used for permanent storage as well as booting. It is also possible to force the Itsy to boot from a daughtercard; this feature is useful when bad code is inadvertently written to the motherboard flash memory.

Since the DRAM controller of the StrongARM SA-1100 supports only fast-page and EDO devices,[2] the memory size is limited by the largest parts commercially available (64 Mb). On the motherboard, a configuration of two 16-bit parts per bank (16 MB/bank) is used rather than four 8-bit parts per bank (32 MB/bank), because for a given memory size the former offers more power-efficient refresh and data access. The 16-bit parts are refreshed more efficiently (about 5.2 mW versus 9.5 mW, for 32 MB) due to a different internal topology that allows a lower refresh frequency. During data access, power is saved because only two parts are active instead of four. For example, memory copy using eight-word bursts at 206 MHz requires about 490 mW rather than 630 mW.

3.1 Communication interfaces

Itsy has been designed as a *USB device* [5]. Not only is the USB intended to be the primary communication link when an Itsy is docked to a USB host computer, but it is also used to recharge the

[2]The StrongARM SA-1110, which offers SDRAM support, was not available when the Itsy was designed.

Processor	StrongARM SA-1100 (59 MHz – 191 MHz)*
DRAM	32 MB (50 ns, EDO)
Flash memory	32 MB (90 ns)
IrDA (infrared)	4 Mb/s
Core voltage	1.5 V/1.23 V (switchable)
Main voltage	3.05 V
LCD	320×200 pixels, 15 grey levels
Battery	Rechargeable Li-ion (2.2 W · h)
Size	118 mm × 65 mm × 16 mm
Weight (w/o daughtercard)	130 g

* While the processor's manual [4] guarantees operation up to a core frequency of 191 MHz at 1.5 V, all Itsy systems built to date function correctly at 206 MHz, and even higher.

Table 1: Itsy v2 specifications.

battery. Itsy provides an RS-232 interface and an external power connection, mainly for low-level development and debugging. A 4 Mb/s IrDA-standard infrared port is also included.

3.2 Input/output devices

While power-saving passive-matrix displays are common on small systems, the 320×200-pixel LCD used on Itsy has a number of unusual characteristics. *Multi-line addressing* technology provides higher contrast than is typical for a 200-line single-scan passive display [6]. The 0.18 mm-pixel pitch is 25 – 30% smaller than is typical for small matrix displays, permitting dense text and crisp graphics. A high performance transflector and 0 – 400 mW LED backlight allow use over a full range of lighting conditions. The main LCD controller is implemented by the processor and offers 15 levels of grey. However, an auxiliary controller and the display's built-in 1-bit/pixel memory make it possible to display a static monochrome image while the processor is in sleep mode (see Section 3.5). A separate polyester-on-glass resistive touchscreen is mounted on top of the LCD.

Itsy has monaural audio with 12-bit A/D and D/A converters. The built-in noise-cancelling microphone and speaker can be bypassed by plugging in a headset. Additionally, there are eight general-purpose push-buttons, a rotary thumbwheel encoder with a push-to-select feature, and a two-axis accelerometer.

3.3 Daughtercard interface

The daughtercard interface has been designed with the flexibility and performance needed for easy development of hardware extensions. To this end, the full memory bus (all 32 data bits and all 26 address signals) is exported to the daughtercard. Since the processor decodes the address space internally, only the address ranges, or banks, that are not used on the motherboard are available for a daughtercard. These are:

Static memory banks: Two banks of 64 MB each can be used to implement any type of static memory (including flash memory, ROM, or RAM) or memory-mapped hardware.

DRAM banks: Two 16-MB banks can be added to bring the total memory size up to 64 MB. For a full 128-MB of DRAM, it is necessary to remove the DRAM from the motherboard, export all four banks to the daughtercard, and use the more power-hungry 32-MB banks. Other configurations are also possible.

PCMCIA banks: These banks allow a designer to implement up to two PCMCIA (or CompactFlash) sockets.

The memory bus is buffered at the daughtercard connector. This reduces the risk that a faulty daughtercard could affect the system and reduces signal-integrity concerns.

Besides the memory bus, every unused feature available on the motherboard is exported to the daughtercard. These include a number of serial protocols: SSP, SDLC, and two UARTs. There are also 14 general-purpose input/output (GPIO) pins, a software modem A/D input and D/A output, and a single general-purpose A/D input. Since the serial ports and the GPIOs share many pins, these features are not all available simultaneously.

3.4 Power system

On Itsy, the power system consists of a Li-ion battery, charger, and three regulated power supplies (switching regulators for main and processor core voltages, and a linear regulator for the analog section). The charger-battery system implements all the customary safety mechanisms for Li-ion cells, including over-voltage, over-current, and over-temperature. These mechanisms are implemented on the motherboard and the battery pack, with critical functions redundantly implemented on both. Although the charger hardware is functional by itself, there are provisions for software intervention. The processor can enable and disable the charger as well as switch between two maximum charge currents for compliance with USB-specified supply limits (100 mA and 500 mA) [5].

Our system uses logic specified to operate between 3.0 V and 3.6 V, but the Itsy power supply is set to about 3.05 V, only slightly above the minimum. Because power increases with the square of the voltage, this reduction from the usual 3.3 V results in a power savings of almost 15%.

Itsy can also monitor its own power and energy use. Voltage measurements and precision current sense resistors with differential amplifiers allow sampling of the power on four separate paths: power from the USB or other external source, power to and from the battery, power into the processor core supply, and combined power into the main and analog supplies. Four additional current sense resistors (for the DRAM and the three supply outputs) can be monitored by external laboratory instruments. In retrospect, an additional sense resistor to measure the StrongARM's 3.05V supply current would also have been very useful. A “gas gauge” circuit integrates the power into and out of the battery to continuously monitor state-of-charge. It has proven challenging to accurately measure the state-of-charge due to the large dynamic current range and the presence of small current measurement offset errors.

3.5 Power management

Itsy offers many power management mechanisms. These include processor support for software-controllable clock frequency and two low-power modes: idle mode and sleep mode. During idle mode, the clock to the processor core is gated off (saving power due to the CMOS circuit technology used), but the rest of the chip remains powered and all peripherals remain enabled. During sleep mode, most of the processor is unpowered, and only the real-time clock and the wake-up circuit remain enabled. The system clock can optionally be left enabled for faster wake-up.

During sleep mode, the LCD can either be disabled or be driven using the auxiliary LCD controller (see Section 3.2). The DRAM is usually put in self-refresh mode during sleep, but it can also be unpowered. These mechanisms allow us to implement a wide continuum of sleep modes, from a "light" sleep mode with LCD enabled, DRAM contents preserved, clock on, and most interrupts configured to wake up the processor, to a "deep" sleep mode, where only the real time clock is maintained.

Additionally, most peripherals, such as LCD, RS-232, and IrDA, can be individually turned on and off. It is also possible to select a processor core voltage of either 1.5 V or 1.23 V. Although 1.23 V is below the manufacturer's specification, it can be safely used at moderate clock rates and yields about a 30% power savings for the core, resulting in a 10% – 20% savings overall (see section 5.3).

4 Software

Our goal for the Itsy software was to produce a very flexible research environment. Among other priorities, we wanted to experiment with power management policies and strategies, rapidly prototype applications and user interfaces, and yet have a full, stable environment for large applications. To this aim, we ported and modified the Linux operating system. In addition, we designed and implemented a new device sharing model (*sessions*), which enables the Itsy to run different application environments concurrently. We now have a complete system that supports several large software environments including X Windows, Java, and the Squeak Smalltalk-80 system.

4.1 Operating system

For the greatest flexibility, we chose Linux as our operating system. Adapting Russell King's ARM Linux distribution (version 2.0.30) to the Itsy required an initial port to the StrongARM SA-1100 and drivers for Itsy-specific devices. We also modified Linux to better support handheld computing, including improvements for power management and non-disk-based filesystems.

4.1.1 Memory-based filesystem

Itsy supports dynamic memory partitioning between process address spaces and memory-resident filesystems. Typical memory-based filesystems waste space with redundant copies of data in the buffer cache and virtual memory system, which is unacceptable for a small handheld device. Fortunately, Linux already provided efficient support for *ramdisks* using a clever implementation technique. When filesystem blocks are flushed, they are written to the ramdisk block device, which simply locks down their buffer cache entries, ensuring that they will remain resident. When the filesystem reads from a block that has never been written, the ramdisk returns a new zero-filled block.

We made an extension to the ramdisk driver to discard empty blocks (blocks containing only zeros) as they are written, instead of locking them down. This simple form of compression was surprisingly effective, allowing us to create large ramdisk filesystems without actually consuming any physical memory until data blocks were used.

4.1.2 Flash-based filesystem

Flash-based filesystems allow stable, writable file storage on handheld devices without disks. Itsy's flash-based filesystem consists of an ordinary Linux filesystem (ext2) that expects to run on top of a disk, plus a block device driver that emulates a disk in a portion of the Itsy's flash memory. Since flash has very different characteristics than disks, such an emulation is non-trivial. In particular, flash must be erased before it can be written, the minimum erase unit is quite large (128 or 256 KB on Itsy), and erasing is quite slow (typically 700 ms per erase unit).

Linux already had a flash-based disk emulation for PCMCIA cards, using the industry-standard FTL (flash translation layer) data structure. We modified this code to work directly with Itsy's flash memory. FTL provides a layer of virtual block addressing. Whenever a virtual block is to be written, the FTL driver stores the data into a new, free physical block that has been previously erased, and updates its virtual-to-physical maps to point to the new block. When pre-erased free blocks run low, the driver reclaims space by choosing an erase unit that contains some superseded physical blocks, moving any non-superseded blocks in that unit to a different, spare unit, and erasing the old unit to become the new spare.

We observed a serious performance problem in this scheme. When the FTL driver reclaims space, it can only free up blocks that it knows are unused. Because FTL is a disk emulator, not a filesystem, it receives only block read and write requests. It never finds out that a block has been freed until the filesystem overwrites the block with new data. In the worst case, each time a virtual block is to be written, the FTL driver knows about only one unused physical block—the one containing the virtual block's old value—so it must erase and recopy a whole erase unit for every write, slowing FTL performance to a crawl. The Linux FTL driver avoids this absolute worst case behavior by defining the virtual size of the flash to be only 95% of its physical size, so that 5% of the physical blocks are always known to be unused. Therefore more than one block can sometimes be reclaimed by a single erase; however, FTL still performs as poorly as if the flash were 95% full no matter how empty it really is.

To correct this problem, we made a small modification to the ext2 filesystem. The filesystem now informs the block driver whenever it frees a virtual block, enabling the driver to reclaim the corresponding physical block ahead of time. This change greatly reduces the number of erases per write when the flash is not full.

4.1.3 Power management

As discussed in Section 3.5, the StrongARM processor provides three modes of operation: run, idle, and sleep. We modified Linux to take advantage of these different modes. Additionally, since the processor may be clocked at a variety of speeds, we provide the capability to rapidly switch among them.

Idle mode can be entered and exited in just a few clock cycles, so we modified the kernel idle loop to enter idle mode. Whenever an interrupt occurs, the processor exits idle mode and returns to normal

Peripheral	Possible sleep behavior	Default Linux sleep behavior
DRAM	self-refresh or off	self-refresh
LCD	static image or off	off
touchscreen	wakeup or off	off
buttons	wakeup or off	one wakeup button, rest off
accelerometers	off	off
serial	off	off
audio	off	off
charger	charging or off	charging

Table 2: Possible behavior of various components when the Itsy is sleeping, and their default behaviors under Linux.

operation. Idle mode is undetectable to the user.

Sleep mode provides a much greater power savings than idle mode. In return, it makes a much bigger impact on the system, both in terms of software and user detectability, since most of the processor is unpowered. In particular, the on-chip peripheral controllers will not work, including the LCD and UARTs. Table 2 shows the possible behaviors of the various peripherals while the processor is sleeping, and the default configuration for a sleeping Itsy when running Linux. Sleep mode can be entered through user request (such as a button press), by the kernel, or forced by a system power fault. Entering sleep mode takes about 150 μs; exiting takes about 10 ms if the clock was left enabled, 157 ms if it was disabled; there appears to be little power difference between these two cases.

We developed a power management module to coordinate suspending and resuming devices when sleep mode is entered or exited. Each device can register callbacks that are executed by the power manager. Before entering sleep mode, the manager queries all devices to determine if they are willing to suspend (a device might refuse if it were in the middle of an IO transaction, for example); if not, the device must state how much time it expects to need before being able to accept a suspension. When all the devices are ready, they are called again to save their current state before the suspension. When the processor exits sleep mode, the OS reestablishes its previous position (stack and registers), and the power manager again calls the devices, this time for reinitialization.

The power manager module is also used when changing the processor clock speed. For a speed increase, first all devices are suspended, then the DRAM and flash timing parameters are changed, followed by the actual processor speed, and finally the devices are resumed. A speed decrease is the same, except the DRAM and flash timing parameters are changed after the processor speed. Except for the LCD, suspending and resuming the devices is necessary only to prevent temporary glitches, and adds about 400 μs to the base switching time of 125 μs.

4.2 Virtualizing Itsy devices

We wanted to run several different high-level software environments on the Itsy. In order to simultaneously support these disparate environments, we built *sessions*, a simple device sharing model with reasonable default behavior and modest implementation effort. In contrast to a unified windowing system that demands compliance from all applications, the Itsy sessions model supports the peaceful coexistence of distinct yet concurrently-executing “worlds,” each of which effectively sees its own virtual Itsy. For example, we can concurrently run full-screen standalone applications, virtual consoles, the X Windows desktop, Java environments, and the Squeak Smalltalk-80 system.

4.2.1 Sessions

A session consists of one or more processes. At any given time, a single session may be distinguished as the active or *foreground* session. The processes in the foreground session receive physical input events, such as button presses or touchscreen samples. These processes also have exclusive (or preferred) access to physical output devices. For example, the session's active framebuffer, if any, is mapped to be visible on the LCD.

Any number of background sessions may co-exist with the foreground session. Processes in background sessions do not receive physical input events, and their output is typically hidden. For example, background audio output is attenuated or muted.

4.2.2 Device sharing model

When a process in a session opens a physical device, it only obtains access to a session-filtered instance of that device. Input generated by each physical device is replicated to all instances in the foreground session. Output is merged from all instances within the foreground session. Output from background sessions may be dropped, attenuated, stored, or handled specially, depending on the device.

Each physical device also exports a *raw* interface that is not session-filtered. Applications requiring continuous access to a device may use the raw interface, bypassing the coherent sharing support offered by the session-filtered interface. For example, a speech-recognizer application running in the background could snoop on all audio input, translate it, and offer the transcribed text to any application in the foreground session. Applications may be made “session-aware” by explicitly monitoring events (such as changes in foreground/background status) via a pseudo-device interface.

4.2.3 Session manager

A user-level process serves as a session manager to coordinate which virtual Itsy instance is currently mapped to the physical hardware. The session manager obtains information about all active sessions and manipulates them through the raw session interface. For example, the session manager can dynamically change the foreground/background status of a session, and can send suspend, resume, or terminate signals to all processes in a session. Figure 1 shows a session manager running.

4.3 Application environments

We support several different high-level graphical user interfaces on the Itsy. These environments all had existing applications that were binary compatible with the Itsy, and all provide the capability for software development on desktop machines. However, as will be discussed in Section 4.4.1, the major impediment to application development for Itsy is the translation of the desktop user interface itself.

The first application environment available on Itsy was Squeak [7], a portable Smalltalk-80 system. Squeak applications have been developed on both Windows and Macintosh systems for Itsy. Existing applications such as a multi-voice music synthesizer run with no changes. A heap image can be saved on either of these systems and run on Itsy merely by changing the display depth and clearing a flag in the SoundPlayer.

The Itsy also supports the *Kaffe* Java system from Transvirtual Technologies. Kaffe includes a Java Virtual Machine and a set of classes that handle graphics, file management, input/output, and networking. The Java applications that we have run include a web browser (based on the ICE browser from ICEsoft), a mail reader, an unmodified Java applet from Datek that provides streaming real-time stock quotes, and a home-grown front end for the Crafty chess program.

Finally, the Itsy can run an X server and a full complement of X client applications, such as xterm, twm, and xv. The Itsy X server code has been integrated into the development branch of XFree86 as of version 3.9.17. Improvements were made to XFree86 to support framebuffers such as the one used for the Itsy. This new code uses about 500 KB less code space than the previous version, yet supports all the depths that the server needs. In addition to being smaller, it is faster than the old version on systems without hardware accelerators.

4.4 Experiences with user interfaces

The previous sections described several application environments that are supported on the Itsy. Although applications in each environment are binary compatible with other ARM-based platforms, or bytecode-compatible with other Java and Smalltalk systems, many desktop user interface (UI) assumptions do not apply to small devices. Additionally, as portable devices shrink further, stylus-based input becomes difficult and other methods will be needed. We have therefore experimented with two non-traditional UIs: speech-based input and output, and gesture-based input.

4.4.1 Translating desktop user interfaces

Our initial attempts at using Squeak's graphical user interface (GUI) on Itsy reminded us just how difficult it is to jump off the desktop without crashing onto the floor. The desktop interaction style is centered around a number of assumptions that are inappropriate for a handheld or wearable device.

The 320×200-pixel Itsy screen is much smaller than that expected by Squeak. Individual windows are often too wide and command menus are too long to be completely visible. This issue was addressed by having the Smalltalk virtual machine support a larger 640×400-pixel virtual display that can be scrolled about the physical display using the cursor control.

Squeak user input makes extensive use of a three-button mouse, relying heavily on the left mouse button. This GUI is mapped onto the Itsy's touchscreen and buttons. When the stylus touches the screen near the cursor, Squeak sees the left button pressed at the current cursor position. As the stylus moves, the cursor tracks it and Squeak sees a mouse drag with the left button held down. On the other hand, when the stylus touches the screen beyond some threshold distance from the cursor, Squeak sees mouse motion with no buttons pressed. To invoke the center and right mouse buttons on Itsy requires explicit button pushing in conjunction with use of the stylus.

While these conventions made the Squeak desktop GUI usable, it is by no means “easy to use.” Handheld interfaces that require precision pointing and tapping are difficult to master. A user cannot be expected to move a stylus on a handheld device with the same ease and accuracy that he moves a mouse on a desktop. Additionally, while a desktop GUI has the user's full attention for extended periods, the only portable application domain that can demand this is games. Clearly, as the Palm OS demonstrates, GUIs must be redesigned to be successful on small or handheld devices.

4.4.2 Speech

One promising interface for a tiny device is speech. For output, the Itsy uses DECtalk, a commercially available text-to-speech engine. Speech input is a much more challenging problem. Two speech recognition engines have been ported to Itsy. The first, from TalkSoft, provides speaker-independent small-vocabulary command-and-control speech recognition in a small memory footprint. We have successfully used DECtalk and TalkSoft to build a speech-based multimedia email program. Although not a full implementation, this program has successfully demonstrated the feasibility of speech-based interaction in realistic environments, even using the built-in microphone in a crowded room.

The second speech recognition engine that has been ported to Itsy is from Dragon Systems, Inc. There are two flavors of the engine: one is a speaker-independent, grammar-based, continuous speech, command-and-control engine, the other one is a speaker-dependent, continuous-speech, dictation engine with a 30,000-word vocabulary (this is the engine that NaturallySpeaking uses at its core). Together, they offer the potential for a rich speech-only user environment.

4.4.3 Gesture

Most desktop computer input methods rely on physical manipulation of a tethered object such as a keyboard or mouse. As systems shrink, motion of the system itself can be exploited for user input. Although tilting a handheld computer to navigate through a document has long been anticipated, it is only recently that sensors have become small and cheap enough that they can be embedded in a handheld device and this vision realized.

We extended the basic idea of tilt-to-scroll to include the use of gestures to issue commands. This user interface, which we call *Rock 'n' Scroll*, lets users gesture to scroll, make selections, and issue commands without resorting to any other input method [8].

A photo album application demonstrates this interface. Tilting the album on either axis scrolls through miniatures of the photographs until the user finds a picture of interest. A gentle fanning gesture zooms in on that picture. Additional fanning gestures can be used to step through the rest of the album, return to the miniatures, or disable/enable scrolling. The pictures are available in both landscape and portrait mode; holding the unit in the new orientation for a few moments is all that is required to select it.

Experiments with a prototype system on an Itsy v1 demonstrated that users could quickly learn to operate Rock 'n' Scroll and offered some insight into user expectations. These positive results, as well as improvements in the sensor technology, encouraged us to incorporate this as a standard input method on Itsy v2, where we have included a two-axis micromachined accelerometer that can sense fore-aft and left-right tilt.

System	OS	CPU	CPU (MHz)	V2.1 MIPS
SGI Indigo2	IRIX 6.2	MIPS R10000	195	214
Pentium P5-120	Windows 95	Pentium P5	120	159
Itsy	**Linux**	**ARM SA1100**	**206**	**150**
Pentium P5-100	Windows 95	Pentium P5	100	138
Mac PowerPC 604	MacOS 7.5.2	PowerPC 604	120	137
Casio E-100	Windows CE 2.11	MIPS VR4121	133	30
Palm III	Palm OS	68328	16	0.8

Table 3: Dhrystone V2.1.

System	CPU (MHz)	espresso	li	compress	geo. mean
Pentium-90	90	111.7	47.9	76.6	84.1
Itsy	**206**	**105.0**	**47.9**	**74.8**	**83.2**
Pentium-75	75	92.0	52.6	68.4	66.2

Table 4: Subset of SPECint92 benchmarks.

5 Performance

To fulfill the project goals, we needed sufficient processing power and memory capacity to run next-generation applications and user interfaces, as well as sufficient battery life to enable us to run realistic user interface experiments. We performed a number of tests to assess the degree to which we met these goals.

5.1 Processing power

Flawed as they may be, standard benchmarks provide a means for comparing widely differing systems. Due to the lack of hardware floating-point support on the SA-1100, we concentrated on integer performance. Table 3 shows the results of the Dhrystone V2.1 benchmark for Itsy and a few other systems [9, 10, 11]. Three of the SPECint92 benchmarks were also run, and the results are shown in Table 4 [9].

By interpolation, Itsy runs the Dhrystone benchmark with performance similar to that of a Pentium P5 running at 110 MHz. Although the Itsy's performance is a few years behind that of desktop computers, it performs much better than contemporary handhelds. For example, Itsy's Dhrystone performance is five times better than the Casio E-100 handheld, and almost 200 times better than a Palm III.

The SPECint92 benchmarks are larger and more complex than Dhrystone, and Itsy performs more poorly on them relative to a Pentium-class processor due to its smaller caches. For this SPECint92 subset, Itsy's performance is comparable to a Pentium running at 90 MHz.

As these results suggest, Itsy has the performance to run real programs normally associated with workstations. On a 206 MHz Itsy, the DECtalk text-to-speech engine runs at only a 47% system load, the command-and-control engines can process speech considerably faster than real time, and the

Application	Static size (MB)	Dynamic size (MB)
X libraries	1.9	–
X client binaries	1.8	–
X server	0.7	2.2
DECtalk	1.0	1.5
TalkSoft	0.4	1.5
Doom binary	0.5	3.0
Doom data file (wad)	4.1	–
Dragon dictation	16.7	25.0

Table 5: Application memory requirements.

Dragon NaturallySpeaking continuous-speech dictation engine runs about 2.4 times slower than real time.

5.2 Memory usage

As mentioned in Section 3, Itsy has 32 MB of DRAM and 32 MB of flash on the motherboard. For historical reasons, the flash is partitioned into two sections. The first partition is 4 MB and contains the boot monitor (400 KB), the compressed kernel (370 KB) and about 3 MB for the compressed root filesystem. The second partition is 28 MB and is used for the flash-based filesystem described in Section 4.1.2.

The DRAM is used for kernel and program execution, a buffer cache, and for the ramdisk containing the root filesystem. At boot time, the compressed root filesystem is uncompressed into the ramdisk. It occupies about 8 MB uncompressed and contains mostly program binaries (3 MB), libraries (4 MB) and configuration files ($<$ 1 MB). Another 5 MB are used by the kernel, daemons and session manager, leaving about 19 MB available for user programs.

Table 5 shows the memory requirements of some non-trivial applications. With 28 MB of flash and 19 MB of DRAM available, all except Dragon dictation fit easily. The X package is surprisingly small: the 4.4 MB total includes the server, libraries (such as X11, Xt, Xaw, Xpm, and Xmu), and some binaries (such as twm, oclock, xv and xterm). The server executable file is only 700 KB, and the runtime size is a little more than 2 MB. Dragon continuous-speech recognition, on the other hand, requires a daughtercard with extra DRAM (32 MB), since its runtime memory footprint is about 25 MB.

5.3 Energy consumption

In this section, we evaluate the energy consumption of the Itsy using two scenarios: intermittent and continuous use. While we focus primarily on power consumption, which depends on the hardware, the system software, and the application, we also report battery life, which depends mainly on the battery size. We will show that the Itsy attains a level of performance comparable to laptop computers while using an order of magnitude less power.

The *intermittent-use scenario* represents bursts of use for short time intervals throughout the day. Here, battery lifetime is determined predominantly by the low-power operating modes provided by the Itsy. The *continuous-use scenario* considers applications such as multimedia playback, where battery life is determined predominantly by the power consumed as the Itsy executes the application.

5.3.1 Methodology

Accurately assessing the average power of a system is challenging. When an Itsy runs under battery power, the Li-ion battery voltage decreases gradually from 4.1 V to roughly 3.4 V, then falls quickly as the battery completely discharges. This behavior affects measurements because the power supply efficiency varies with battery voltage. For example, in sleep mode the power drain from the battery decreases 6% – 9% as the battery voltage decreases, while at high loads, it is less dependent on the battery voltage (2% – 4% change for most applications).[3]

Given these constraints, we started each experiment with a fully charged battery. During the test, the Itsy would periodically send a character followed by a new line over the serial line to a host computer. When the characters stopped, the battery was considered exhausted.[4] We define the battery lifetime as the measured time plus one half of the character transmission interval for each experiment. The calculated error introduced by this measurement procedure is less than 1% for all of the experiments [12].

In order to measure the average power, we recorded the average battery current and voltage continually during each experiment, multiplied each pair together to calculate the power and averaged over all of the meaningful samples (the first and the last were discarded). The worst case calculated error introduced by this measurement procedure is less than 9% for all of the experiments, 5% if deep sleep is excluded [12].

The power and battery lifetime data are presented in Table 6. Data for the intermittent-use scenario is given in rows 1 to 9, and the continuous-use scenario in rows 10 to 16. The idle percentage column shows how often the processor was in idle mode, and provides an indication of the computational load on the system. Battery capacity is the product of measured power and lifetime. The sleep mode experiments were run on three Itsy units; the rest of the experiments were run on five units, except for experiments 1 and 6 which were both run only on a single unit. Previously unused battery packs were used for each set of experiments. Some unit-to-unit variation was observed, as were run-to-run variations in battery capacity. Further variations were introduced by the time-varying nature of the chosen benchmarks. Taken together, we expect our reported results to be repeatable to within $\pm$9% over typical variations in hardware and test conditions. The small variation in calculated battery capacity is consistent with this estimate.

5.3.2 Intermittent-use scenario

The intermittent-use scenario represents traditional personal information manager applications, such as a calendar or a phone list. The system is mostly inactive, and thus, the energy consumption is

[3]Power is the product of voltage and current. Energy is power integrated over time.

[4]Note that while an Itsy cannot wake up and transmit a character when the battery drops below about 3 V, the DRAM data can be retained in sleep mode down to 2.7 V, and operation resumed after re-charging the battery.

#	Experiment	Clock Speed (MHz)	Processor Idle (%)	System Power (mW)	Battery Lifetime (h)	Battery Capacity ($W \cdot h$)
	Intermittent-use Scenario					
1	system idle, run mode	206	0	540	3.8	2.03
2	system idle, idle mode	206	95	100	23.0	2.29
3	system idle, idle mode	133	95	82.4	28.0	2.31
4	system idle, idle mode	59	95	69.4	33.3	2.31
5	system idle, idle mode, low voltage	59	95	55.6	41.5	2.31
6	sleep mode, static LCD image	–	–	27.5	81.5	2.24
7	sleep mode, mem. DC*	–	–	11.6	203.0	2.35
8	sleep mode	–	–	8.39	279.0	2.34
9	deep sleep mode	–	–	5.41	429.5	2.32
	Continuous-use Scenario					
10	playing wav file	206	93	311	7.08	2.20
11	playing wav file	59	83	280	7.92	2.22
12	text to speech†	206	53	397	5.50	2.18
13	text to speech†	74	<1	397	5.53	2.20
14	text to speech,† low voltage	74	<1	354	6.24	2.21
15	dictation,‡ mem. DC*	206	<1	738	2.83	2.09
16	MPEG-1 with audio	206	16	822	2.51	2.06

* Memory daughtercard (32 MB DRAM) † DECtalk ‡ NaturallySpeaking from Dragon Systems, Inc.

Table 6: The power consumption, battery lifetime, and effective battery capacity of Itsy v2.

dominated by long periods in low-power operating modes (described in Section 4.1.3).

When the Itsy has no work to perform, our Linux kernel places the StrongARM processor into idle mode. To determine an upper bound on idle mode power savings, we tried a kernel that disabled this behavior (experiment 1) and compared it to our usual kernel (experiment 2). In both experiments, the only processes running on the Itsy were the session manager (see Section 4.2.3) and several daemons. In experiment 2, the processor was in idle mode for 95% of the clock cycles. The data show an 81% power savings by using idle mode when the kernel is idle.[5]

In experiments 3 – 5, the same low background workload was used again, but the idle mode parameters were varied. Comparing experiments 2 – 4, we observe that simply lowering the clock speed yields a significant power savings. Although in idle mode the clock driving the processor core is disabled, the clock driving the peripheral circuits is not. Therefore, the power still decreases with the clock speed. Since a large part of the peripherals also runs off the core voltage, we can reduce the core voltage to further reduce the power. As shown by experiments 4 and 5, at 59 MHz dropping the core voltage from 1.5 V to 1.23 V reduces system power by approximately 20%, and over 40% from

[5] In fairness, for a processor without an idle mode, some portion of this power savings could be obtained by a rewrite of the kernel idle routine.

the 206 MHz base case. Clearly, the clock speed and the voltage level can have a significant impact on power consumption. We examine this further in the next section.

Much more power can be saved by putting the system into sleep mode. As discussed in Section 4.1.3, there is a wakeup delay and functionality cost that makes it inappropriate when either continuous I/O activity or rapid response to inputs is required. However, sleep mode could be effectively used by, for example, a phone book application to display a static image of a phone number. In this case, sleep mode uses about half the power of idle mode (compare experiments 5 and 6). Sleep mode could also be effectively used during user interactions if the 10 ms wakeup latency is tolerable. When a static image is not needed, turning off the display saves an additional 58% (experiment 8). However, adding 32 MB of DRAM on a daughtercard (experiment 7) requires an additional 3.2 mW for self-refresh. Deep sleep mode (experiment 9), requires the least power, but does not preserve the contents of the DRAM.

The ability to control which components are enabled when the system has been put to sleep is an important feature, for it avoids mandating that software developers use the hardware in predetermined ways. For example, if the functionality provided by the CODEC is not required for an application, it can be turned off, saving about 60 mW. Similarly, 3 – 10 mW can be saved by turning off the RS-232 port.[6] The intensity of the backlight can also be controlled by software, varying its power cost from 0 to 400 mW.

5.3.3 Continuous-use scenario

The continuous-use scenario represents applications that impose substantial continuous computing workloads. The processor spends relatively little time idle, and power consumption is determined largely by the running application.

Playing an audio file in wav format is our least demanding sample application. Experiments 10 and 11 show how we can exploit the very high idle percentage at 206 MHz by reducing the clock speed to 59 MHz for a 10% power savings. By comparison to the corresponding idle mode experiments (2 and 4), it can be seen that most of this power reduction is due to more power efficient idling, and the power actually devoted to the application appears to be only weakly dependent on clock speed.[7]

The power used by the text-to-speech synthesis tests does not depend at all on clock speed. Experiment 12 shows 53% idle cycles at 206 MHz. As the clock speed is reduced to 74 MHz (experiment 13), minor sound artifacts (occasional clicks) begin to appear and the idle cycles disappear, but the power consumption does not change. This behavior is perhaps counterintuitive, and requires some explanation. For CMOS circuits without static power dissipation, the energy required for a given quantum of computation is independent of clock speed. However, real circuits have finite static power dissipation, so the slower (hence longer) you run, the more energy a computation takes. Therefore, in experiment 13, the gains from reduced idle time roughly offset losses from increased static power dissipation. Other factors, such as the energy cost of memory accesses,[8] also play an important role. The key, however, is that if the clock speed is low enough, the core voltage of the processor can be reduced, which will result in a power savings: experiment 14 shows a power reduction of about 10%.

[6] Actual savings depends on the characteristics of the connected host.
[7] About 2/3 of the power difference between this benchmark and idle mode is used by the speaker.
[8] On the Itsy, memory accesses are less efficient at low clock speeds.

Dragon NaturallySpeaking (experiment 15), a continuous speech recognizer, really stretches the Itsy's capabilities. As described in Section 5.1, the recognizer runs about 2.4 times slower than real time, so there are no idle cycles. The battery life of 2.83 hours thus corresponds to a bit over an hour of actual dictation. Note too that the application's memory requirements force us to add a memory daughtercard. Powering the DRAM on the daughtercard, accessing it, and refreshing it was found to consume about 10 mW.

Our final application, 16 frame-per-second MPEG video with sound (experiment 16), is the most power hungry. The 16% idle value suggests that there will be relatively little opportunity for clock speed reduction. Still, even with an 822 mW power drain, video can be played 2.5 hours on a single battery charge. Note that the use of an external headset will greatly reduce the power required: we have measured a difference of up to 40% for playing audio files at different volumes.

6 Related work

Researchers wishing to go beyond using laptop computers to explore off-the-desktop computing have used a variety of systems.

One of the best known is the *ParcTab* [13] developed at Xerox PARC. This is a badge computer, conceived of as a leaf that is wirelessly connected to a tree of computing infrastructure. Each 105 mm $\times$ 78 mm $\times$ 24 mm ParcTab weighs 215 g, and includes an IR link, a 12-MHz 8-bit microprocessor, 128 KB of RAM, and a 128 $\times$ 64 pixel monochrome LCD. Users interact with the ParcTab using a small number of buttons and the touchscreen; text can be input using either an on-screen keyboard or the Unistroke character alphabet. While Itsy shares some attributes with a ParcTab, Itsy is significantly lighter, more powerful, and expandable.

A number of groups working in wearable computing have tried to overcome the performance and expansion limitations of smaller machines by using PC/104 hardware [14]. Systems based on x86 processors are assembled by stacking 110 mm $\times$ 110 mm $\times$ 15 mm modules, and a number of fanny-pack-sized wearable systems have been built. While the latest PC/104-based systems can be roughly twice as fast an Itsy, their size and power demands preclude their use in handheld devices. Although recent low-power x86 processor announcements from several companies have renewed interest in small, PC-compatible systems, we believe that alternative architectures will continue to have a significant power advantage.

Commercial handhelds such as the Apple Newton and the Palm have also proven to be useful research tools. While the size and weight of the Newton stretch the bounds of what we now consider a handheld, a reasonable development environment and ready availability led to its wide use. The Palm's size, weight, price, and tools make it very attractive. However, its limited processor performance and memory size preclude many applications that can be hosted on Itsy. While the Handspring Visor offers expansion capabilities, it offers no additional performance.

While Itsy was never made commercially available, it did inspire the creation of the Compaq iPAQ H3600 Pocket PC [15]. This handheld incorporates much of what made Itsy an attractive research platform, offering a fast StrongARM processor, a large memory, and easy expandability. Unlike Itsy, the iPAQ has a 3rd-generation reflective color display, making it an appealing applications delivery platform. In a departure from previous handheld products, the iPAQ is flash-based rather than ROM-based, allowing alternate operating systems to be installed. An Open Handhelds website [16] was recently announced to encourage and facilitate the creation of open source software for use on hand-held and wearable computers. While initially sponsored and hosted by Compaq, outside participation has grown and additional sponsors are expected. A Linux 2.4 port and other utilities for the iPAQ are available now at this site.

With the proliferation of ever smaller and more powerful devices comes the need to save energy. Indeed, researchers have long been working on methods to reduce power consumption [17, 18]. However, most efforts either have been limited by the design of the systems they performed their studies on [18, 19, 20, 21], or have simulated their results [17, 22]. We see the Itsy as a significant step forward with its abilities of fine-grain control and measurement of power usage.

7 Conclusions

The experience of building Itsy confirms that a system is only as good as its weakest component. Striking the right balance during component selection is key to building small devices. While the size is bounded by a few fundamental choices such as processor and display, and the battery to power them, a tyrannical approach to the selection of the remaining components and features is necessary to prevent bloat. For example, a thumbwheel encoder takes the same area as 8 MB of flash. Our experience with version 1 drove us to provide eight times as much flash on the version 2 system, bringing it in balance with the amount of DRAM. Our version 1 power system was kept simple for the sake of fast implementation, but provided sufficient insight to build a version 2 system that left little room for further improvement.

However, by far the most common user complaint about the Itsy hardware is the difficulty of reading the display in the absence of sunlight or bright room light. There are a number of contributing causes. While our display has a better contrast ratio than most passive matrix transflective LCDs, the small pixel size encourages the use of tiny fonts, which are less readable. The situation is exacerbated by our resistive touchscreen, which cuts display brightness by about 25%. The touchscreen also has a coating that reduces glare from overhead lights, but noticeably reduces image sharpness, particularly with small fonts.

There are many other display and touchscreen technologies available in today's market. Most, however, are more power-hungry, are difficult to obtain in low volume for a research program like ours, and/or have other limitations that would make them poor candidates for use in Itsy. On balance we believe our display and touchscreen choices were a satisfactory compromise.

The Itsy pocket computer has demonstrated that it is possible to put desktop power in a palm-sized system that fits in the user's pocket. The Linux operating system has proven to be a good choice, as it has allowed significant implementation flexibility and provided a familiar platform for researchers moving from the desktop. However, as our initial experience using Itsy confirmed, having the cycles to run desktop applications is not sufficient, because many do not translate well to tiny devices. In particular, new approaches to user interfaces and interaction methods are needed.

The Itsy's ability to provide continuous speech recognition for over an hour demonstrates that it is already capable of supporting the most demanding next-generation applications. At the same time, its flexible power control mechanisms let it minimize power consumption and provide over a day of service for many user scenarios. The ease by which the hardware may be extended has been demonstrated by a number of projects, including a PCMCIA daughtercard, CMOS cameras, and the initial version of the Rock 'n' Scroll electronics. Finally, several ongoing efforts demonstrate the appeal and success of the Itsy pocket computer as an experimental platform. These efforts include the work at CMU on wearable computers [23], at EPFL on CMOS cameras [24], at the University of Colorado at Boulder on energy management and optimization [25], and by our commercial partners on software radio and entertainment.

8 Acknowledgements

A large number of people contributed to the success of the Itsy project. David Chaiken designed a USB daughtercard for the Itsy v1 which became the basis for the onboard USB hardware in Itsy v2. Peter Dettori wrote USB drivers for the Itsy and for both Linux and Windows 2000 hosts. Tim Rowledge did the initial port of Squeak. Dirk Grunwald, Godmar Back, and Harold Chaput, together with Transvirtual, ported Kaffe. Jim Gettys' vision and powers of persuasion resulted in Keith Packard's port of the X Windows system to the Itsy. Matthew Schnee, Tom Kopec, and the rest of Compaq's former DECtalk group ported DECtalk. Dragon Systems, Inc. and TalkSoft both ported their speech recognition software to the Itsy under contract. Sharon Perl ported the scribble handwriting-recognition software to the Itsy. Jennifer M. Anderson wrote the initial battery monitor device driver. Barton Sano created multimedia content for the Itsy. Wayne Mack provided extensive assistance in the manufacturing of the Itsy units. Fran McGroary-Dehn managed the external relationships. We are also grateful to Annie Warren Bedichek, Glenn Cooper, Drew Kramer, and Jason Wold for operational support. Finally, we would like to acknowledge the enthusiastic support of our management during this major project.

References

[1] R. Stephany, K. Anne, J. Bell, G. Cheney, J. Eno, G. Hoeppner, G. Joe, R. Kaye, J. Lear, T. Litch, J. Meyer, J. Montanaro, K. Patton, T. Pham, R. Reis, M. Silla, J. Slaton, K. Snyder, and R. Witek. A 200MHz 32b 0.5W CMOS RISC microprocessor. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 238–239,443, February 1998.

[2] Marc A. Viredaz. The Itsy pocket computer version 1.5: User's manual. Technical Note TN-54, WRL, Compaq, Palo Alto, CA (USA), July 1998.

[3] Keith I. Farkas, Jason Flinn, Godmar Back, Dirk Grunwald, and Jennifer M. Anderson. Quantifying the energy consumption of a pocket computer and a Java virtual machine. *SIGMETRICS*, June 2000.

[4] Intel. *Intel® StrongARM® SA-1100 Microprocessor: Developer's Manual*, August 1999.

[5] *Universal Serial Bus Specification*, September 1998. Revision 1.1.

[6] T. Kurumisawa, A. Ito, S. Yamazaki, and S. Iino. High-performance ultra-low-power 640 $\times$ 200 reflective STN display system using four-line selection method. In *Society for Information Display International Symposium Digest of Technical Papers*, pages 351–354, May 1996.

[7] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: the story of Squeak, a practical Smalltalk written in itself. In *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 318–326, October 1997.

[8] Joel F. Bartlett. Rock 'n' Scroll is here to stay. *IEEE Computer Graphics and Applications*, 20(3):40–45, May/June 2000.

[9] Performance Database at the Netlib Depository. http://www.netlib.org/performance.

[10] Dhrystone for Palm devices. http://vvv.geocities.co.jp/SiliconValley/1324/Dhrystone.html.

[11] Windows CE Benchmarks. http://netsurf.pdacentral.com/wincelair/wincespeed.htm.

[12] Marc A. Viredaz and Deborah A. Wallach. Power evaluation of Itsy version 2.3. Technical Note TN-57, WRL, Compaq, Palo Alto, CA (USA), October 2000.

[13] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. An overview of the ParcTab ubiquitous computing experiment. *IEEE Personal Communications*, pages 28–43, December 1995.

[14] PC/104 Consortium. *PC/104 Specification*, June 1996. Version 2.3.

[15] Compaq Computer Corporation. www.compaq.com/products/handhelds/pocketpc/.

[16] handhelds.org. www.handhelds.org.

[17] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *First Symposium on Operating Systems Design and Implementation*, pages 13–23, November 1994.

[18] Mani B. Srivastava and Anantha P. Chandrakasan. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on Very Large Scaale Integration (VLSI) Systems*, 4(1):42–55, March 1996.

[19] Jacob R. Lorch and Alan J. Smith. Scheduling techniques for reducing processor energy use in MacOS. *Wireless Networks*, 3(5):311–324, October 1997.

[20] Carla Ellis. The case for higher-level power management. In *The 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, pages 162–167, Rio Rico, AZ (USA), March 1999.

[21] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th Symposium on Operating Systems Principles*, pages 48–63, Kiawah Island Resort, SC (USA), December 1999.

[22] Trevor Pering, Tom Burd, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *IEEE Symposium on Low Power Electronics*. IEEE Symposium on Low Power Electronics, 1995.

[23] The Wearable Group at Carnegie Mellon. http://www.cs.cmu.edu/~wearable/itsy.

[24] Olivier Saudan. Interfacing a mobile camera with the Itsy pocket computer. 7th semester project, LAMI, EPFL, Lausanne (CH), February 1999.

[25] Dirk Grunwald, Philip Levis, Charles B. Morrey III, Michael Neufeld, and Keith I. Farkas. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation,* San Diego, CA (USA), October 2000. To appear.