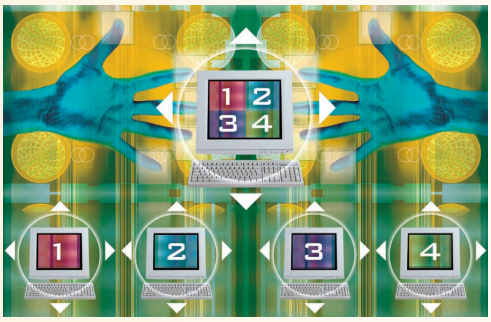
Itsy: Stretching the Bounds of Mobile Computing

A prototype pocket computer that has enough processing power and memory capacity to run cycle-hungry applications such as continuous-speech recognition and real-time MPEG-1 movie decoding has proved to be a useful experimental tool for interesting applications, systems work, and power studies.



William R. Hamburgen
Deborah A. Wallach
Marc A. Viredaz
Lawrence S. Brakmo
Carl A. Waldspurger
Joel F. Bartlett
Timothy Mann
Keith I. Farkas
Compaq Computer Corporation, Corporate Research

The advent of fast, power-thrifty microprocessors has made possible pocket-size computers with performance approaching that of desktop PCs. This new class of mobile computers enables applications and user-interface modalities not feasible with traditional personal digital assistants and cell phones, while placing new demands on batteries and power management. We built Compaq's Itsy pocket computer research prototype to explore the possibilities, demands, and limitations of mobile computing.

Our primary hardware goals were to attain high performance with minimal power consumption, size, and weight. At the same time, we needed a rich feature set to support user-interface and applications research and the flexibility to easily add new capabilities. To meet these goals, we used daughtercards to provide Itsy with comprehensive expansion capability. Fine-grain hardware control supports flexible power management and monitoring. Developers can use the Linux operating system with extensions for a flash file system, resource sharing, and power management to rapidly prototype operating system extensions and new applications. Itsy has sufficient processing power and memory capacity to run cycle-hungry applications such as continuous speech recognition, a full-fledged Java virtual machine, and real-time MPEG-1 movie decoding.

HARDWARE

We began our hardware effort by constructing 75 systems that we used to start software development.[1] The experience we gained in building and using these systems influenced our subsequent design, Itsy v2. Figure 1 shows this design's general architecture, and Table 1 lists its primary specifications.

Our design focused on two goals: packing maximum performance into a unit that people can comfortably carry all day in a pocket or purse and enabling easy customization and extension of the system hardware and software. Itsy is only slightly larger than a credit card, but it incorporates these other desirable features. Criteria such as cost or suitability for volume manufacturing, which are critical for commercial products, played no significant role.

A small system's battery and display are generally its largest and heaviest components, so they establish a lower bound on the system's size and weight. For Itsy, we selected a lithium-ion cell just large enough to provide a full day of intermittent use, and the smallest available LCD with sufficient resolution for a rich graphical interface.

We ruthlessly excluded any features or components that would bloat the system. For example, version 1 users wanted a thumbwheel encoder, a cursor button, a good speaker, and a stylus slot, but because all four of these features would not fit within our space budget, we excluded the stylus slot. Similar considerations led us to omit a bulky stereo headphone jack and codec in favor of a smaller monaural headset jack and a monaural codec that includes a touch-screen controller. Finally, a radio transceiver was clearly desirable, but we found no obvious best choice to include in the base system. Therefore, we relegated experimental radios to the daughtercard or serial interfaces. As a result of these choices, the complete Itsy is only 70 percent larger in volume than it would be if it contained only the battery and display.

0018-9162/01/$10.00 © 2001 IEEE

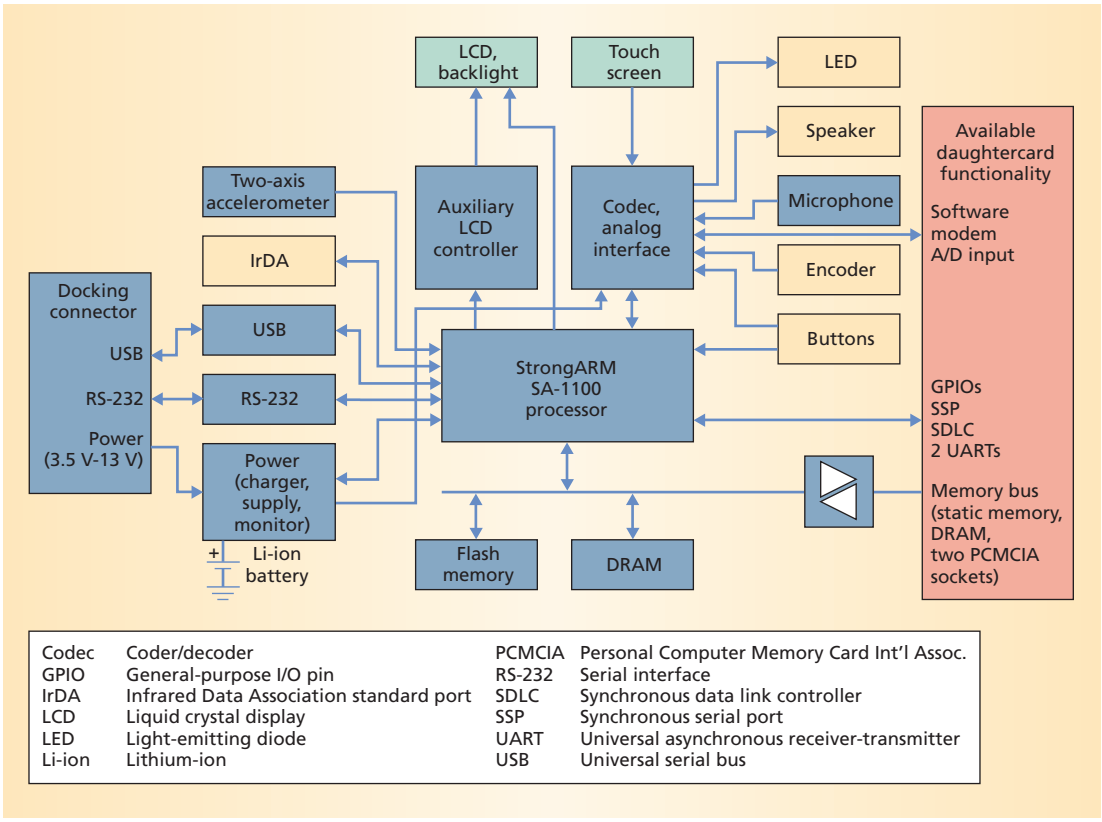


Figure 1. Itsy v2 architecture. Most of the hardware is implemented on the motherboard (depicted in blue). The LCD and the touch screen attach to the motherboard via dedicated connectors. Other parts of the hardware (shown in yellow) are implemented on a separate flexible circuit board. A daughtercard connector is available to interface with additional hardware. Daughtercards can use any of the features listed in the red box.

Processor

The StrongARM SA-1100 is a low-power 32-bit microprocessor that implements the ARM instruction set.[2,3] This processor was a clear choice because its integer performance approaches that of desktop processors, but it uses an order of magnitude less power. It also provides a useful collection of peripheral devices, as well as power-saving features that researchers can exploit. To minimize energy use, the StrongARM supports software-controllable clock frequency and two low-power modes: idle and sleep. In idle mode, the clock to the processor core is gated off, saving power due to the CMOS circuit technology, while the rest of the chip remains powered and all peripherals remain enabled. In sleep mode, most of the processor is unpowered, and only the real-time clock and the wake-up circuit remain enabled. Optionally, the system clock can remain enabled for faster wake-up.

Display

In contrast to the usual power-saving passive-matrix displays developers commonly choose for small systems, Itsy's LCD has some particularly useful characteristics. Its 0.18-mm pixel pitch is 25 to 30 percent smaller than the typical pitch of small matrix displays, permitting dense text and crisp graphics. Its multiline addressing technology provides higher contrast than is typical of a passive display. Finally, the LCD's built-in 1-bit-per-pixel memory and a programmable-logic-device (PLD) auxiliary controller make it possible to display a static monochrome image while the processor is in sleep mode.

Memory

The most frequent complaint from Itsy v1 prototype users was the limitation of having only 4 Mbytes of flash memory, so we started the v2 motherboard design with the memory system. As Figure 2 shows, we chose a micro-ball-grid array package for the flash instead of a peripheral lead package. Although the micro-BGA calls for a more complicated assembly process, it offers three times the mounting density. We chose the motherboard width to allow dense tiling of the flash across the board's full width. We arranged the DRAMs on the opposite side to match the flash tiling. Itsy v2 has twice as much DRAM and eight times as much flash as Itsy v1, with only a 3 percent increase in board area.

Table 1. Itsy v2 specifications (without daughtercard).

Component or characteristic	Specification
Processor	StrongARM SA-1100 (59 MHz–191 MHz)*
Dynamic RAM (DRAM)	32 Mbytes (50 ns, extended data out)
Flash memory	32 Mbytes (90 ns)
Processor-core voltage	1.5 V or 1.23 V (selectable)
Main voltage	3.05 V
LCD	320 × 200 pixels, 15 gray levels
Battery	Rechargeable Li-ion (2.2 W × h)
Size	118 mm × 65 mm × 16 mm
Weight	130 gm

*The processor's manual guarantees operation up to a 191-MHz core frequency at 1.5 V, but all Itsy systems built to date function correctly at 206 MHz and even higher.

After choosing the number of DRAM chips (4) and their individual capacity (64 Mbits), we selected the device width. Our choices were to implement one bank using four 8-bit parts (32 Mbytes per bank) or two banks of two 16-bit parts each (16 Mbytes per bank). Because the StrongARM only supports up to four banks, the one-bank option offers better expansion capabilities. However, we chose the two-bank design for two reasons. First, refreshing the 16-bit parts consumes less power—about 5.2 mW versus 9.5 mW for 32 Mbytes—because 16-bit parts have a different internal topology. Second, reading data out of the 16-bit parts consumes less power because only two parts are active instead of four. For example, copying memory using eight-word bursts at 206 MHz requires about 490 mW rather than 630 mW.

Power system

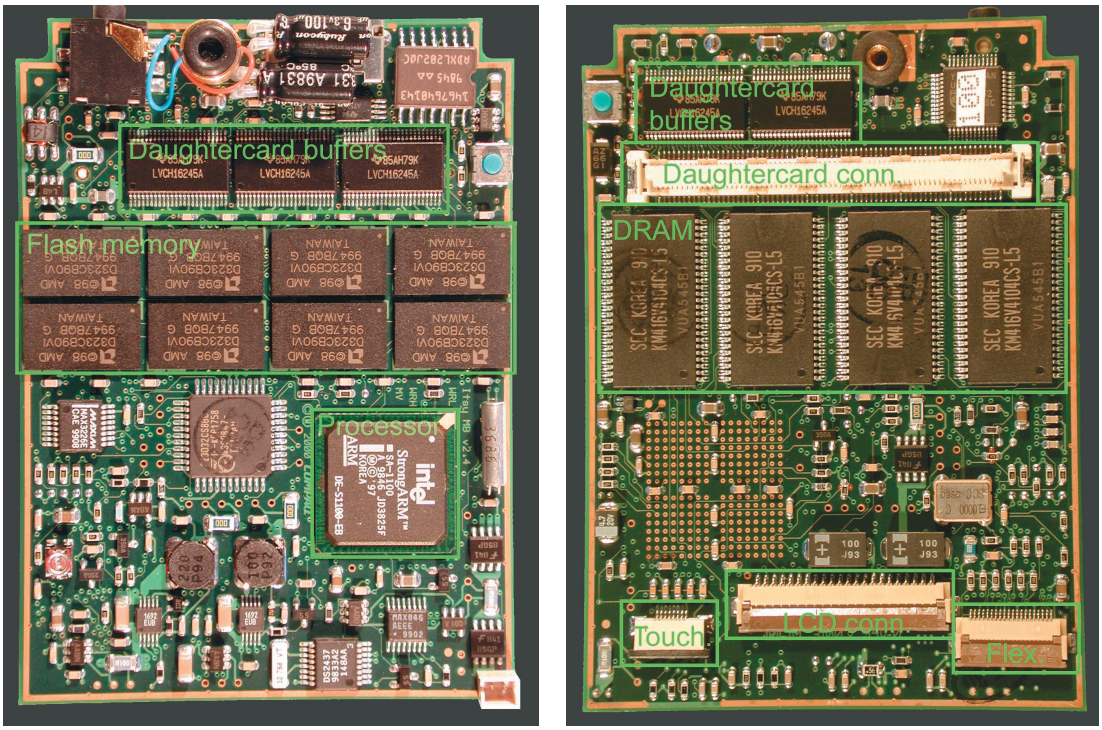
Our DRAM selection demonstrates how we used architectural decisions and component selection to stretch battery life. However, to make the best use of limited battery energy, we had to consider both power supply and consumption.

Because battery voltage varies widely during use, the design needs voltage regulators to provide the system with constant supply voltages. Linear regulators are small, cheap, and widely used in small devices, but they have poor efficiency, particularly when the input voltage is much larger than the output. Therefore, we used switching regulators for our two main supplies. However, because clean audio was a key requirement for speech recognition, we chose a small, low-noise, linear regulator for the analog circuitry.

Itsy uses logic specified to operate at 3.3 ± 0.3 V, but the power supply is set to 3.05 V, only slightly above the minimum. Because power consumption increases with the square of the voltage, this reduction from the usual 3.3 V, combined with the use of a switching regulator, results in an almost 15 percent power savings. Although a lower system voltage reduces noise margins, test systems have operated reliably at as low as 2.7 V, indicating that this design point was a reasonable trade-off.

Power-saving mechanisms

These power strategies were necessary but not sufficient, so we developed additional hardware features that allow the software to make the best use of available energy. The hardware does not automatically disable external peripherals when the processor enters



(a) (b)

Figure 2. The (a) top and (b) bottom sides of the Itsy v2 motherboard. Using micro-ball-grid array packages for the processor and flash memory offers a very high mounting density.

sleep mode. Instead, the software can turn each unit on and off individually. This strategy lets the operating system disable any of these units while the processor is running, or conversely, any of the units can remain active while the processor is in sleep mode. For example, the operating system usually puts the DRAM in self-refresh mode during sleep, but it can also choose to completely unpower the DRAM. We can use these mechanisms to implement a wide variety of sleep modes, ranging from deep sleep, which maintains only the real-time clock, to light sleep, which keeps the LCD enabled, the DRAM contents preserved, the clock on, and most interrupts configured to wake up the processor.

Itsy's design lets the software choose to power the processor core at either 1.5 V or 1.23 V. Although 1.23 V is below the manufacturer's specification, it is safe at moderate clock rates and yields about a 30 percent power savings for the core, translating to 10 to 20 percent overall.

Itsy can 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: from the USB or other external source, to and from the battery, into the processor core supply, and combined into the main and analog supplies. External laboratory instruments can monitor four additional current-sense resistors (for the DRAM and the three supply outputs). A battery monitor circuit integrates power into and out of the battery to continuously monitor the state of charge.

Accurately measuring the state of charge has proven challenging because the battery monitor chip has two limitations: It cannot handle Itsy's large dynamic-current range, and it provides no method of correcting for current-measurement input offset errors, which are significant in a system that spends much of its time in sleep mode.

Daughtercard interface

To facilitate development of high-performance hardware extensions, the daughtercard interface exports the full memory bus—all 32 data bits and all 26 address signals—and most other useful signals. These other signals implement serial ports, 14 general-purpose I/O pins (GPIOs), a software modem, and a single general-purpose analog input. Because the serial ports and the GPIOs share many processor pins, these features are not all available simultaneously.

SOFTWARE

To exploit Itsy's flexible hardware, we needed flexible software. We also wanted a comprehensive, robust environment for large applications. Open-source software was attractive because it eliminated barriers to sharing code. To meet these goals, we selected Linux, initially porting Russell King's ARM distribution (version 2.0.30) to an SA-1100 evaluation system and then to the Itsy platform. We then made significant additional changes to better support handheld computing, including improvements for power management and memory-based file systems. In addition, we designed and implemented *sessions*, a new device-sharing model that lets Itsy run different application environments concurrently.

Sessions, a new device-sharing model, lets Itsy run different application environments concurrently.

File systems

Itsy uses the Linux Ramdisk driver to provide dynamic memory partitioning between process address spaces and memory-resident file systems. This implementation is ideal for a handheld device because it does not waste space on redundant copies of data in the buffer cache and virtual memory system.

For additional savings, we modified the Ramdisk driver to discard blocks containing only zeros. This simple but surprisingly effective form of compression allowed us to create a large file system that does not consume physical memory until it stores actual data.

Changes to a Ramdisk are lost when power fails, so Itsy also has a flash-based file system for stable, writeable storage. This consists of an ordinary Linux ext2 file system that expects to run on top of a disk, plus a block device driver that emulates a disk in a portion of the flash. Because flash has very different properties from a disk, this emulation is not trivial: The system must erase flash before writing it, the minimum erase unit is large—128 or 256 Kbytes on Itsy—and erasing is slow—typically 700 ms per erase unit.

We used the industry-standard flash translation layer (FTL) data structure for disk emulation. We based our driver on code from the Linux PCMCIA subsystem, modified to work with Itsy's onboard flash. An FTL driver keeps a map from virtual disk block addresses to physical flash addresses. It handles reads simply by looking in the map to find the correct data. Writes are more complicated.

When the driver must write to a virtual block, it finds a free—previously erased—physical block, writes the data into it, and updates the map. The physical block previously mapped to this virtual address, if any, is now unused, but the driver cannot erase it if other blocks in the same erase unit are still in use. When no free blocks remain, the driver reclaims space by choosing an erase unit that contains some unused blocks, moving any in-use blocks still in the unit to a spare unit, and erasing the old unit, which becomes the new spare.

We observed a serious performance problem in this scheme. When the FTL driver reclaims space, it can free up only blocks that it knows are unused. FTL is

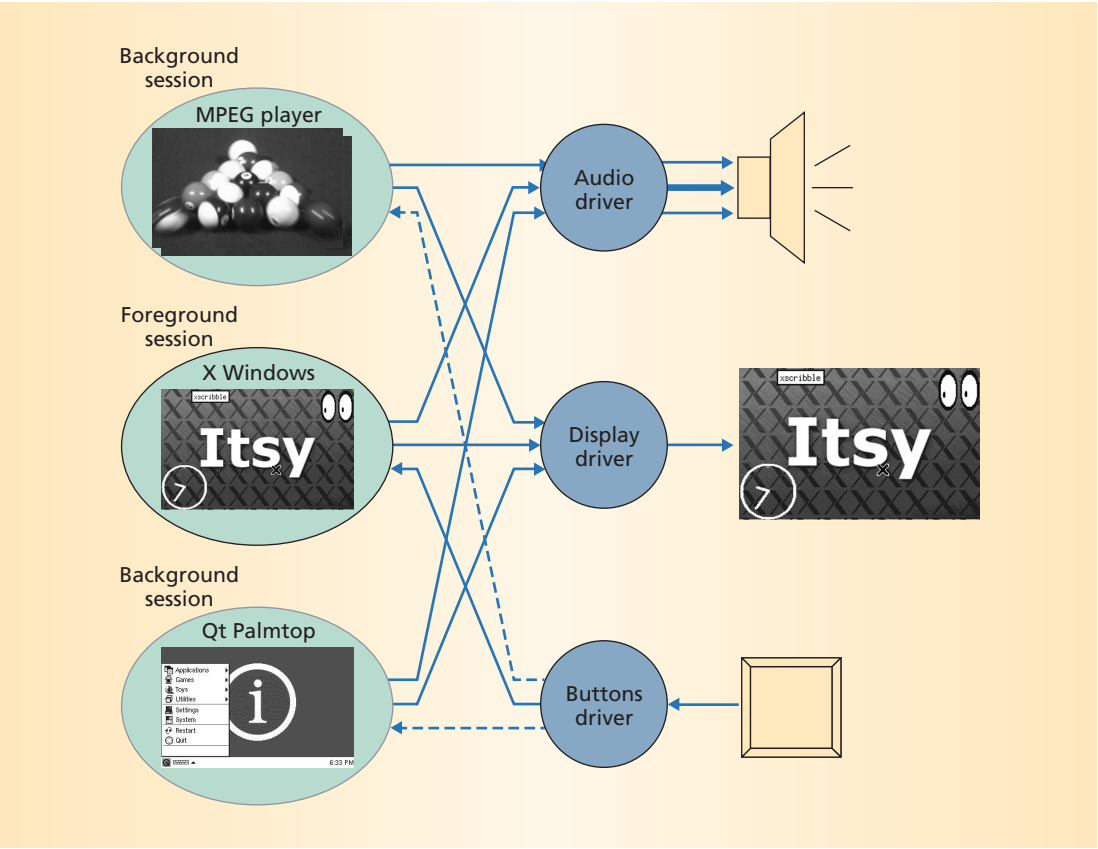


Figure 3. An example of sessions. The session labeled X Windows is in the foreground, and the others are in the background. The audio driver mixes its inputs, attenuating the inputs from the background sessions. The display driver shows one of the foreground session's frame buffers. When the user presses a button, only the foreground session is notified.

a disk emulator, not a file system, so it receives only virtual block read and write requests. Therefore, it does not find out that the driver has freed a physical block until the file system overwrites the corresponding virtual block with new data. In the worst case, when the FTL driver must write to a virtual block, it 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 the worst case by defining the flash's virtual size as only 95 percent of its physical size, so that 5 percent of the physical blocks are always known to be unused. As a result, a single erase can sometimes reclaim more than one block. However, FTL still performs as poorly as if the flash were 95 percent full, no matter how empty it really is.

To correct this problem, we made a small modification to the Linux ext2 file system. The file system now informs the block driver when 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.

Power management

To let Itsy operate at reduced power, we modified Linux to take advantage of the StrongARM processor's low-powered operating modes—idle and sleep—and variable clock speeds.

Exploiting idle mode is straightforward. The processor can enter and exit the mode in just a few clock cycles, so we modified the kernel idle loop to enter idle mode. When an interrupt occurs, the processor exits idle mode and returns to normal operation. The user cannot detect idle mode.

Sleep mode saves more power than idle mode. At the same time, it makes a bigger impact on the system, both in terms of software and user detectability, because sleep mode unpowers most of the processor. In particular, the on-chip peripheral controllers, including the LCD controller and the UARTs, don't work. Sleep mode can be initiated by user request (such as a button press), by the kernel, or by a system power fault. Entering sleep mode takes about 150 µs; exiting takes about 10 µs if the clock was left enabled, 157 µs if it was disabled.

We developed a power management module that coordinates the suspension and resumption of devices when sleep mode starts and ends. Each device registers callbacks that the power manager executes to determine whether or when the devices are ready to suspend. Another callback demands that the devices save their state immediately, pending suspension. When the processor exits sleep mode, the operating system reestablishes its previous state (stack and registers), and the power manager calls the devices for reinitialization.

The power manager module also suspends and resumes devices when changing the processor clock speed. Suspending and resuming devices, except the LCD, is necessary only to prevent temporary glitches, and it adds about 400 µs to the base clock switching time of 125 µs. Resetting the LCD takes much longer, but the processor can perform useful computation during this time.

Virtualizing Itsy devices

Unlike a unified windowing system that demands compliance from all applications, the sessions device-sharing model supports the peaceful coexistence of distinct yet concurrently executing "worlds," each of which sees its own virtual Itsy. Thus, Itsy can concurrently run systems as disparate as X Windows, Java, and Squeak, as well as full-screen stand-alone applications and virtual consoles.

A session consists of one or more processes. At any given time, one session is distinguished as the foreground session; the others are background sessions. As Figure 3 shows, foreground-session processes receive physical input events, such as button presses or touch-screen samples. They also have exclusive—or preferred—access to physical output devices. For example, a session's active frame buffer, if any, is visibly mapped onto the LCD. Processes in background sessions, on the other hand, do not receive physical input events, and their output is typically hidden.

When a process opens a physical device, it accesses only a session-filtered instance of that device. The input that each physical device generates is replicated to all instances in the foreground session. Output is merged from all instances in the foreground session. A device can drop, attenuate (for example, for audio), store, or specially handle background-session output. A user-level process serves as a session manager to determine which virtual Itsy instance is currently mapped to the physical hardware. Figure 4 shows a session manager running on Itsy.

Each physical device also exports a raw interface that applications requiring continuous access can use. For example, a speech recognition application running in the background can use the raw interface to snoop on all audio input, translate it, and offer the transcribed text to any application in the foreground session.

EXPERIENCES WITH USER INTERFACES

Itsy's system software makes porting existing desktop applications easy, but unfortunately, some assumptions of the desktop interaction style are inappropriate for small portable devices. Our initial attempts at using Squeak's graphical user interface on Itsy demonstrated the difficulty of mapping a device with a large display and three-button mouse to a device with a small display and a stylus. We developed conventions that made the Squeak GUI usable, but it was by no means easy to use.

Users cannot move a stylus on a handheld device as easily and accurately as they move a mouse on a desktop. As a result, mastering handheld interfaces that require precision pointing and tapping is difficult. Additionally, a desktop GUI has the user's full attention for extended periods, but the only portable application domain that can demand such attention is gaming. Clearly, as the Palm operating system demonstrates, GUIs must be specially designed for small systems.

As portable devices shrink further, stylus-based input becomes even more difficult. This problem led us to experiment with two nontraditional user interfaces: speech-based input and output and gesture-based input.

Figure 4. A session manager running on Itsy v2. Several applications are simultaneously running in the background, including an MPEG player, a battery monitor, and a voice organizer.

Speech

One promising interface for a tiny device is speech. For output, Itsy uses DECtalk, a commercial text-to-speech engine. For speech input, a more challenging problem, we had two speech recognition systems ported to Itsy. The first, 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 e-mail program. Although not a complete implementation, this program has successfully demonstrated the feasibility of speech-based interaction in realistic environments, even using Itsy's built-in microphone in a crowded room.

Dragon Systems ported its speech recognition system to Itsy. This system includes both a speaker-independent, grammar-based, continuous-speech, command-and-control engine and Dragon's NaturallySpeaking, a speaker-dependent, continuous-speech dictation engine with a 30,000-word vocabulary. Together, the two engines offer the potential for a rich speech-only user environment.

The promise of speech recognition has been partially realized in the sense that adequate performance is no longer an obstacle to using speech as an interface for pocket-sized devices. However, we must still meet the challenge of building effective speech-centered or multimodal user interfaces.

Gesture

Most desktop-computer input methods rely on physical manipulation of an object such as a keyboard or mouse. As systems shrink, we can use the motion of the system itself for user input. Tilting a handheld

Figure 5. A photo album application on Itsy. When a user makes a gentle fanning gesture with the Itsy, a two-axis micromachined accelerometer senses the fore and aft and left and right tilting, and the photo album application displays the next picture.

computer to navigate through a document has long been anticipated, but sensors have only recently become small enough and cheap enough for developers to embed them in handheld devices that implement the tilt-to-scroll method.

We extended the tilt-to-scroll method to include the use of gestures to issue commands. Our 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.[4]

A photo album application demonstrates this interface. The user tilts the album on either axis to scroll through miniature photographs until finding a picture of interest. When the user makes a gentle fanning gesture, the album zooms in on that picture. The user can make additional fanning gestures to step through the rest of the album, return to the miniatures, or disable and enable scrolling. Pictures are available in landscape and portrait mode; holding the unit in the new orientation for a few moments reorients the picture.

Early experiments with a handheld mockup demonstrated that users quickly learn to operate Rock 'n' Scroll and gave us some insight into user expectations. These positive results, as well as improvements in sensor technology, encouraged us to incorporate Rock 'n' Scroll as a standard input method on Itsy. Figure 5 shows an implemention of the photo album application on Itsy. A two-axis micromachined accelerometer senses fore and aft and left and right tilting, and software converts the accelerometer's outputs to gesture commands. In our user study, we observed that tilting the mockup to play a simple game fascinated nearly all participants. This observation was confirmed by the enthusiastic reception of our port of id Software's Doom game in which users tilt the Itsy to navigate through a three-dimensional environment.

HEAVY LIFTING WITH A TINY BATTERY

To meet 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 run realistic user interface experiments. We ran performance and energy consumption tests to assess how well we fulfilled these needs.

Performance

By running the Dhrystone benchmark on Itsy and interpolating published results from other systems,[5] we found that Itsy's integer performance is similar to that of a Pentium P5 system running at 110 MHz. On the larger, more complex SPECint92 benchmarks, Itsy performs more poorly because it has smaller caches. For the SPECint92 subset that we compiled and ran, Itsy's performance was comparable with a Pentium running at 90 MHz.[6]

These results show that Itsy has the performance capability to run programs normally associated with desktop systems. On a 206-MHz Itsy, the DECtalk text-to-speech engine runs at only a 47 percent system load, the command-and-control engines process speech considerably faster than real time, and Dragon's NaturallySpeaking dictation engine runs about 2.4 times slower than real time.

Energy consumption

We used an automated test rig[7] to thoroughly evaluate Itsy's energy use. Table 2 shows the results. Because most of our applications, such as playing an audio or video stream, have a fixed duration by definition, we can characterize them by either average power consumption—the energy per unit of time—or average energy consumption. Sleep mode and idle mode also fall into this category. In other cases, such as our batch-mode voice recognition experiment, the energy required for a quantum of work is a more relevant metric than power.

Table 2 shows that in sleep mode Itsy can preserve the contents of its 32 Mbytes of DRAM for almost 13 days on a single battery charge. If we use a daughtercard to add an additional 32 Mbytes of memory, Itsy still retains data for almost nine days. When Itsy is on but not doing any work (idle mode), it can stay alive for 22 to 32 hours. We also can put Itsy into sleep mode while keeping the LCD and interrupts on, thus faking an idle system. In that case, a battery charge lasts more than three days.

However, a handheld computer's main purpose is to perform useful work. Itsy can play an audio file for 6.9 to 7.7 hours and generate speech from a text file for 5.3 hours. It can perform continuous speech recognition for 2.7 hours. The recognizer runs about 2.4 times slower than real time, so this corresponds to slightly more than one hour of speech dictation.

Table 2. Itsy v2's power consumption, battery lifetime, and effective battery capacity.

Experiment	Clock speed (MHz)	Processor idle (%)	System power (mW)	Battery lifetime (h)	Battery capacity (W × h)
Deep sleep mode			4.58	500.0	2.29
Sleep mode			7.40	308.5	2.28
Sleep mode, mem. dc*			10.6	215.0	2.27
Sleep mode, static LCD image			26.2	87.0	2.27
System idle, idle mode	59	95	69.5	32.3	2.25
System idle, idle mode	206	95	101	22.0	2.23
Playing audio file (WAV)	59	83	278	7.75	2.16
Playing audio file (WAV)	206	93	310	6.88	2.13
Text to speech**	74	<1	397	5.35	2.12
Text to speech**	206	53	401	5.29	2.12
Dictation, *** mem. dc*	206	<1	757	2.67	2.02
Playing MPEG-1 video file with audio	206	16	821	2.42	1.99
System idle, idle mode, low core voltage	59	95	55.4	40.6	2.25
Text to speech, ** low core voltage	74	<1	352	6.11	2.15

*Memory daughtercard (32-Mbyte DRAM), **DECtalk, ***Dragon's NaturallySpeaking

Finally, Itsy can decode and play an MPEG-1 video file for 2.4 hours.

Users typically need bursts of computation, interspersed with periods of sleep or idle mode, so a realistic power use scenario is a mixture of the numbers shown in Table 2. This data indicates that most users are unlikely to run out of power if they recharge the battery every night.

Table 2 also shows that the relationship between processor clock frequency and overall power is not intuitive. In a system with many components, the clock frequency directly affects some components but affects others only indirectly or not at all. For example, in idle mode, decreasing the frequency from 206 MHz to 59 MHz can save about 30 percent of the power. When Itsy plays an audio file, however, the power savings drops to 10 percent because the power the speaker dissipates is independent of the clock frequency. Finally, the power the system requires to generate speech varies little with frequency.

Our studies also show that changing the clock frequency alone produces limited savings at best. Instead, the voltage should change at the same time as the frequency. Although many next-generation processors will provide this functionality, the current StrongARM does not. However, we implemented a similar mechanism by allowing the core voltage to take two possible values. The last two lines of Table 2 show that this scheme saves an additional 20 percent of the power in idle mode and 11 percent in speech generation.

The Itsy pocket computer has been a useful tool for exploring the bounds of mobile computing. It has proved powerful and flexible enough for interesting applications, systems work, and power studies. Designers both inside and outside our organization have built numerous daughtercards, including several CMOS cameras,[8] a PCMCIA adapter with a large battery, a low-powered radio, and many memory expansion cards. The Linux operating system, once thought too unwieldy for handhelds, worked well for Itsy. Linux is now being tried on other small devices, such as the commercially available Compaq iPAQ H3600 series of handhelds, and IBM Research is even using Linux on a wristwatch prototype.[9] Although we demonstrated the possibilities of industrial-strength voice recognition, the Rock 'n' Scroll gesture-based interface generated the most interest, and we expect small accelerometers to be widely used in future systems.

Our measurements suggest that to manage power effectively, a handheld system must have a way to assess its own power consumption. Systems that accurately measure their own power consumption, as Itsy does, can more easily exploit processors that support frequency and voltage scaling. Relying on predefined policies without such feedback is unlikely to be as successful. All these lessons will be important in the design of future systems, as users' needs drive the incorporation of more capabilities into smaller packages. ✷

References

1. M.A. Viredaz, *The Itsy Pocket Computer Version 1.5: User's Manual*, tech. note TN-54, Western Research Laboratory, Compaq Computer Corp., Palo Alto, Calif., 1998.
2. *Intel StrongARM SA-1100 Microprocessor: Developer's Manual*, Intel Corp., Santa Clara, Calif., 1999.
3. R. Stephany et al., "A 200-MHz 32b 0.5W CMOS RISC Microprocessor," *Proc. IEEE Int'l Solid-State Circuits Conf.*, IEEE Press, Piscataway, N.J., 1998, pp. 238-239, 443.
4. J.F. Bartlett, "Rock 'n' Scroll Is Here to Stay," *IEEE Computer Graphics and Applications*, May/June 2000, pp. 40-45.
5. "Performance Database at the Netlib Depository," http://www.netlib.org/performance.
6. J.F. Bartlett et al., *The Itsy Pocket Computer*, research report 2000/6, Western Research Laboratory, Compaq Computer Corp., Palo Alto, Calif., 2000.
7. M.A. Viredaz and D.A. Wallach, *Power Evaluation of Itsy Version 2.4*, tech. note TN-59, WRL, Compaq Computer Corp., Palo Alto, Calif., 2001.
8. J.F. Bartlett, *A Simple CMOS Camera for Itsy*, tech. note TN-58, WRL, Compaq Computer Corp., Palo Alto, Calif, 2001.
9. C. Narayanaswami and M.T. Raghunath, "Application Design for a Smart Watch with a High-Resolution Display," *Proc. 4th Int'l Symp. Wearable Computers*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 7-14.

William R. Hamburgen *is a member of the research staff at Compaq Computer Corp.'s Western Research Laboratory. His research interests include packaging and power systems for the smallest mobile computers. He received an MS in mechanical engineering from Stanford University. He is a member of the IEEE and the ASME. Contact him at bill.hamburgen@alum.mit.edu.*

Deborah A. Wallach *is a member of the research staff at Compaq Computer Corp.'s Western Research Laboratory. Her research interests include mobile computing, operating systems, and power management. She received a PhD in computer science from the Massachusetts Institute of Technology. Contact her at deborah.wallach@compaq.com.*

Marc A. Viredaz *is a member of the research staff at Compaq Computer Corp.'s Western Research Laboratory. His research interests are computer architecture, parallel systems, and low-power computing. He received a PhD in computer engineering from the Swiss Federal Institute of Technology at Lausanne. He is a member of the IEEE and the IEEE Computer Society. Contact him at viredaz@computer.org.*

Lawrence S. Brakmo *is a member of the research staff at Compaq Computer Corp.'s Western Research Laboratory. His research interests include operating systems, power management, and mobile computing. He received a PhD in computer science from the University of Arizona. Contact him at lawrence.brakmo@compaq.com.*

Carl A. Waldspurger *is a senior member of the technical staff at VMware Inc. His research interests include operating systems, virtual machines, resource management, and mobile computing. He received a PhD in computer science from the Massachusetts Institute of Technology. He is a member of the IEEE and the ACM. Contact him at carl@waldspurger.org.*

Joel F. Bartlett *is a member of the research staff at Compaq Computer Corp.'s Western Research Laboratory. His primary research interest is "off-the-desktop computing." He received an MS in computer science and engineering from Stanford University. He is a member of the ACM. Contact him at joel.bartlett@compaq.com.*

Timothy Mann *is a member of the research staff at Compaq Computer Corp.'s Systems Research Center. His research interests include operating systems, file systems, distributed computing, and software configuration management. He received a PhD in computer science from Stanford University. He is a member of the IEEE and the ACM. Contact him at tim.mann@compaq.com.*

Keith I. Farkas *is a member of the research staff at Compaq Computer Corp.'s Western Research Laboratory. His research interests include microprocessor design and software/hardware techniques for extending the battery lifetime of mobile computers. He received a PhD in electrical and computer engineering from the University of Toronto. He is a member of the IEEE and the ACM. Contact him at keith.farkas@compaq.com.*