

Energy trade-offs in the IBM Wristwatch computer

Noboru Kamijoh, Tadanobu Inoue, C. Michael Olsen⁺, M. T. Raghunath⁺, Chandra Narayanaswami⁺

{kamijo,inouet}@jp.ibm.com, {cmolsen,mtr,chandras}@us.ibm.com

IBM Research Division, Tokyo, Japan

⁺ *IBM Research Division, Yorktown Heights, NY, USA*

Abstract

We recently demonstrated a high function wrist watch computer prototype that runs the Linux operating system and also X11 graphics libraries. In this paper we describe the unique energy related challenges and tradeoffs we encountered while building this watch. We show that the usage duty factor for the device heavily dictates which of the powers, active power or sleep power, needs to be minimized more aggressively in order to achieve the longest perceived battery life. We also describe the energy issues that percolate through several layers of software all the way from device usage scenarios, applications, user interfaces, system level software to device drivers and the need to systematically address all of them to achieve the battery life dictated by the hardware components and the capacity of the battery in the device.

1 Introduction

We built the high function IBM wrist watch computer prototype to study several areas of mobile computing such as user interfaces [1], high resolution displays [2], system software, wireless communication, security, and interaction patterns between various pervasive devices. We view this watch as a wearable computing platform rather than a special purpose device. Therefore our goals differ quite significantly from those posed to the designer of a traditional wrist watch with just time keeping functions.

We chose a watch form factor (see Figure 1) because watches are easily accessible and get misplaced less often than PDAs and cell phones since they are worn, not carried. It is also commonly believed that many people glance at their watches up to forty times a day and this we think is a good reason to put some additional useful information, such as upcoming appointments on the watch face. The watch is also an ideal device for conveying information alerts to the user, since it is instantly viewable.

The watch form factor also takes many of the packaging, user interface, and power problems to the extreme which appealed to several of us.



Figure 1. Wrist watch prototypes.

Our watch has a touch sensitive screen and a roller wheel for user interaction. The main card has an ARM 7 based Cirrus EP 7211 processor, 8MB of Flash memory and 8MB of DRAM, and serial, IrDA, and expansion interfaces. We have two monochrome displays, a 96x120 reflective LCD and a 640x480 Organic LED (OLED) display [2]. Bluetooth™ functionality is provided by an auxiliary card that connects to the main card in some watches. The watch is powered by a rechargeable Lithium polymer batter with 60mAh capacity at a nominal voltage of 3.7V. Figure 2 shows the hardware block diagram.

We chose the Linux operating system for our watch because of the availability of source code and a wide variety of software tools, and programmer familiarity. Third party software developers are less likely to be interested in learning new programming interfaces unless the platform is already widely deployed. For the same reason we chose the X11 graphics library instead of defining a new graphics library.

People are often impressed by the amount of function we have managed to fit into such a small package. However, a question that often gets asked is how long the batteries last. Obviously the battery life on a watch like ours will not compare favorably with conventional watches that have limited functionality. Notwithstanding,

battery life is an important aspect that we paid attention to, and is at the heart of many trade-offs in the design of the entire system [3]. In the rest of this paper, we discuss the trade-offs we made and the motivations behind these trade-offs.

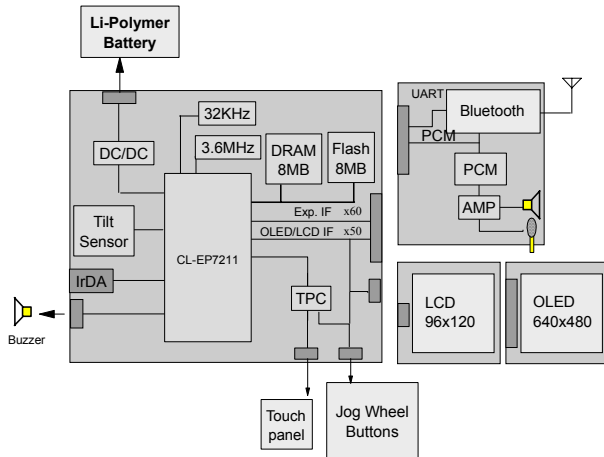


Figure 2. Hardware block diagram.

1.1 The Energy Challenge

The holy grail, energy wise, in the design of mobile devices is to enable them to be self sustaining [4]. Calculators powered by solar cells and self-winding mechanical watches are examples of devices that have attained this goal.

Though the energy challenge is often interpreted as making the device run as long as possible and be as energy efficient as possible, a more pragmatic viewpoint is whether battery life can exceed some acceptance thresholds. Under normal usage patterns, cell phone batteries last about a week which appears to be an acceptable threshold. An analogy can be drawn from automobiles. Both car buyers and manufacturers do not agonize excessively and solely over gas mileage and pay attention to form and function because most people don't need to refuel on each trip, and when they do, it is fairly easy to refuel. The device is likely to get used if it provides services to the user that outweigh the difficulty for caring for it. So we have the dual challenge of providing useful function while minimizing the effort of caring for it.

Devices that are unable to attain the goal of self sustenance often use primary cells and attempt to make the replacement of the batteries easy and less frequent. If the energy requirement is such that the user will need to replace the battery too often, rechargeable batteries have to be employed to minimize user aggravation and to protect the environment. However, while rechargeable batteries can be charged a several hundred times, they generally hold four to five times less energy than

non-rechargeable ones for comparable size and weight. Devices that operate on rechargeable batteries must also attempt to reduce the time taken to charge up the battery.

If the recharging can be serendipitous, i.e., combined with some other benefit that is delivered to the user, such as receiving information from the Internet, the user may not perceive recharging as an inconvenience.

1.2 Challenges in a wrist watch form factor

In addition to the general energy challenge faced by other wearable computing devices, there are several additional challenges imposed by the choice of a wrist watch form factor.

The simplest way to increase the battery life is to use a larger capacity battery which will be larger and heavier, but the size and weight requirements on a wrist watch place an upper bound on the capacity of the battery that can be used to power it. With the best available rechargeable Lithium polymer batteries today, the maximum capacity of a battery measuring 3cm x 2cm x 0.5cm, that can be fit into a regular size watch, is about 200 mAh.

Second, replacing batteries on a watch is generally difficult due to the small size of the watch and the need to make the watch water resistant. A significant number of customer complaints with watches directly result from users trying to replace the batteries. Traditional watch manufacturers attempt to make the battery last so long that the user is more inclined to buy a new watch when it is time to replace the battery.

The third problem relates to user perception. Users are not accustomed to having to recharge wrist watches, but may be willing to recharge other devices such as cell phones and PDAs. It is important to make the user perceive a high function wrist watch as being similar to these other devices rather than traditional wrist watches.

As mentioned earlier, an advantage of the wrist watch is that it is instantly viewable. A constraint that arises from this aspect is that the watch should preferably have some useful information on its display at all times. While saving energy by turning off the display is an option on many devices, doing so on a watch may take a significant advantage away unless it can be done so cleverly that the display is always on when the user is looking at the watch.

In the following sections we describe the energy related tradeoffs associated to the device usage model, the hardware, system level software and application level software. We end with some suggestions for further improvement of battery life.

2 Device usage model

Wearable computing devices are generally in one of two modes, sleeping or active. The device is in the active state typically when the device is doing something for the user; e.g., performing some computation, obtaining and displaying data etc. The rest of the time, the device sleeps. It transitions to the active mode in response to some action by the user or the environment. Depending on how long the device takes to come out of the sleep mode, there may also be states in between these two extremes, such as an idle mode where the device responds more quickly to the user than if it were in the sleep mode.

As a gross approximation one may characterize the device using two power metrics: The power consumed in the active mode (P_{active}) and the power consumed in the sleep mode (P_{sleep}). One can also approximate the actual usage of the device using a single metric: the usage duty factor (D) which is the fraction of the time the device spends in the active mode.

Informal surveys reveal that owners of Palm Pilots™ use them about ten times a day for about 30 seconds at a time. This adds up to about 5 minutes per day or a duty factor of about 0.0035. Users of the ParcTAB [5] reported that their device was on for less than 100 seconds at a time. With today's hardware, the device can periodically go into an idle mode, unbeknownst to the user, even when the user is actively interacting because of the reaction time of the human user and further reduce the actual duty factor. The exceptions to low duty factors tend to be devices that perform active functions even when the user is not consciously and actively using the device. An example of such a device is an MP3 player watch where the user only needs to initiate the play function and cause the watch to actively run and play the music requested.

Observations of the usage patterns of wristwatches suggest that the amount of time the user actually spends interacting with the advanced features on a digital watch is generally an insignificant fraction of the total time. This is true for many high function wrist watches as well. Calculators probably have an even smaller duty factor.

Based on these metrics, the average power consumed by the device is given by $P_{\text{sleep}}(1-D) + P_{\text{active}}D$. If we further define the ratio of active power to sleep power as the power factor ratio: $\text{PFR} = P_{\text{active}}/P_{\text{sleep}}$, then the total power consumed by the device is $P_{\text{sleep}}(1-D) + \text{PFR} * P_{\text{sleep}}D = P_{\text{sleep}}(1-D + \text{PFR}*D)$.

The PFR for our watch ranges from around 30 to 100 as seen from table in section 3. In comparison, PFR for a PalmPilot™ ranges from 60 to 280 [6,7], the Psion Series 5™ PDA ranges from 70 to 240 [8], and the Compaq Itsy ranges from 30 to 90 [9,10].

The battery life can be approximated as Battery Capacity (mWh) / Average power consumed. This is an approximation since battery capacity is not really a constant but is a function of the precise wave form of the load as opposed to just the average [11,12]. Nevertheless, it is useful to examine the effects of the different parameters on the battery life.

Figure 3 below shows the interplay between the usage duty factor and the predicted battery life for the watch for different power factor ratios. The duty factor is shown on a logarithmic scale. The battery life is normalized to one when the device is in the sleep mode all of the time. Once the hardware is built, the PFR is fixed, and the predominant way of extending the battery life is to minimize the actual duty factor.

Ideally, both the minimum sleep current and the maximum current consumption must both be minimized. But which is more important depends on the duty factor.

At the frequently encountered low duty factors, it is very important to focus on minimizing the sleep power because reducing the PFR has less perceivable impact on relative battery life because we are at the left end of the curves in Figure 3. On the other hand, if the duty factor is much higher, say two hours a day, then we operate in the middle of the graph and it is important make the PFR smaller by reducing the active current consumption very aggressively.

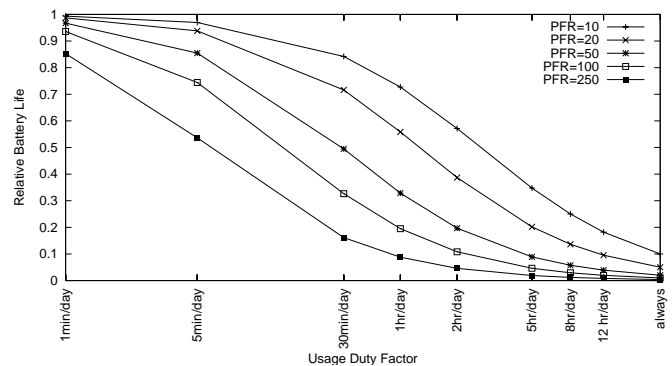


Figure 3. Relative battery life versus duty factor.

3 Hardware level energy trade-offs

The upper and lower bounds on the battery life for a device are determined by hardware. The maximum battery life depends on the minimum sleep current that can be achieved. Similarly, the minimum battery life is dictated by the maximum current consumption when all hardware components are turned on.

We attempted to keep the energy requirement as low as possible by choosing energy efficient components whenever possible. However, energy efficiency often comes with less function, e.g., task specific hardware can generally operate at a lower energy cost compared to a programmable processor, and smaller memories consume less power to refresh than larger ones. Compared to other wrist watches our design trades off energy efficiency for greater function, as is evident from the 32-bit processor and large amount of memory we incorporated.

Fitting all of the components onto a small board, measuring 27.5 mm x 35.3 mm, was very challenging. Even when more energy efficient components were available, we were unable to use them because they were larger and would have increased the size of the circuit board. For instance we used a voltage regulator with a leakage current of 100 μ A since it was a factor of 8 smaller than a voltage regulator with a leakage current of 10 μ A. Often controls are provided to turn off portions of the hardware to save energy. For example, in some systems multiple DRAM banks are used and some banks are powered off when the software knows that a particular DRAM bank is not being used. However, we do not have the room on the main board to be able to fit several memory modules and hence cannot apply multiple DRAM banks to mitigate the power problem in the sleep mode. Likewise we did not have any room to have a battery backup or a capacitor to preserve DRAM contents if the main battery was discharged.

In order to drive all of the components in our watch, except for BluetoothTM, we needed a battery that could supply currents as high as 50 mA. This constraint ruled out coin cells and led us to choose a rechargeable Lithium Polymer battery. A rechargeable battery was also appropriate since we did not want to open the watch to replace the battery often.

Within the constraints of function and size, we designed the hardware to be as energy efficient as possible. The processor we chose, an EP7211 chip [16] from Cirrus Logic, supported most of our required peripherals such on chip ROM, integrated LCD controller, IrDA controller, UART, serial, PCM and serial bus interfaces, etc., and we did not require the space or power for additional glue logic. The processor itself was used in bare die package instead of the regular package to save both space and power. We run the processor at 18Mhz though it can run up to 74 Mhz. The processor supports two power saving modes called IDLE and STANDBY which are described below. In addition, we ensured the input devices on the watch, namely a touch panel, a roller wheel and a tilt sensor, consumed no power in the standby mode.

Another significant power saving feature we incorporated into the OLED display was the ability rapidly

turn the pixels on and off and to control its brightness by adjusting the ratio of the time that the pixels are on to the time they are off. We also have a clear instruction in the OLED that clears the display without requiring the CPU to zero all the pixels individually.

Table 1 lists the manufacturer specified power consumption for various components in operational and standby state in our system. (Vdd=3.3V in most cases.)

	Operational	Sleep
Component	max/typ [mW]	max/typ [mW]
CPU (EP 7211)	50/-	0.010/-
DRAM (8MB)	363/-	0.660/-
Flash (8MB)	72/-	-/0.001
Touchpanel ctrller	1.8/-	0.008/-
IrDA	86/66	0.003/0.000
LCD	-/3	0.017/0.000
Codec	27/15	0.031
SPK Amp	7.2/4.8	0.002
DC-DC 3V	0.6/0.3	0.003/0.000
DC-DC 2.5V	0.6/0.3	0.003/0.000
Bluetooth (early hw)	198/132	1.82

Table 1. Manufacturer specified power consumption in operational mode and in sleep/disable mode.

The range for power quoted by the specifications sheets for some components is rather large and the actual consumption is determined by the usage scenario and how the components are configured and interoperate, e.g., can the Flash and the DRAM be consuming their peak powers at the same time due to the bus architecture. Tools are needed to measure power accurately [13]. Also, measuring the power consumed by the various components is not straightforward in many situations because there may be no way to turn off a particular component explicitly, e.g., the DRAM, once Linux is running. In these situations we devise a set of tests that are similar to power on self test, and measure the power consumed by various components before the operating system is loaded. One more issue we had was that the battery capacity also reduces over time and some of our current batteries appear to have only 35mAh as opposed to 60mAh when they were fresh.

Furthermore we managed to reduce the operational voltage to be a little lower, i.e. from 3.3V to 3.0V, by selecting all other components such as Flash memory, DRAM, touch panel controller and the IrDA module to be the same and this reduced the power consumption by an estimated 17.3%, assuming static CMOS. The voltage regulators consume 200 μ A and 45 μ A is consumed in bias circuits. Table 2 shows the current consumption in the watch in the different states of its usage. (Note, multiply by 3.7V nominal battery voltage to get corresponding power.)

As a comparison, typical standby currents of the Palm IIIxTM (4 MB of memory) is 300 μ A, that of the Psion

Series 5™ is 540µA (8 MB of RAM). In active modes (with the backlight off though), the current consumption in the Palm IIIx™ ranges from around 15 mA to 65 mA [6,7], and that for the Psion Series 5™ ranges from 35 mA to 130 mA [8]. The backlights on the Palm IIIx add about 40 mA and on the Psion Series 5 add about 70 mA to the above numbers.

How far can we go with hardware improvements? Even with the 200mAh in a state of the art Lithium polymer battery, if the system designer wants the battery to last a year, to a first order approximation, ignoring battery non-linearities and dependence on battery recovery processes on peak current drained and the self discharge of the battery, the average current consumption in the device should be $200/(365*24) = 23\mu A$ or less. This is an order of magnitude lower than the Palm IIIx™ consumes with 4MB RAM. We also have to fight about a 5% self discharge rate per month in Lithium polymer batteries [14].

Processor	LCD	DRAM	Total current
OPERATION 14mA	On 348 µA	Active access 0.3-40mA	15-55mA average of 27.5mA
IDLE <3744µA	On 348 µA	Processor Refresh > 298 µA	4.6 mA
STANDBY 22 µA	On 348 µA	Self Refresh 298 µA	913 µA
STANDBY 22 µA	Off 0µA	Self Refresh 298 µA	565 µA

Table 2. Current consumption of main components in various states (measured at battery terminal).

4 System software level energy trade-offs

As noted earlier, battery life depends to a large extent on the power consumed in the sleep state if the usage duty factor is low. So reducing the power consumed in the sleep state is the first issue we looked at.

4.1 Kernel Optimizations

When the system is in the sleep mode, the task scheduler in the Linux kernel sees no tasks that are ready to run. In this situation, the kernel switches the processor into the IDLE mode. Also the kernel relies on a timer to interrupt it every 10 ms in order to guarantee fair scheduling amongst the tasks that are ready to run. Every 10 ms when the timer interrupt occurs the processor switches to the active mode for a short duration when it updates the kernel time variables and checks the task

queue. If there are no tasks that are runnable, the kernel puts the processor back into the IDLE mode. The processor also comes out of the IDLE mode when there is an external interrupt as would occur if the user wanted to interact with the watch. Also tasks can request to be woken up at some point in the future. The kernel maintains a list of such tasks in a timer list. At each timer tick the kernel checks to see if sufficient time has elapsed to wake up one or more sleeping tasks.

As observed earlier, the energy required by the processor in the IDLE mode is much higher than the energy required in the STANDBY mode. So we first focused on trying to get the processor into the STANDBY mode instead of the IDLE mode when there was no work to do.

On the ARM processor we used in our watch the 10 ms timer is not available in the STANDBY mode. However I/O interrupts work and a real time clock (RTC) which operates on a one second granularity is available. Also the processor may take up to 250ms to get out of STANDBY mode but coming out of the IDLE mode typically takes only a few clock cycles. From a user perspective, the 250ms delay is perceptible, but not annoyingly so.

In order to take advantage of the more efficient STANDBY mode, we had to rewrite the Linux timing methodology for our system. The basic idea is as follows: Whenever the scheduler found that it could put the processor in the IDLE mode, we scan the timer list to see how much time needs to elapse before any of the tasks need to be woken up. If this time-out interval is long enough we put the processor in STANDBY mode after programming the real-time clock (RTC) to wake us up before this time-out. When we come out of the STANDBY mode either on a RTC interrupt or some other interrupt, we adjust the kernel time variables so that the kernel knows how much time has really elapsed. If the nearest task is too close for the 1 second granularity of the RTC, we go into IDLE mode instead and program the 10ms timer to wake us up at the time-out period.

Our changes to the Linux kernel seem to work fine with all of the applications we have tested so far, though we have not tested this code with a more demanding set of user workloads.

Figure 4 shows a pseudo code snippet of the changes we made. This code is from the "main idle loop" in Linux, and is in the file arch/arm/kernel/process.c. The lines in bold are the statements we have added. The operation is as follows. Whenever the current task that is executing in Linux has run to completion, control is returned to the main idle loop. At the very top of the loop, the current task is checked to see if it needs scheduling. If there is no current task that needs scheduling, *work_candidate()* is called to resolve if there are other tasks that are eligible for scheduling. If there are no tasks, then there's no work

to be done at the current time. So to determine when there is work to be done, we parse the Linux timer list in `get_nearest_timeout()` to find the nearest time-out. The returned time-out value is then used by `adjust_hw_timer()` to program either the RTC (if the time is long enough) or the 10ms timer to expire before or at that time-out value. `adjust_hw_timer()` also sets the flag `B_STANDBY_VALID` if the time is long enough to go into STANDBY mode.

The selection, programming and handling of the hardware timers and the related issue of keeping accurate time is complex and we expect to publish the details in the future.

```
while(1) {
    if ( !current->need_resched && !hlt_counter ) {
        cli();
        if ( !work_candidate() ) {
            get_nearest_timeout();
            adjust_hw_timer(); }
        sti();
        if ( B_STANDBY_VALID )
            proc_stdby();
        else
            proc_idle(); }
    current->policy = SCHED_YIELD;
    schedule();
}
```

Figure 4. Modified Linux 2.2.1 "main idle loop".

4.2 Other Optimizations

Once we have done all that we can to reduce the power consumption in the sleep state, the next focus from a system software perspective is to ensure that the duration spent in the active state in response to user action is minimized. In order to reduce the time spent in the active state, device drivers are optimized to reduce the number of instructions that are executed in response to interrupts. Device drivers also use interrupts and eliminate polling or kernel timer based scheduling whenever possible.




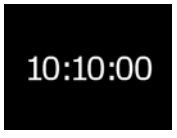
As noted above, we run the X11 graphics library on our watches. The X11 code consists of the Xserver and the Xlib library that applications can link to. The Xserver typically draws directly to a frame buffer and in our case would typically directly write to the LCD or the OLED. Individual byte or word writes to these devices are expensive in terms of power consumption and can also result in a visible flicker. We address both of these problems by letting the Xserver write to a shadow display buffer maintained in DRAM. When we know that the Xserver has completed all of its drawing calls and will go into a mode where it waits for the next event, we copy the contents of the shadow display buffer to the actual display.

5 Application level energy trade-offs

In the default mode the application displays the time and calendar information. By using the touch screen and the roller wheel the user can navigate to other screens and access information such as phone lists, things to do, etc. The application program is written in C and uses the X11, math and standard C libraries.

While it is clear that the application code may have control over the energy consumed in the active mode, it is interesting to note that in the case of our watch it has an impact on the energy consumed in the sleep mode as well. Our watch shows the current time on the display when the watch is in the sleep mode. Both the LCD and OLED displays consume less energy if fewer pixels are turned on. So the application code can help reduce the amount of energy required in the sleep mode by turning on fewer pixels. The impact of the number of pixels on the energy required is greater on the OLED display, but on the LCD the difference is not as pronounced.

On the 640x480 OLED display, we calculated the number of pixels that had to be turned on to display both an analog clock face (with hands) and a numeric time display. We found that an analog clock face could be displayed with fewer pixels. We can reduce the number of pixels for a digital display by reducing the font size, but this impacts readability. In addition, an analog clock face was generally perceived to be much easier to read (since it also indicated the amount of time remaining till some point in the future as well as the current time) and appeared more elegant. The different screens we compared are shown in Figure 5 below in actual size.

	Hollow minute hand Pixel count: 3849 (1.25%)
	Thinner hands Pixel count: 3280 (1.07%)
	Triangular hands Pixel count: 4088 (1.33%)
	120 pixel tall Verdana font Pixel count: 13916 (4.53%)


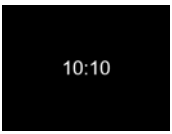
	120 pixel tall Arial font Pixel count: 9492 (3.09%)
	80 pixel tall Arial font, w/o seconds Pixel count: 2589 (0.84%)

Figure 5. Screen images and pixel count for various methods of displaying time.

Eliminating the second hand increased the time the watch could spend in sleep state from one second to sixty seconds as it was sufficient for the watch to wake up once a minute to update the hands rather than once a second.

We examined all the different screens presented by the application software, making changes to convey the same information with fewer pixels. Fonts, icons and graphics shown on the OLED are all designed to minimize the number of pixels that are turned on. On an emissive display with a fine dot pitch, we find that fine lines (even those that are just one pixel wide), are quite visible. Accordingly, fonts that use thin lines are preferred to bold fonts. Icons are designed to use outlines instead of solid fills so that fewer pixels are turned on. Thin line graphics are used whenever possible.

In a memory constrained device such as ours, we first concentrate on reducing the size (memory footprint) of the libraries and application executables to ensure that they fit. However, often there is a trade-off between size and the dynamic number of instructions executed to accomplish a certain task. For instance we can get code to execute faster (i.e., fewer number of instructions) if we use macros instead of functions or expand functions in-line. Reducing the number of instructions to perform a certain task is desirable from an energy perspective since the processor spends a shorter duration in the active mode. However, the memory is generally far more precious to trade-off for the reduction in energy consumption, especially if the duty factor is small to begin with. As seen in the graphs in Figure 3, the battery life curves are relatively flat when the duty factor is small.

Application software optimizations that result in fewer number of instructions executed without increasing the footprint are desirable. In particular, application software tries to avoid busy loops that wait for events to occur. Instead, the application relies on *select* calls into the Linux kernel, which in turn results in the kernel putting the application to sleep till the event occurs. When there are no active applications the kernel goes into the STANDBY mode as described above. So eliminating busy loops in the application helps the kernel save energy by going into the STANDBY mode quicker.

Select calls are also used by the application to get woken up at precise points of time in the future. When an application goes into a select call with a timeout, the kernel creates an entry on the timer list that gets triggered when after the desired time interval. As described above, our modifications to the Linux kernel look at the timer list to decide how long the processor stays in the STANDBY mode.

Reducing the duty factor for the OLED display saves power by reducing its brightness. At night (between the hours of 10PM to 6AM, say) the watch face can be dimmed automatically to a level configurable by the user.

We have three alternative ways in which the watch can receive data from other devices, infrared, Bluetooth™ and serial connection on the docking station/charger for the watch. The most convenient way of getting data into the watch is Bluetooth and is recommended when the battery is freshly charged. Infrared consumes less power than the Bluetooth and can be used at other times. When the user does not want to deplete the watch battery for getting data updates, putting the watch on the charger and using the docking station is recommended. We do not have power measurements on the Bluetooth module since they are early prototypes and do not implement all the power saving modes of the Bluetooth protocol.

Understanding the usage pattern for the device thoroughly can further help to reduce the power consumption. For example if the user sleeps regularly between 10PM and 6AM and does not look at or operate the watch during that interval, the display and the wireless communication subsystems could be turned off. Knowing the exact usage pattern is very important to help further reduce the duty factor of the device.

6 Future Work

In order to gracefully degrade the operation of the device as the battery is depleted we need to measure the remaining battery energy and also to define a set of degradation policies. So when the battery reaches below a certain threshold, the most power consuming subsystems may be disabled from operation and the processor may be allowed to operate only at lower frequencies to reduce the peak current drained from the battery.

For example, when the system is in sleep mode, most of the power is consumed by the display and the DRAM refresh. By definition we did not want the display to be turned off since it is annoying to glance at your watch and not see the time. Such a policy is fine when the battery is still reasonably charged. However, if the battery has been largely depleted, the system may shut the display off to conserve power as it may be more useful in this stage to prolong the time keeping ability. When this mode is entered, user would tap the watch face to turn the display

on. Since the display has turned blank it serves as a warning to the user to recharge or change his battery.

Another possibility for saving power in the sleep state is turning off the DRAM. This is not straightforward since there is a significant amount of dynamic state and data maintained in the DRAM and this needs to be saved. Issues such as how much energy is required to save the state in Flash and how long it takes to save and restore the data need to be explored.

Over time we expect the hardware components to operate at lower energy levels and also that battery capacity will improve. Using swappable batteries is one pragmatic solution. Several current processors operate at 2.5 V and so when they start operating at 1.0V at some point in the future the improvement for the active power consumed by the processor will be a factor of $2.5 \times 2.5 = 6.25$ since the power consumed by the circuits is proportional to the square of the voltage. Improved voltage regulators are also needed to cut regulator losses due to the mismatch in the circuit and battery voltages.

Another important question is whether the current required for the device to be in sleep mode can be provided by external means so that the user pays only for active use of the device and not for the sleep mode. For example if we consider solar energy that is incident of the watch face and attempt to use it to charge the watch battery, are we home free? The average solar energy incident on the surface of earth is about 164 W/m^2 accounting for variations in location, time of day, and seasons. The conversion efficiency of the best solar panels is around 9%. If we assume that the is $3 \text{ cm} \times 2 \text{ cm}$ watch face has transparent solar cells such as a Grätzelcell[15], and use the above power and efficiency numbers (precise efficiency numbers on transparent solar cells are not readily available), the amount of power that can be redirected towards charging the watch battery is $164 \times 0.1 \times 0.03 \times 0.02 = 9.84 \text{ mw}$. We need to derate this number further by a factor of fifty or so since the user is not out in the sun all the time. Overall, it appears we might soon be able to self power the sleep mode.

7 Conclusions

Power consumption is very important for wearable computers, and the smaller they get, the more significant this issue becomes, and as we have shown, a wrist watch formfactor is pushing the limit.

We studied the power consumption problem at several levels such as hardware, operating system scheduler, and the application level. We illustrated various trade-offs at these different levels and how they impact overall battery life. In many cases, we found that we had to trade-off energy efficiency for other factors such as function, size and usability, that were more important.

We have covered a significant distance in the race to design high function devices that are small, truly wearable and usable. We believe that such devices will achieve acceptable battery life times in the near future.

8 References

- 1 C. Narayanaswami, M. T. Raghunath, "Application Design for a Smart Watch with a High Resolution Display," *Proceedings of the International Symposium on Wearable Computing*, Atlanta, Georgia, 2000, pages 7-14.
- 2 J. Sanford and E. Schlig, "Direct view active matrix VGA OLED-on-crystalline-silicon display," *2001 SID International Symposium digest of technical papers*, Vol. XXXII.
- 3 Smailagic, A., Siewiorek, D., "System Level Design as Applied to CMU Wearable Computers", *Journal of VLSI Signal Processing Systems*, Kluwer Academic Publishers, Vol. 21, No. 3, 1999.
- 4 T. Starner, "Human powered wearable computing," *IBM Systems Journal*, Vol 35, Nos. 3&4, 1996.
- 5 Want, R., Schilit, B., Adams, N., Gold, R., Petersen, K., Goldberg, D., Ellis, J., and Weiser, M. "The PARCTAB Ubiquitous Computing Experiment," Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995
- 6 M. Newman, J. Hong, "A Look at Power Consumption and Performance on the 3Com Palm Pilot" <http://www.google.com/search?q=cache:guir.cs.berkeley.edu/projects/p6/finalpaper.html+standby+current+measurement+on+the+palm+pilot&hl=en>
- 7 Coutinho, L. "Power Supply of Palm Pilot," <http://www.massena.com/darrin/pilot/luiz/item7.htm>
- 8 Ulrich Hornstein, "Power Consumption of a Psion S5 8MB with CF," http://home.t-online.de/home/u.hornstein/ps_power_consumption.htm
- 9 J. Flinn, K. I. Farkas, and J. Anderson, "Power and Energy Characterization of the Itsy Pocket Computer (Version 1.5) Compaq Western Research Laboratory Technical Note TN-56, February, 2000
- 10 W. R. Hamburg, D. A. Wallach, A. Viredaz, L. S. Brakmo, C.A. Waldspurger, J.F. Bartlett, T. Mann, K. I. Furkas, Itsy: "Stretching the Bounds of Mobile Computing", pp. 28-36, *IEEE Computer*, April 2001
- 11 D. Linden, "Handbook of Batteries", 2nd Edition, McGraw Hill, 1994.
- 12 T. Martin and D. Siewiorek, "Non-Ideal Battery Behavior and Its Impact on Power Performance Trade-offs in Wearable Computing," *Proceedings 1999 International Symposium Wearable Computers*, San Francisco, CA, October 18-19, 1999.

- 13 J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications", *Proceedings 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, Louisiana, February, 1999.
- 14 Lithium Polymer battery discharge rates [Http: // www. idongho.com/battery.htm](http://www.idongho.com/battery.htm), [http://www.selfcharge.com /technicalb.html](http://www.selfcharge.com/technicalb.html), <http://battery.rnd.lgchem.co.kr/english/doc/oran-product2.html>, <http://www.chungpak.com/new2.htm>
- 15 ETRI developed nanoparticle oxide-based solar cells <http://www.etri.re.kr/nano.htm>
- 16 Cirrus EP7211 datasheet, <http://www.cirrus.com/pubs/ep7211-1.pdf?DocumentID=123>