LEAKAGE AWARE HARDWARE AND STOCHASTIC POWER SCHEDULING FOR SMART MOBILE DEVICES

Daniel Dilbie*
School of Electrical and Computer Engineering
Addis Ababa Institute of Technology, Addis Ababa University

ABSTRACT

A core requirement of effective and efficient management of energy is a good understanding of where and how the energy is used; how much of the system's energy is consumed by which parts of the system and under what circumstances.

In this work, a Smartphone is developed, hereafter referred to as the XLP, from the ground up with modular architecture where each module is supplied through an active switch matrix which takes the module to the lowest possible energy state. The basic notion of this architecture is achieving near zero-leakage power for modules which are not being used. This significantly reduces the idle power consumption which accounts for more than 60% of the average power consumed in smart phones [1, 2]. In addition to this approach on the hardware architecture, a stochastic dynamic power scheduling and on-demand power and clock gating policies are developed. A number of possible policies are presented and, under given conditions, one of them is proved to be optimal using the energy response time product (ERP) metrics.

The XLP is compared with three commercial smart phones: Openmoko Freerunner, HTC Dream and Google's Nexus One on similar tests and usage scenarios. The comparison is on performance versus power consumption. The XLP is proved to have the lowest power consumption (up to 3x lower) on competitive performance levels.

Keywords: *Zero-leakage; on-demand power gating; active switch matrix; ERP; profile-driven; XLP.*

INTRODUCTION

Power management is one of the most important considerations during the design of real-time systems, especially for battery operated systems due to the limited battery capacity. So, how to process tasks with less energy while guaranteeing all the time constraints is always a critical problem [3, 4]. Dynamic power management techniques leverage on the runtime characteristics to reduce power when systems are serving light workloads or are idle. In contrast, static power reduction methods such as synthesis of efficient hardware and compilation for low power are applied at design time.

In the case of many consumer electronics devices, especially mobile phones, battery capacity is severely restricted due to constraints on size and weight of the device. This implies that energy efficiency of these devices is very important to their usability. Hence, optimal management of power consumption in these devices is critical. At the same time, device functionality is increasing rapidly. Modern high-end mobile phones combine the functionality of a pocket-sized communication device with PC-like capabilities, resulting in what are generally referred to as smart phones. They integrate such diverse functionality as voice communication, audio and video playback, web browsing, short-message and email communication, media downloads, gaming and more. The rich functionality increases the pressure on battery lifetime, and deepens the need for effective energy management.

In this work the critical problem of energy management is addressed by designing and building a Smartphone. The design approach is to provide both hardware and software optimizations for aggressive power conservation whilst delivering the full system performance users opt for. The approach focuses on crunching down the static power consumption which accounts for more than half of the energy loss in mobile devices. Moreover, the most power hungry components in mobile systems are identified and a stochastic model of these components is developed to optimize the energy usage. Generally, the power management policies are designed to reduce the aggregate energy usage, not just the instantaneous power. This is only possible if one can reduce the product of the power required to do a given task and the time it takes to finish the task.

*E-mail: daniel@aait.edu.et

The hardware optimization involves creating a system with independently powered modules where every module has its own dedicated power supply line and a power switch which is controlled by the power processor. This allows the power processor to turn off an idle module or force it to optimum sleep state. A module at sleep or standby will consume significant energy over an extended time due to leakage, but if the supply is cutoff there will be no leakage and hence no power will be consumed by unused modules. Considering that they stay idle for most of the time, this approach will save a great deal of the energy in smart phones depending on the usage scenarios. The software optimization is mainly to tackle the dynamic power consumption of the system. Even though the validity of dynamic power management schemes in reducing the overall energy of the system is debatable, given the right scheduling algorithms and the available operating modes of the processor one can achieve significant savings in the system's energy usage. To validate this claim, a stochastic model is developed for most power consuming modules in the system. A module will have a set of operating states, S and corresponding power consumption, P_i in each state, but at a given time there is a state, $S_i \subset S$ with corresponding power consumption P_i and response time t_{si} (time required for transition from state S_i to the active state) for which the aggregate energy usage of the module will be the minimum. This algorithm is implemented as a core power scheduling function in the Linux dynamic power management kernel for the XLP.

RELATED WORKS

The concept of critical speed (or threshold speed), denoted as S_{cri}, is adopted by previous work on leakage aware scheduling [8, 9]. When processor is active (not idle), S_{cri} is defined as the available speed to execute a task with the minimum energy consumption. Either increasing or decreasing the processor speed to execute the job would consume more total energy than executing with S_{cri}. However, if all the jobs are executed with S_{cri}, the execution time to finish a job is extended and it will cause some jobs missing deadlines. In order to balance power management and time constraints, all the jobs would be executed with the speeds between critical speed and processor's maximum speed.

Several efforts have been reported in literature for CPU-centric techniques for profile-driven power management, while application aware strategies for energy centric partitioning and management will lead to efficient dynamic power management solutions for dedicated systems. The tradeoff between performance and low power consumption becomes unmanageable for general purpose applications [13].

A lot of conventional power management schemes are utilized in [7, 9, and 11]. The traditional voltage and frequency scaling accompanied by relatively new techniques, such as procrastination, adiabatic systems, low power processes, etc. are thoroughly discussed and reference design and implementations schemes are proposed. Most of them are applied at design and synthesis time for static power (leakage) reduction.

Zero-leakage design

When the processor is idle, which means the processor is not executing tasks in active mode, the major portion of the power consumption would be the static power. Static power consumption consists of the power consumed by the sub-threshold leakage and the reverse bias junction current. These two currents increase significantly with adaptive body biasing [12]. The static power per device is given by:

$$P_{stat} = V_{dd}\, I_{sub}\, |V_{bs}|\, I_j \qquad (1)$$

Where V_{dd} is the supply voltage, I_{sub} is the sub-threshold current, V_{bs} is the body bias voltage and I_j is the reverse bias junction current. It is apparent that if the unused components can be switched off independent of the other components, then it will be possible to reduce the static power consumption of the component to zero level. If the supply is cut off, (i.e. V_{dd} = 0), the individual transistors in the IC will not get biased and, therefore, will not have a body bias current which eliminates the static power consumption, so that P_{stat} = 0. Practically, we can't just turn off every idle module, because there will be a penalty of additional power to bring it back to active state and an extended response time. So, to get an optimum performance per power, we need to maximize the energy response time product. Some modules can be turned off others may be forced to a sleep mode or may be left idle whichever gives the smallest total energy usage.

Table1: Summary of the policies considered

NEVEROFF	Whenever the module goes idle, it remains idle until a job arrives.
INSTANTOFF	Whenever the module goes idle, it turns off. It remains off until there is no work to process, and begins to turn on as soon as work arrives.
SLEEP(S)	Whenever a module goes idle, it goes into the sleep state, S. It remains in sleep state until there is no work to process, and begins to wake up as soon as work arrives.

OPTIMAL POLICIES

There is a clear tradeoff between leaving idle modules on, and thus minimizing mean response time, versus turning idle modules off (or putting them to sleep), which hurts response time but may save power. Optimizing this tradeoff is a difficult problem, since there are an infinite number of possible management policies. The goal is to find a simple class of system management policies, which optimize (or nearly optimize) the above tradeoff. To capture the tradeoff involved in energy and performance, the Energy-Response time Product (ERP) metric, also known as the Energy-Delay Product (EDP) is utilized. For a given control policy π, the ERP is given by [6],

$$ERP^{\pi} = E[P^{\pi}]\,.\,E[T^{\pi}] \qquad (2)$$

Where $E[P^{\pi}]$ is the long-run average power consumed under the control policy π, and $E[T^{\pi}]$ is mean module's response time under policy π. Minimizing *ERP* can be seen as maximizing the "performance-per-watt", with performance being defined as the inverse of mean response time. While *ERP* is widely accepted as a suitable metric to capture energy-performance tradeoffs, this is the first work to analytically address optimizing the metric of *ERP* in battery powered mobile devices.

Policy definition

A specific set of management policies defined in Table 1 is considered and proved that it contains the optimal policy for the activity between the main applications processor and different other modules in the system.

To begin with, the communications modules are considered. Assume the job arrival process is Poisson with a known mean arrival rate. There is an infinite range of policies that one could consider for managing a single module, for example, when the module goes idle, one could immediately turn it off (INSTANTOFF), or alternatively, move the module to a specific sleep state (SLEEP). One could also just leave the module idle when it has no work to do (NEVEROFF). Another possibility is to turn an idle module off with some probability *p*, and leave it idle with probability *(1–p)*. One could also delay turning ON an OFF a module until a certain number of jobs have accumulated in a "queue". Also, when turning ON an OFF module, one could transition through SLEEP states, with each successive transition moving the module closer to the ON state. Within this wide range of policies, it can be shown that one of the policies, NEVEROFF, INSTANTOFF or SLEEP, is always optimal for a given module. For the passive modules, the INSTANTOFF policy is undoubtedly optimal, because there is no significant response time to bring them from OFF to ON. Therefore the passive modules: Audio module (codec and amplifier), display module (LCD panel and backlight) are managed by the INSTANTOFF policy.

PROCESS MODEL

Suppose jobs arrive from the application to the communications processor according to a Poisson process.

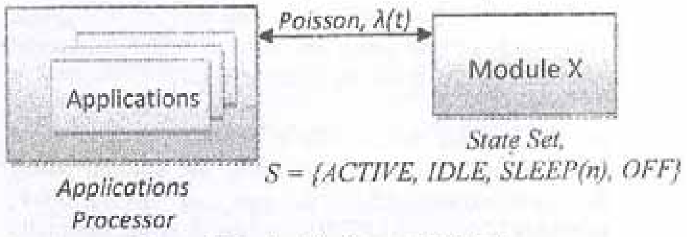

Figure 1 Stochastic Process Model

A fixed arrival rate, λ or a time-varying arrival rate, $\lambda\,(t)$ can be considered. Another assumption is that the job sizes are independent and identically distributed according to an exponentially distributed random variable *S*, with rate μ. The quantity $\rho\,(t) = \lambda\,(t).\,E[S]$ is used to denote the instantaneous load, or the rate at which work is entering the system at time *t*.

The module can be in one of the following states: active (busy), idle, off, or any one of $N - 1$ sleeps states: S_1, S_2, S_{N-1}. For convenience, the idle is sometimes referred to as S_0 and the off state as S_N. The associated power values are P_{ACTIVE}, P_{IDLE}, P_{S0}, P_{S1}, $P_{SN} = P_{OFF}$ the ordering is assumed to be: $P_{ACTIVE} > P_{IDLE} > P_{S1} > \ldots > P_{SN-1} > P_{OFF} = 0$. The module can only serve jobs in the active state. The time to transition from initial state, S_i to final state, S_f, is denoted by $T_{Si \to Sf}$ and it is a constant (not a random variable). Rather obviously, we assume $T_{ACTIVE \to IDLE} = T_{IDLE \to ACTIVE} = 0$. Further, the average power consumed while transitioning from state S_i to S_f is given by $P_{Si \to Sf}$

For analytical tractability, the above model will be relaxed a little. It will be assumed that the time to transition from a state to any state with lower power is zero.

Therefore, $T_{ON \to OFF} = T_{Si \to OFF} = 0$, *for all i*. This assumption is justified because the time to transition back to a higher power state is generally considerably larger than the time to transition to the lower power state, and hence dominates the performance penalties. Further, it will be assumed that the time to transition from a state S_i to any higher power state is only dependent on the low power state, and this is denoted as T_{Si}.

Therefore, $T_{OFF \to IDLE} = T_{OFF \to Si} = T_{OFF}$ for all i. Note that $0 = T_{IDLE} < T_{S1} < \ldots < T_{SN-1} < T_{OFF}$ This assumption is justified because in current implementations there is no way to go between two sleep states without first transitioning through the IDLE state. Regarding power usage, it is assumed that when transitioning from a lower power state, S_i, to a higher power state S_f, the power consumed is $P_{Si \to Sf} = P_{ACTIVE}$. The results of this paper are derived under the model assumptions. So, it can be concluded as:

For the process model with a Poisson (λ) arrival and exponentially distributed job sizes, the optimal policy for minimizing ERP is one of NEVEROFF, INSTANTOFF or SLEEP(S), where S is the optimally chosen sleep state among the existing sleep states.

The above statement can be described as follows:

Let Π_{mixed} denote the class of randomized policies whereby a module immediately transitions to power state S_i $(i \in \{0, \ldots, N\})$ with probability p_i on becoming idle. Given that the module went into power state S_i with probability q_{ij} it stays in S_i and waits until j jobs accumulate in the queue, where $\sum_{j=1}^{\infty} q_{ij} = 1$. Once the target number of jobs has accumulated, the module immediately begins transitioning to the ON state, and stays there until going idle.

Now consider a policy $\pi \in \Pi_{mixed}$. The mean response time for policy π under a Poisson (λ) arrival process with $Exp(\mu)$ job sizes is given by:

$$E[T] = \frac{\sum_{i=0}^{N} p_i \sum_{j=1}^{\infty} q_{ij} r_{ij}}{\sum_{i=0}^{N} p_i \sum_{j=1}^{\infty} q_{ij}\left(j + \lambda T_{S_i}\right)} \tag{3}$$

Where,

$$r_{ij} = \frac{j + \lambda T_{S_i}}{\mu - \lambda} + \left[j T_{S_i} + \frac{j(j-1)}{2\lambda} + \frac{\lambda T_{S_i}^2}{2} \right] \tag{4}$$

and the average power for policy π is given by:

$$E[P] = \frac{\sum_{i=0}^{N} p_i \sum_{j=1}^{\infty} q_{ij}\left[j\left(\rho P_{ON} + (1-\rho) P_{S_i}\right) + \lambda T_{S_i} P_{ON} \right]}{\sum_{i=0}^{N} p_i \sum_{j=1}^{\infty} q_{ij}\left(j + \lambda T_{S_i}\right)}$$

Therefore, assuming a Poisson (λ) arrival process, and Exp (μ) job sizes, the mean response time and mean power for NEVEROFF, INSTANTOFF and SLEEP are given by:

$$E[T] = \frac{1}{\mu - \lambda} + \frac{T_{S_i}\left(1 + \lambda T_{S_i}/2\right)}{1 + \lambda T_{S_i}} \tag{5}$$

$$E[P] = \frac{\rho P_{ON} + (1-\rho) P_{S_i} + \lambda T_{S_i} P_{ON}}{1 + \lambda T_{S_i}} \tag{6}$$

Where S_i = IDLE for NEVEROFF, S_i = OFF for INSTANTOFF, and S_i is the sleep state that the module transition to in SLEEP.

The proof for this optimal policy definition is given in the appendix.

HARDWARE IMPLEMENTATION

A new Smartphone is designed and built with the novel architecture discussed above. An ARM11 based SoC is used as the main processor with integrated, but separately powered graphics and multimedia engines. The ARM core is powered as low as 0.8V in deep sleep state. A high density low power 256MB mobile DDR RAM is used as system memory and a 2GB NAND flash as storage memory. For wireless connectivity, a 2.5G cellular radio, 54Mbps Wi-Fi and Bluetooth 3.0 capable devices are used. The display system is built around 3.5" 16 million color TFT LCD panel with resistive touch screen and white LED backlight. All the necessary sensors, such as accelerometer, ambient light and temperature sensors are included for context awareness. Most importantly an intelligent Power management logic is built to control the power flow to all the modules on board.

SOFTWARE IMPLEMENTATION

The algorithm is designed to determine the next state of the different modules in the system that yields the minimum ERP for a predefined (expected) response time and energy. The PMU then updates the states by actuating the switches and/or passing control commands.

The pseudo-code below shows a high level implementation of the DPM assuming a single application is running.

```
While(app_is_running)

 Aquire::
 //get profile updates and
 Get_profile(*app,*Pbuffer);
 //read history
 refer_history(*app,*Hbuffer,ta)
 /*tag gives the option as
  recent, modified, or all.*/

 Process::
  determine_modules_in_use;
  get_performance_index (t, P);
  optimize_tradeoff{t,P,E};
  //here using ERP
  predict_for_next_iteration;

 Update::
  update_switch_matrix_state;
  update_cpu_operating_point
  (voltage, Core_freq, bus_freq);

 Save::
  save_state_for_next_iteration;
  Update_global_history
  (if_change_is_significant);
  /* wait for the update cycle
   to finish and continue */
```

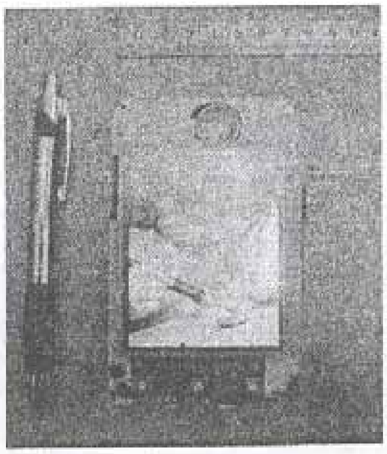
Figure 2 Photo of XLP

EXPERIMENTAL RESULTS

Prior to running any test, it would be good to establish the baseline power state of the device, when no applications are running. There are two different cases to consider: suspended and idle. For the idle case, there is also the application-independent power consumption of the backlight to consider.

Suspended Mode

A mobile phone will typically spend a large amount of time in a state where it is not actively used. This means that the application processor is idle, while the communications processor performs a low level of activity, as it must remain connected to the network be able to receive calls, SMS messages, etc. As this state tends to dominate the time during which the phone is switched on, the power consumed in this state is critical to battery lifetime.

The power management driver in the Linux kernel running on the application processor aggressively suspends to RAM during idle periods, whereby all necessary state is written to RAM and the devices are put into low-power sleep modes (where appropriate). To quantify power use while suspended, the device is forced to sleep (pressing sleep/wake key) and the power is measured over a 120 second period. Fig. 3 shows the results, averaged over 10 iterations. The average aggregate power is 68.6mW, with a relative standard deviation (RSD) of 8.2 %. The large fluctuations are largely due to the GSM (14.4% RSD). The GSM subsystem power clearly dominates while suspended, consuming approximately 50% of the overall power.

Despite maintaining full state, RAM consumes negligible power—less than 3mW.

Idle Mode

The device is in the idle state if it is fully awake (not suspended) but no application is running. This case constitutes the static contribution to power of an active system. This case is run with the backlight turned off, but the rest of the display subsystem enabled. Fig 4 shows the power consumed in the idle state. As with the suspend benchmark, 10 iterations was made, each of 120 seconds in the idle state. Power consumed in this state was very stable, with an RSD of 2.6 %, influenced largely by GSM, which varied with an RSD of 30%. All other components showed an RSD below 1%.

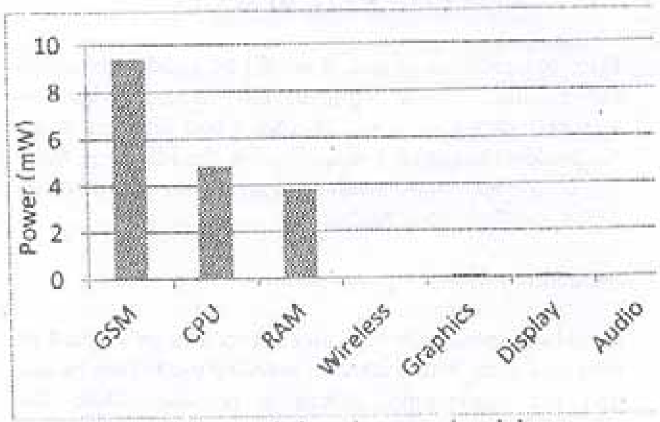

Figure 3 Suspended mode power breakdown. The aggregate power is 18.3mW.

Display and Backlight

It is measured how the content displayed on the LCD affected its power consumption: 16.55mW for a completely white screen, and 37.1mW for a black screen. Display content can therefore affect overall power consumption by up to 21mW.

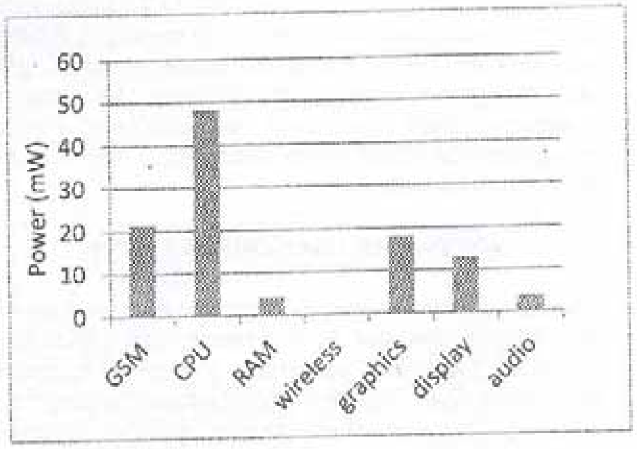

Figure 4 Idle mode power breakdown with backlight turned off. Aggregate power is 108.6mW.

Figure 5 shows the power consumed by the display backlight over the range of available brightness levels. That level is an integer value between 0 and 100 programmed into the power-management module, used to control backlight current. The minimum backlight power is 0mW- at brightness level of 0 (off), the maximum 288mW, and the default corresponds to a brightness level of 50, consuming 48.2mW.

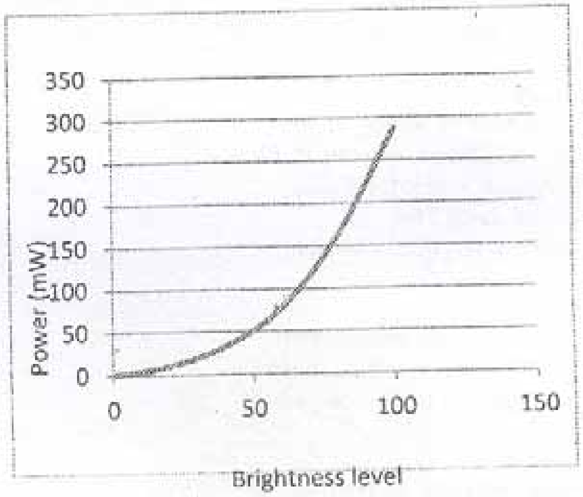

Figure 5 Brightness level and corresponding power consumption of the backlight

MICRO BENCHMARKS

Micro-benchmarks are used to determine the contribution to overall power from various system components. Specifically the benchmarks encompass the application processor (CPU and memory), the flash storage devices, and the network interfaces

CPU and RAM

The power consumption of the CPU and RAM is measured by running the Freescale utilities which includes: frequency and voltage scaling, bus speed governor, memory test, and core utilities (provide user space procedures to debug and reconfigure the ARM core). The tests also consider memory bound and CPU bound conditions.

The ARM core is run at different frequencies and corresponding bus frequency for DDR SDRAM interface. The ARM core frequencies, core voltage and bus frequencies experimented in this test are given below in Table 2.

Table 2: Voltage and frequency scaling values

CPU frequency (MHZ)	Core voltage (V)	Bus speed (MHZ)
40	0.800	40
100	1.000	64
320	1.175	180
400	1.275	210
480	1.375	360
533	1.520	400

Flash Storage

Storage on XLP is provided by 1GiB of internal NAND flash, and an external micro Secure Digital (SD) card slot. To measure their maximum power consumption, the Linux dd program is used to perform streaming reads and writes. For reads a 64 MiB file is copied, filled with random data, to /dev/null in 4 KiB blocks. For writes, 8MiB of random data was written, with an fsync between successive 4KiB blocks to ensure predictability of writes. Between each iteration the page cache is flushed. Fig. 6.4 shows the power consumed by the NAND flash and SD card, as well as the CPU and RAM, averaged over 10 iterations of each workload. Table 3 shows the corresponding data throughput, efficiency (including NAND/SD power and the CPU and RAM power to support it), and idle power consumption. The power and throughput RSD is less than 5% in all cases.

Table 3: Flash storage power and performance

Metric	NAND	SD
idle	0.1mW	0.32mW
Read		
Throughput (MiB/s)	5.4	2.85
Efficiency (MiB/J)	73.1	48
Write		
Throughput (KiB/s)	1124.3	380
Efficiency (MiB/J)	18	8.2

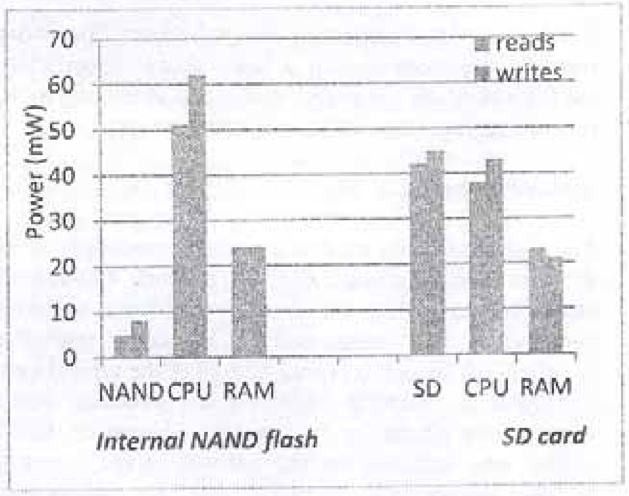

Figure 6 Flash storage power consumption

Network

In this case only the GPRS is used to connect to the network as the Wi-Fi chip is not populated on the device. The test consisted of downloading a file via HTTP. The files contained random data, and were 50KiB of size. The results of 10 iterations of the benchmark are shown in Fig. 7. It showed a throughput of 5.8 ± 1.2 KiB/s over GPRS. However, it shows power consumption far exceeding the contribution of the RAM and CPU. Despite highly-variable throughput, GPRS showed relatively consistent power consumption with an RSD of approximately 2 %.

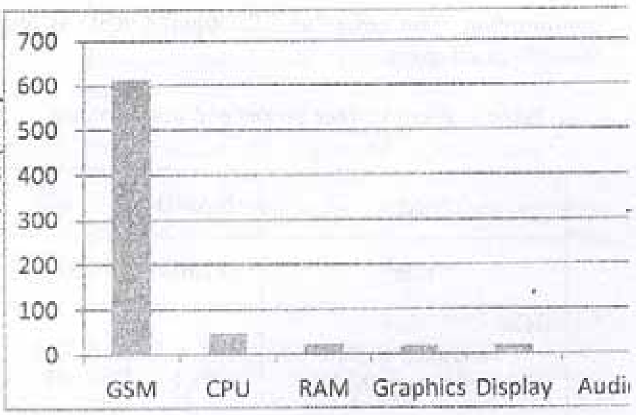

Figure7 Network power breakdown. Aggregate power is 715.1mW

USAGE SCENARIOS

These are used to determine power consumption under typical usage scenarios of a Smartphone. Specifically the following are examined: audio and video playback, text messaging, voice calls, and web browsing.

Audio Playback

The system is being used as a portable media player in this test. The sample music is a 12.3MiB, 400-second stereo 44.1 kHz MP3, with the output to a pair of stereo headphones. The measurements are taken with the backlight off (which is representative of the typical case of someone listening to music or podcasts while carrying the phone in their pocket). However, GSM power was included, as the realistic usage scenario includes the phone being ready to receive calls or text messages. The audio file is stored on the SD card. Between successive iterations the buffer cache is flushed to ensure that the audio file was re-read each time.

Figure 8 shows the power breakdown for this benchmark at maximum volume for 10 iterations.

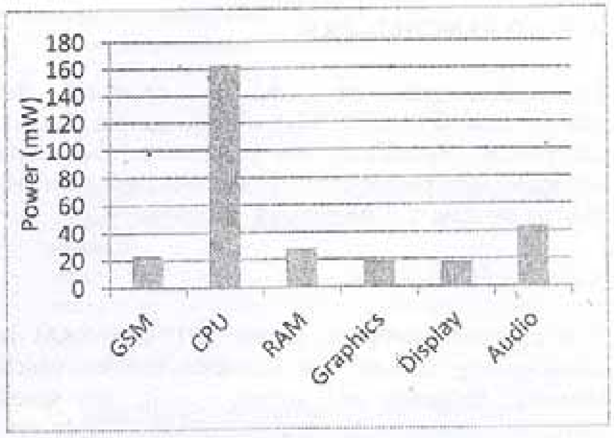

Figure 8 Audio playback power breakdown. Aggregate power is 294.0mW

Video Playback

The power requirements for playing a video file was also determined to complete the media player usage scenario. Here a 7 minute, 18MiB MPEG-4-encoded video clip (no sound) was played with a command line media player with G-streamer backend. Again a flush of the buffer cache was forced between iterations. The power averaged over 10 iterations is shown in Fig. 9. The backlight power is included in the results with brightness levels of 20%, 50%, 75% and 100%. GSM power is again included. The energy cost of loading the video from the SD card is negligible, with an average power of 2.4mW over the length of the benchmark.

Voice Call

The power consumption of the device when making a GSM phone call is shown in Fig.10. The benchmark is trace-based, and includes dialing a number, and making a 50-second call. The dialed device was configured to automatically accept the call after 3 seconds. Thus, the time spent in the call was approximately 40 seconds, assuming a 7-second connection time. The total benchmark runs for 60 seconds. GSM power clearly dominates in this benchmark at 832mW ± 89mW. Note that the average power consumed by the backlight is lower than in other benchmarks, since it is disabled during the call. The backlight is active for approximately 25% of the total benchmark.

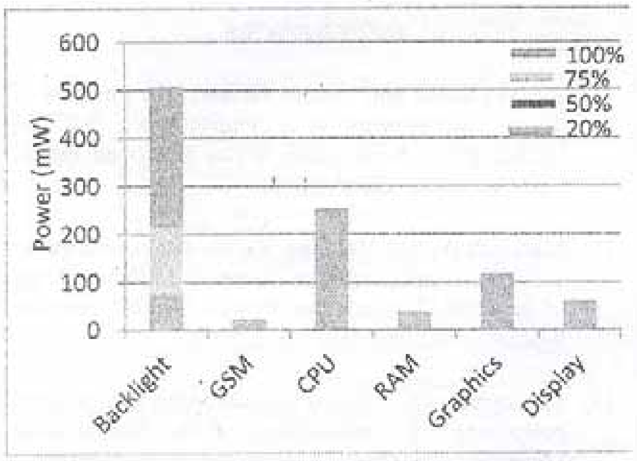

Figure 9 Video playback power breakdown. Aggregate power is 492mW

Text Messaging

The cost of sending an SMS is benchmarked by using a trace of real phone usage. This consists of loading the contacts application and selecting a contact, typing and sending a 32-character message, lasting a total of 60 seconds. To ensure the full cost of the GSM transaction is included, the power is measured for an additional 20 seconds. The average result of 10 iterations of this benchmark is shown in Fig. 11. Again, the power for four backlight brightness levels is shown. Power consumed is dominated by the display components. The GSM radio shows an average power of 62 ±19mW, accounting for 28% of the aggregate power (excluding backlight).

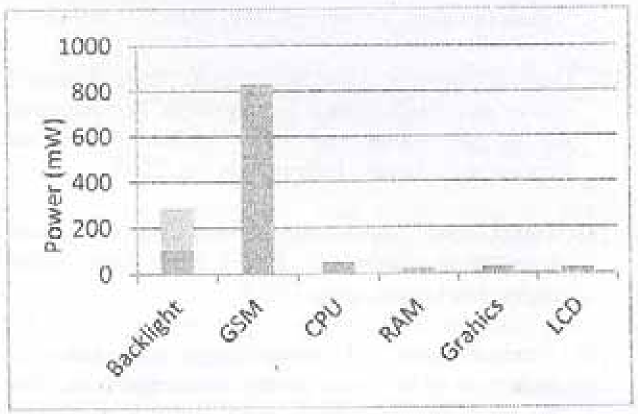

Figure 10 Power usages during voice call. Aggregate power consumed is 952mW

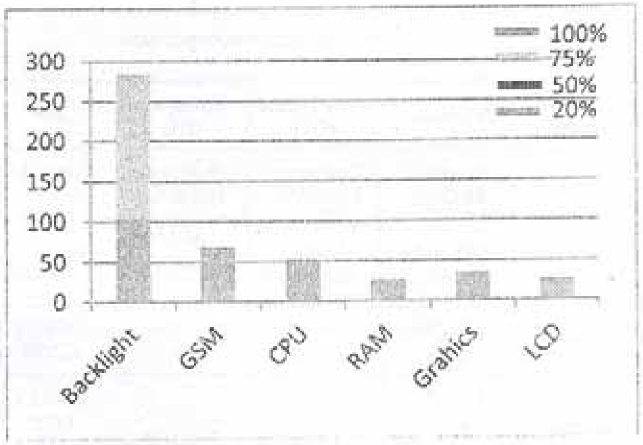

Figure 11 Power breakdown for using SMS. Aggregate power consumed is 206mW

PERFORMANCE COMPARISON

The XLP's power consumption and performance is compared with modern commercial smart phones. The choice of devices to be compared with the XLP is based on the availability of detailed breakdown of power and performance benchmarks and the similarity of hardware and software features of these devices [1].

Comparison Validations

Like the cases where the power consumption of the XLP was probed, the three devices under test are also subject to the same scenarios and their power usage is recorded. To make a decisive convection of the low power features of the XLP, the differences are taken out and plotted independently and only the similar test cases and results are plotted and compared. The additional power consumed or saved by the Android OS is negligible in all the benchmarks where the comparison is validated.

The devices under comparison are Google's Nexus one (N1), HTC dream (G1) and Openmoko Freerunner. Note that the Nexus one has an additional power consumption of the OLED display which can rise up to 1131mW at full brightness. Table 4 summarizes the basic modules (features) of the devices under consideration.

Table 4: Devices under comparison

Component	Freerunner	HTC dream (G1)	Nexus one (N1)	*XLP*
SoC	Samsung S3C2442	Qualcomm MSM7201	Qualcomm QSD8250	Freescale i.MX335
CPU	ARM926T 400MHz	ARM11 528MHz	ARM Cortex_A8 1GHz	ARM11EJ-S 533MHz
RAM	128MiB SDR	192MiB mDDR	512MiB mDDR	256MiB mDDR
Display	3.5" TFT LCD 480x640	3.2" TFT LCD 320x480	3.7" OLED 480x800	3.5" TFT LCD 480x320
Cellular	2G	3G	3G	2.5G
Kernel	Linux 2.6.29	Linux 2.6.29	Linux 2.6.29	Linux 2.6.31

Table 5 below shows the power consumption of these devices under the different scenarios discussed above. In all the cases, except voice call XLP is proved to have the lowest power consumption. The additional power on the voice call is attributed to the inferior power management in the 2G baseband compared to the more advanced 3G power management techniques.

Table 5: Average power consumption comparison (LCD backlight turned off)

Benchmark	Average power consumption (mW)			
	Freerunner	G1	N1	*XLP*
Suspend	103.2	26.6	24.9	***18.3***
Idle	333.7	161.2	333.9	***108.6***
Audio	419.0	459.7	322.4	***294.2***
Video	558.8	568.3	526.3	492.*0*
Voice call	1135.4	822.4	764.8	***951.8***
Web	500.0	430.4	538.0	

REFERENCES

[1] Aaron Carroll and Gernot Heiser, "*An analysis of power consumption in a Smartphone*", NICTA, University of New South Wales and open kernel labs, Australia, March 2010.

[2] Mashesri A. and Vardhan V. "*Power consumption breakdown on a modern laptop.*" In Proceedings of the 2004 Workshop on Power-Aware Computer Systems, Portland, OR, USA, 2004.

[3] Sagahyroon, A. "*Power consumption in handheld computers.*" In Proceedings of the International Symposium on Circuits and Systems, Dec. 2006.

[4] Bircher, W. L. and John L. K. "*Analysis of dynamic power management on multi- core Processors.*" In Proceedings of the 22nd International Conference on Supercomputing Island of Kos, Greece, June 2008.

[5] Bircher, W. L. and John, L. K. "*Complete system power estimation: A trickle-down approach based on performance events.*" In Proceedings of the IEEE International symposium on Performance Analysis of Systems and Software, San Jose, CA, Apr. 2007.

[6] Anshul Gandhia, Varun Guptaa, Mor Harchol-Baltera and Michael A. Kozuchb, "*Optimality Analysis of Energy-Performance Trade-off for Server Farm Management.*" Computer Science Department, Carnegie Mellon University and Intel research center, Pittsburgh, PA, 2010.

[7] Gordon Gammie, Alice Wang, et al "*SmartReflex™ power and Performance management Technologies for 90 nm, 65nm and 45nm Mobile application Processors*". Texas Instruments Inc, 2008.

[8] Turbo Boost Technology in Intel® Core™ micro architecture (Nehalem) Based Processors, white paper, Intel corp., march 2008.

[9] Peter Nilsson, "*A methodology for arithmetic reduction of the static power consumption verified on filter architectures.*", IEEE 2008 International conference on Microelectronics.

[10] Shein-yang Wu, et al. "*A highly manufacturable 28nm CMOS low power platform technology with fully functional 64Mb SRAM using dual/tripe gate oxide process*". IEEE 2009 symposium on VLSI technology.

[11] Jian-Jia Chen and Tei-Wei Kuo, "*Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems*," Proceedings of the 2007 IEEE/ACM international conference on Computer aided design, Piscataway, NJ, USA, 2007.

[12] Ravindra Jejurikar and Rajesh Gupta, "*Dynamic slack reclamation with Procrastination scheduling in real-time embedded systems*," 42nd annual conference on Design automation, New York, 2005.

[13] Magesh Kumar, et al "*Profile-Based Technique for Dynamic Power Management in Embedded Systems*", IEEE 2008 International Conference on Electronic Design, Penang Malaysia, Dec. 2008.

[14] H. Mair, et al. "*A 65-nm mobile multimedia applications processor with an adaptive power management scheme to compensate for variations*" Texas Instruments Inc. Digest of technical Paper IEEE 2007 Symposium on VLSI circuits.

[15] Abul Sarwar "*CMOS Power Consumption and C_{pd} Calculation*" Texas Instruments Inc. Applications report, 1997.

[16] Yongwen Pan, Man Lin "*Dynamic leakage aware power management with procrastination method*" IEEE 2009 International conference on electronic design.

[17] Dan Hillman "*Using Mobilize Power Management IP for Dynamic & Static Power Reduction In SoC at 130 nm*", Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2005.

[18] Peter Nilsson "*Approaching Green Electronics: Power Efficient Arithmetic in Nano-scale CMOS*" IEEE 2010 international conference on green circuits and systems, 21-23 June 2010, Shanghai, China.

APPENDIX

Proof for the optimal policy definition

First, note that if the module is in the ON state and there is work in the system, then the optimal policy never transitions into a sleep state. Suppose, by contradiction, an optimal policy π transitioned into a sleep state at time t_0 with work in the queue and then later transitioned through some SLEEP state until finally transitioning to the ON state at time t_1. This could be transformed into a policy π' with equivalent power consumption, but lower mean response time by deferring the powering down until all the work present in the system at t_0 has finished (say at t_2), and then transitioning through the same sleep states as π, finally transitioning to the ON (or IDLE) state at time $t_2 + (t_1 - t_0)$.

Next, we prove that the only instants at which an optimal policy takes actions will be job completions, job arrivals, or when the module finishes transition from a low power state to a higher power state. Here it's assumed that once a transition to a SLEEP, IDLE or ON state has been initiated from a lower power state, it cannot be interrupted. We have already argued that no actions happen during a busy period when the module is in the ON state. Therefore to prove that control actions only happen at the claimed events, it remains to show that actions do not occur while the module is in IDLE or SLEEP states (and not in transition or ON) and an arrival has not occurred. To achieve this, it suffices to show that there exists a Markovian optimal control for the ERP metric.

Note that,

$$E[T] = \lim_{T\to\infty} \frac{1}{\lambda T}\, E\left[\int_{t=0}^{T} N(t)dt\right], \text{ and}$$

$$E[P] = \lim_{T\to\infty} \frac{1}{T}\, E\left[\int_{t=0}^{T} P(t)dt\right].$$

Where *N(t)* and *P(t)* denote the number of jobs and power consumption, respectively, at time *t*. Thus the optimal decision at time *t* depends only on the future evolution of the system, and not on the finite history in *[0, t]*. (Note that these statements are not true if we replace *E[T]* and *E[P]* by their discounted versions. By the memory-less property of the Poisson arrival process, the claim follows.

The proof proceeds via renewal reward theory. Define a renewal cycle for the module as the time from when a module goes IDLE (has zero work), until it next goes IDLE again. Thus we can express:

$$\frac{E[\textit{total response time per cycle}]}{E[\textit{number of jobs per cycle}]};$$

$$\frac{E[\textit{total energy per cycle}]}{E[\textit{duration per cycle}]}$$

Now consider a specific case, where the server goes into sleep state S_i on becoming IDLE, and starts transitioning to the on state when n_i jobs accumulate. There can be more arrivals while the module is turning on. Let's denote the number of arrivals during transition from S_i by X_i, and note that X_i is distributed as a Poisson random variable with mean λTS_i . Thus, after the module turns on, it has $n_i + X_i$ jobs in the queue, and thus the time until the module goes idle is distributed as a sum of $n_i + X_i$ busy periods of an M/M/1 system. The sum of the response times of jobs that are served during this renewal cycle has two components:

1. Sum of waiting times of all jobs before the module turns on (term 1 below): The waiting time of the jth of the first n_i jobs is $\sum_{k=j+1}^{n_i} T_\lambda(k) + T_{S_i}$, where $\{T_\lambda(.)\}$ are i.i.d. Exp(λ) random variables, and $T_\lambda(k)$ denotes the time between the (k - 1)st and kth arrival of the cycle. By the properties of the Poisson arrival process, the (unordered) waiting time of each of the X_i jobs is an independent U([0, T_{Si}]) random variable. Adding and taking expectation, we get term1 as shown in (*).

2. Sum of the response times from when the module turns on until it goes IDLE (term 2 below). Since the sum of response time of the jobs that are served during the renewal cycle is the same for any non-preemptive size-independent scheduling policy, we will find it convenient to schedule the jobs as follows: We first schedule the first of $n_i + X_i$ arrivals and do not schedule any of the $n_i + X_i - 1$ remaining jobs until the busy period started by the first job completes. Then we schedule the second of the $n_i + X_i$ jobs, holding the remaining jobs until the busy period started by this job ends, and so on. The sum of the response times is thus given by the sum of response times in $n_i + X_i$ i.i.d. M/M/1 busy periods, and the additional waiting time experienced by the initial n_i+X_i arrivals. By renewal theory, the expectation of the sum of response times of the jobs served in an M/M/1 busy period with arrival rate λ and service rate μ is given by the product of the mean number of jobs served in a busy period $\left(\frac{1}{1-\frac{\lambda}{\mu}}\right)$

and the mean response time per job $\left(\frac{1}{\mu-\lambda}\right)$. This gives the first component of term 2.

The additional waiting time of the j^th^ of the $n_i + X_i$ initial arrivals due to our scheduling policy is given by the sum of durations of j – 1 M/M/1 busy periods, each of expected length $\frac{1}{\mu-\lambda}$. Adding this up for all the $n_i + X_i$ jobs and taking expectation, we get the second component of term 2.

$$\overbrace{n_i\left(\frac{n_i-1}{2\lambda}+T_{S_i}\right)+E[X_i]\frac{T_{S_i}}{2}}^{\text{Term 1}}+\underbrace{\frac{1}{1-\rho}\cdot\frac{n_i+E[X_i]}{\mu-\lambda}+E\left[\frac{(n_i+X_i)(n_i+X_i-1)}{2(\mu-\lambda)}\right]}_{\text{Term 2}} \quad (*)$$

$$=\frac{1}{1-\rho}\left(\frac{n_i+E[X_i]}{\mu-\lambda}+\left[n_iT_{S_i}+\frac{n_i(n_i-1)}{2\lambda}+\frac{\lambda T_{S_i}^2}{2}\right]\right)=\frac{r_{in_i}}{1-\rho}$$

The final expression is obtained by combining the above with the renewal reward equation, and noting that the mean number of jobs served in this renewal cycle is given by $\frac{n_i+E[X_i]}{1-\rho}$.

$$E[T]=\frac{E[total\ response\ time\ per\ cycle]}{E[number\ of\ jobs\ per\ cycle]}=$$

$$\frac{\sum_{i=0}^{N}P_i\sum_{n_i=1}^{\infty}q_{in_i}\frac{r_{in_i}}{1-\rho}}{\sum_{i=0}^{N}P_i\sum_{n_i=1}^{\infty}q_{in_i}\frac{n_i+\lambda T_{S_i}}{1-\rho}}=$$

$$\frac{\sum_{i=0}^{N}P_i\sum_{j=1}^{\infty}q_{ij}r_{ij}}{\sum_{i=0}^{N}P_i\sum_{j=1}^{\infty}q_{ij}\left(j+\lambda T_{S_i}\right)}$$

The proof for *E[P]* is analogous. The duration of a cycle is composed of three different times:

1. Time spent waiting for n_i jobs to queue up: The expected duration is $\frac{n_i}{\lambda}$, with expected total energy consumed given by $\frac{n_i}{\lambda}P_{S_i}$.
2. Time to wake up the module: This is T_{S_i} with total energy consumed by the module during this time as $T_{S_i}P_{ON}$.
3. $(n_i + X_i)$ busy periods: The expected time it takes for the module to go idle again is the expected duration of $n_i + X_i$ busy periods, given by $\frac{n_i+\lambda T_{S_i}}{\mu-\lambda}$ with total energy consumed being $\frac{n_i+\lambda T_{S_i}}{\mu-\lambda}P_{ON}$

Thus, we have:

$$\frac{E[Total\ energy\ per\ cycle]}{E[duration\ per\ cycle]}=$$

$$\frac{\sum_{i=0}^{N}P_i\sum_{j=1}^{\infty}q_{ij}\left[\frac{j}{\lambda}P_{S_i}+T_{S_i}P_{ON}+\frac{j+\lambda T_{S_i}}{\mu-\lambda}P_{ON}\right]}{\sum_{i=0}^{N}P_i\sum_{j=1}^{\infty}q_{ij}\left[\frac{j}{\lambda}+T_{S_i}+\frac{j+\lambda T_{S_i}}{\mu-\lambda}\right]}$$

$$=\frac{\sum_{i=0}^{N}P_i\sum_{j=1}^{\infty}q_{ij}\left(j\left(\rho P_{ON}+(1-\rho)P_{S_i}\right)+\lambda T_{S_i}P_{ON}\right)}{\sum_{i=0}^{N}P_i\sum_{j=1}^{\infty}q_{ij}\left(j+\lambda T_{S_i}\right)}$$