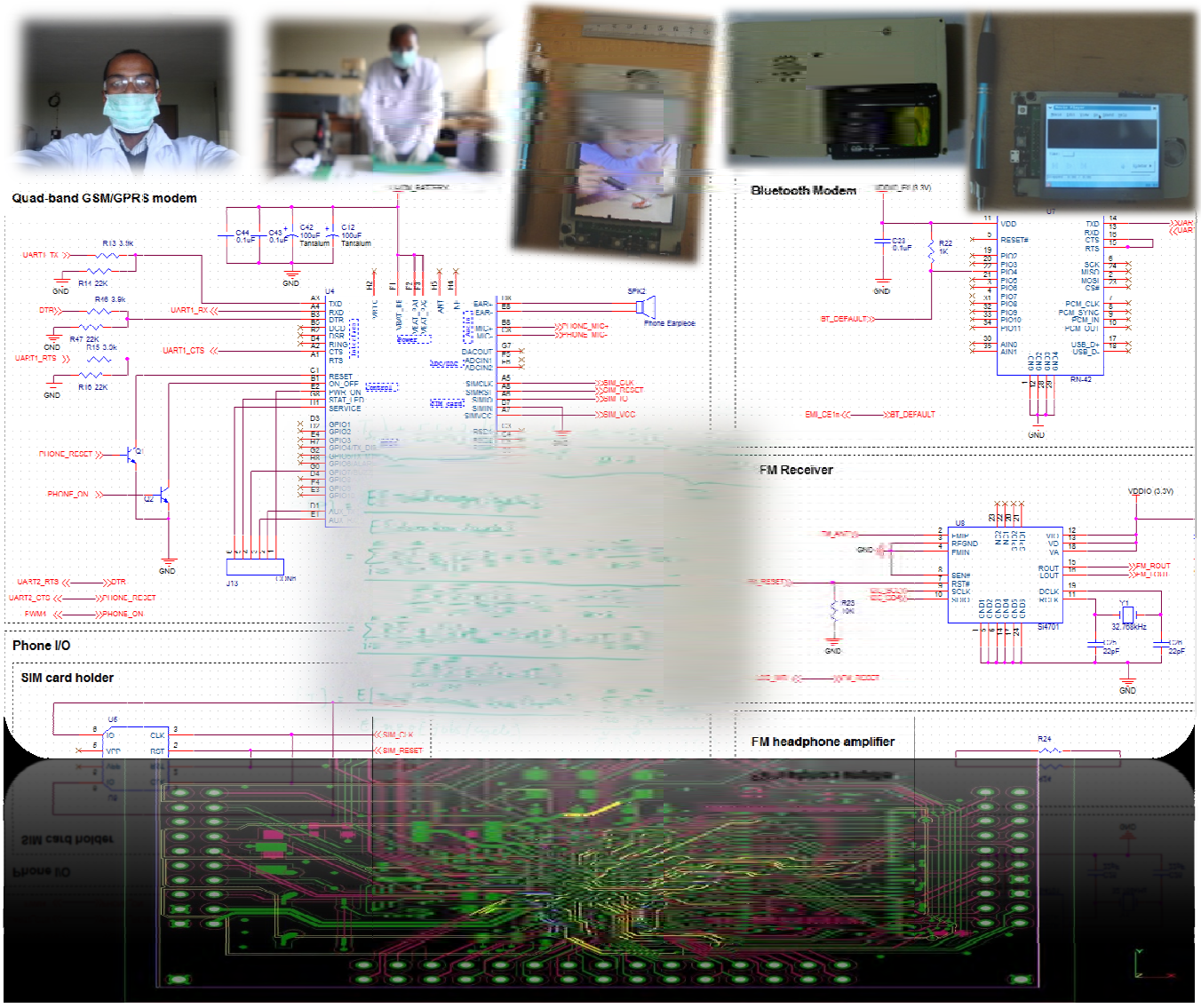
Leakage Aware Hardware Architecture and Dynamic Power Scheduling For Mobile Devices



By: Daniel Dilbie

Department of Electrical and Computer Engineering,
Microelectronic Engineering Stream

Addis Ababa Institute of Technology

Addis Ababa University

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Leakage Aware Hardware Architecture and Dynamic Power Scheduling for Mobile Devices

A thesis submitted to School of Graduate Studies and Research in Partial Fulfillment of the Requirement for the Masters Degree

By: Daniel Dilbie Tessema

July, 2011

Declaration

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

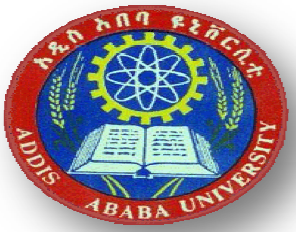
Name: Daniel Dilbie Signature: _____

Place: Addis Ababa, Ethiopia

Date of Submission: _____

This thesis is submitted for examination with my approval as university advisor.

Advisor: Dr. Getachew Alemu Signature: _____



Addis Ababa University

Addis Ababa Institute of Technology

Department of Electrical and Computer Engineering

Leakage Aware Hardware Architecture and Dynamic Power Scheduling for Mobile Devices

By: Daniel Dilbie Tessema

APPROVED BY BOARD OF EXAMINERS

_____	_____
Chairman, Department of Graduate Committee	Signature
Dr. Getachew Alemu _____	_____
Advisor	Signature
_____	_____
Internal Examiner	Signature
_____	_____
External Examiner	Signature

Acknowledgements

Designing modern, cutting-edge electronic systems is not something people living in underdeveloped nations, like Ethiopia would like to do mainly because of the unavailability of resources, facilities, and shortage of expertise. To make matters worse, the financial system doesn't allow us to buy stuff online. Even, the sand-cheap capacitors and resistors, we can neither get them here nor can we buy them from abroad in a regular way.

Regardless of all the challenges and the difficulties, the successful completion of this research project is not a result of merely my efforts, but a combined effort and dedication of several individuals, companies and institutions. It is well known that a comprehensive work like this which includes research, design and implementation could not be accomplished by an individual.

My deepest gratitude goes to my friend Abebe Tsegaye, who was studying at Northeastern University in Boston, MA while I was doing this research. He was always there to take my orders, purchase whatever I wanted and send over to me promptly. I would like to thank my father who was constantly motivating and forcing me to work hard and finish my thesis in time. I would also like to thank my advisor, Dr. Getachew Alemu for all the reviews and comments he gave me on my work.

I have no space here to mention all the people whom I owe my gratitude, but there are some I couldn't afford to leave out. I want to say how much thankful I am to all Freescalers who make designing with their products like a charm with all the elaborate documentation, samples, and great customer support and all-rounded community forum. Finally I would like to thank my friends, Roger from Nokia, Mariano from Freescale and Gerald of Texas Instruments for their dedication and their generous experience sharing during the various stages of design, debugging and troubleshooting.

Abstract

In the last couple of decades, battery powered mobile devices such as smart phones have become one of the most prolific electronic devices in history. With this has come an exploding demand for performance and features that cover almost every aspect of our digital multimedia interconnected lives including 3-D gaming, still and video cameras, WAN, Bluetooth, high-speed data connections, and so on. As ever increasing features continue to be integrated into these products, there is an ongoing need to develop innovative ways to reduce power consumption and extend battery life.

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 is memory mapped and updated periodically by the main applications processor in the system. The basic notion of this architecture is achieving 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. In addition to this novel 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 XLP and all these three devices use the ARM microprocessor and run the Linux kernel. The comparison is on performance and power consumption. The XLP is proved to have the lowest power consumption on competitive performance levels.

About This Document

This document is a formal report of the thesis on Leakage aware hardware architecture and dynamic power scheduling. The theoretical backgrounds and the physical implementations are described. The results obtained are analyzed and conclusions and inferences are made as appropriate. For more technical details, please refer to the CD ROM attached to this document.

I. Definitions, Acronyms, and Abbreviations

The following table defines the acronyms and abbreviations used in this document

Term	Definition
AAC	Advanced Audio Coding – lossy compression and encoding scheme for digital audio designed to be a successor of mp3
ADC	Analog to digital converter
address translation	Address conversion from virtual domain to physical domain
AMBA®	Advanced Microcontroller Bus Architecture
AMOLED	Active matrix Organic Light Emitting Diode
API	Application Programming Interface
ARM®	Advanced RISC Machines processor architecture
BSP	Board support Package – software package written for a specific platform
bus	path between several devices through data lines
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CODEC	Coder/decoder or compression/decompression algorithm—used to encode and decode (or compress and decompress) various types of data
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit—generic term used to describe a processing core
CRC	Cyclic Redundancy Check—Bit error protection method for data communication
DDR	Dual Data Rate – a synchronous data transfer both on the rising and falling edge of the synchronizing clock signal
DMA	Direct Memory Access—an independent block that can initiate memory-to-memory data transfers
DPM	Dynamic Power Management
DRAM	Dynamic Random Access Memory
DVFS	Dynamic Voltage and Frequency Scaling
EMC	ElectroMagnetic Compatibility – a measure of the emission and immunity of a certain electronic device against standard values
EMI	External Memory Interface—controls all IC external memory accesses (read/write/erase/program) from all the masters in the system

EMI	Also used to mean ElectroMagnetic Interference – according to the context
Endian	Refers to byte ordering of data in memory. Little endian means that the least significant byte of the data is stored in a lower address than the most significant byte. In big endian, the order of the bytes is reversed
ERP	Energy Response-time Product – a metric used to capture the tradeoff between power consumption and performance
FIFO	First In First Out
Flash	A non-volatile storage device similar to EEPROM, where erasing can be done only in blocks or the entire chip
Flush	Procedure to reach cache coherency. Refers to removing a data line from cache.
GPIO	General Purpose Input/Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
HD	High Definition – high resolution video standard (HD 720p and full HD 1080p)
i.MX®	Intelligent Multimedia eXtension – Freescale's family of multimedia enhanced applications processors
I2C	Inter-Integrated Circuit – a two wire inter IC data communication standard
IPU	Image Processing Unit —supports video and graphics processing functions and provides an interface to video/still image sensors and displays
ISM	International Scientific and Medical band – a free frequency band with center frequency of 2.4GHz
ISR	Interrupt Service Routine
JTAG	Joint Test Access Group - (IEEE Standard 1149.1) A standard specifying how to control and monitor the pins of compliant devices on a printed circuit board
LCD	Liquid Crystal Display
LED	Light Emitting Diode
line	Refers to a unit of information in the cache that is associated with a tag
LTIB	Linux Target Image Builder
MMC	Multi Media Card
MMU	Memory Management Unit—a component responsible for memory protection and address translation
MPEG	Moving Picture Experts Group—an ISO committee that generates standards for digital video compression and audio. It is also the name of the algorithms used to compress moving pictures and video
MTD	Memory Technology Devices
NAND Flash	Flash ROM technology—NAND Flash architecture is one of two flash technologies used in mass storage devices.

PCB	Printed Circuit Board
PHY	Physical interface
PWM	Pulse Width Modulation
QVGA	Quarter Video Graphics Array
PLL	Phase Locked Loop—an electronic circuit controlling an oscillator so that it maintains a constant phase angle (a lock) on the frequency of an input, or reference, signal
RAM	Random Access Memory
RGB	The RGB color model is based on the additive model in which Red, Green, and Blue light are combined to create other colors. The abbreviation RGB comes from the three primary colors in additive light models
ROM	Read Only Memory
ROM bootstrap	Internal boot code encompassing the main boot flow as well as exception vectors
SAHARA®	Symmetric/Asymmetric Hashing and Random Accelerator
SCC	SeCurity Controller—a security hardware module
SD	Secure Digital (Smart Media) – Secure data storage card made of flash memory
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SMS	Short Message Service
SoC	System on a Chip
SPBA	Shared Peripheral Bus Arbiter—a three-to-one IP-Bus arbiter, with a resource-locking mechanism
SPI	Serial Peripheral Interface—a full-duplex synchronous serial interface for connecting low-/medium-bandwidth external devices using four wires.
TFT	Thin Film Transistor
UART	Universal Asynchronous Receiver/Transmitter—asynchronous serial communication to external devices
USB	Universal Serial Bus—an external bus standard that supports high speed data transfers.
USB OTG	USB On The Go—an extension of the USB 2.0 specification for connecting peripheral devices to each other by configuring as a host or device.
Wi-Fi	Wireless Fidelity – A short range, high bandwidth wireless data communication standard in the ISM band

II. Trademarks and Standards

Trade marks

- I.MX, LTIB and SAHARA are registered trademarks of Freescale Semiconductor Inc.
- ARM, ARM9, ARM11, ARM Cortex, Jazelle and AMBA are trademarks of ARM PLC.
- OMAP, Davinci, and SmartReflex are trademarks of Texas Instruments Inc.
- Core, Nehalem and turboboost are trademarks of Intel Corp.
- QT and Qtopia are trademarks of Trolltech/Nokia

Standards

- AAC is the default audio format for most modern devices, including XLP, it is standardized by ISO and IEC as part of the MPEG-2 and MPEG-4 specifications.
- Bluetooth is created by Ericsson, but standardized and maintained by Bluetooth Interest group (Bluetooth SIG)
- I2C is a two wire inter-IC data communication standard proprietary to Philips
- MPG, MPEG-2 and MPEG-4 are video data standards set by the Moving Picture Experts Group, an ISO committee.
- SPI is a four wire, medium bandwidth synchronous data communication standard proprietary to Motorola
- Wi-Fi is standardized and maintained by the Wi-Fi Alliance
- WMV and WMA are Windows video and audio formats respectively. These are Microsoft's video and audio encoding standards.

Table of contents

Chapter 4

Implementation

Chapter 5

Results

List of figures

List of Tables

Chapter 1
Introduction

Power management is one of the most important parts 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. With the rapid advances in integration level and functionality of System on Chip (SoC) for mobile battery operated systems one of the key challenges is the management and conservation of available power. 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. With the ever shrinking process technology *(32nm as of 2010 and 24nm in the near future [10])* static power loss will become comparable to the dynamic power loss.

Designers today are challenged not only by new process technologies capable of incorporating more and more devices on a single chip, but also managing the increase in the power that goes along with it. Techniques such as clock gating, low power processes, low power IP and lower supply voltage used with each new generation of process technology have helped designers of mobile applications to stem the tide of ever increasing power.[7], [8], [11] ,[15]

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 of 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 [17].

These 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 project 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 angry components in mobile systems are identified and a stochastic model of these components is developed to statistically and in real time control the power delivered to these modules. Generally, the power management policies are designed to reduce energy, not just 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.

The hardware optimization involves creating a system with independent modules and powering them through a matrix of power switches (gate matrix). Every module, therefore, 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 instead of disabling or forcing it to a 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, $S_i \in S$ 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. A proof of this theory is provided in chapter 3. This algorithm is implemented as a core power scheduling function in the modified Linux dynamic power management kernel.

The test and validation approach is to measure the power consumption of the device broken down to the device's major subsystems, under a wide range of realistic usage scenarios. Specifically, breakdown of power distribution to CPU, memory, display, graphics, audio, storage and networking is done. Then the results are compared and contrasted with three modern smart phones whose power consumption have been studied and published in the literature under similar benchmarks and usage scenarios. The devices used for comparison are Openmoko Neo freerunner, the HTC Dream and Google's Nexus One. All of these devices run the Linux kernel and use the ARM microprocessor as in the case of the XLP.

Chapter 2

Literature Review

A lot of extensive work has been done in power consumption reduction each employing different methods and algorithms, such as "dynamic voltage and frequency scaling" [7] [12], Intel's turbo boost technology [8], Arithmetic reduction of static power consumption [10] profile based systems [13] and etc. but few dare combine hardware and software solutions for both static and dynamic power reduction techniques to deliver comprehensive and adaptive power management schemes.

In [18], [19], and [16] the concept of critical speed and critical interval is introduced which presented the main motivation for the proposed research. The concept of *critical speed* (or *threshold speed*), denoted as S_{cri}, is adopted by previous work on leakage aware scheduling [18, 19, 17]. When processor is active (not idle), S_{cri} is defined as the available speed to execute a cycle 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}. But 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. Usually, the maximum speed is normalized to be 1.

Several efforts have been reported in literature for CPU-centric techniques for profile-driven power management [18] [19], while application aware strategies for energy centric partitioning and management will lead to efficient DPM solutions for dedicated systems, the tradeoff between performance and low power consumption becomes unmanageable for general purpose applications.

In [7], [8], [11] and [15] 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.

Aaron Carroll and Gernot Heiser perform a detailed analysis of the power consumption in modern Android smart phones [1]. They determined where significant amount of energy is consumed in typical smart phones. They made a direct power measurement on the physical device and graphically indicate the power breakdown showing the share of the major components in the device. They provide a detailed benchmark for the Neo Freerunner, the HTC dream and Google's Nexus One. Their bench marks are used for comparison in this paper.

Mahesri and Vardhan [4] perform an analysis of power consumption on a laptop system. They show that the CPU and display are the main consumers of energy for their class of system, and that other components contribute substantially only when they are used intensively. Their approach to component power measurement is driven partially by direct power measurement, but largely by deduction using modeling and off-line piece-wise analysis. Their results mirror Carroll and Heiser's observations as well as the results shown in this paper that RAM power is insignificant in real workloads.

Bircher and John in their first work [4] look at component power estimation using modeling techniques. They demonstrate an error of less than 9% on average across all tested subsystems, including memory, chipset, disk, CPU, and I/O.

In a later work, Bircher and John [5] measure the power consumption of the CPU, memory controller, RAM, I/O, video and disk subsystems under a number of workloads. Their results show that CPU and disk consume the majority of the power, with the RAM and video systems consuming very little. However, they also show that RAM power can indeed exceed CPU power for highly memory-bound workloads.

Sagahyroon [8] perform an analysis, Similar to Carroll and Heiser's, on a handheld PC. They show significant consumption in the display subsystems, particularly in backlight brightness. Unlike Carol and Heiser, however, the results suggest that the CPU, and its operating frequency, is important to overall power consumption. They also show significant dynamic power consumption in the graphics subsystems.

Chapter 3

Methodology

3.1. Background

Power Model

Usually, a processor has two modes: *dormant mode* and *active mode* [10]. If a processor is turned to dormant mode (idle mode), only very little energy would be consumed, denoted by P_{sleep} [13]. If a processor is in active mode, the active power consumption P_{act} can be divided into two parts [13]: *dynamic power consumption (P_{dyn})* and *static power consumption (P_{stat})*. Dynamic power consumption consists of the switching power for charging and discharging the load capacitance, and is given by [13]:

$$P_{dyn} = C_{eff} V_{dd}^2 \quad (3.1)$$

Where C_{eff} is the effective switching capacitance, V_{dd} is the supply voltage and f *is* the operating frequency. Static power consumption consists of the power consumed by the subthreshold 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 [13]:

$$P_{stat} = V_{dd} I_{sub} |V_{bs}| I_j \quad (3.2)$$

where V_{dd} is the supply voltage, I_{sub} is the subthreshold current, V_{bs} is the body bias voltage and I_j is the reverse bias junction current. 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 [18].

A typical battery powered system, such as a Smartphone, is composed of many processors and other supporting analog and digital integrated circuits. A typical Smartphone would have at least three processors, main Applications processor, baseband processor and multimedia/graphics processor. It also contains several smaller ICs, such as ADCs and DACs, transceivers, audio codecs, amplifiers, etc.

Each of these components is responsible for the static and dynamic power consumption of the system. In real usage scenarios, most of these components are idle or unused for most of the time.

3.2. Zero-leakage architecture

From the above discussion 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. According to eq. (3.2), 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, $P_{stat} = 0$.

For a Smartphone, which stays in the standby state for the majority of the time, the above approach will save significant energy. According to previous studies [1], [4] more than 60% of energy consumed in smart phones is due to static power consumption. The figure below shows a conceptual block diagram of the hardware design.

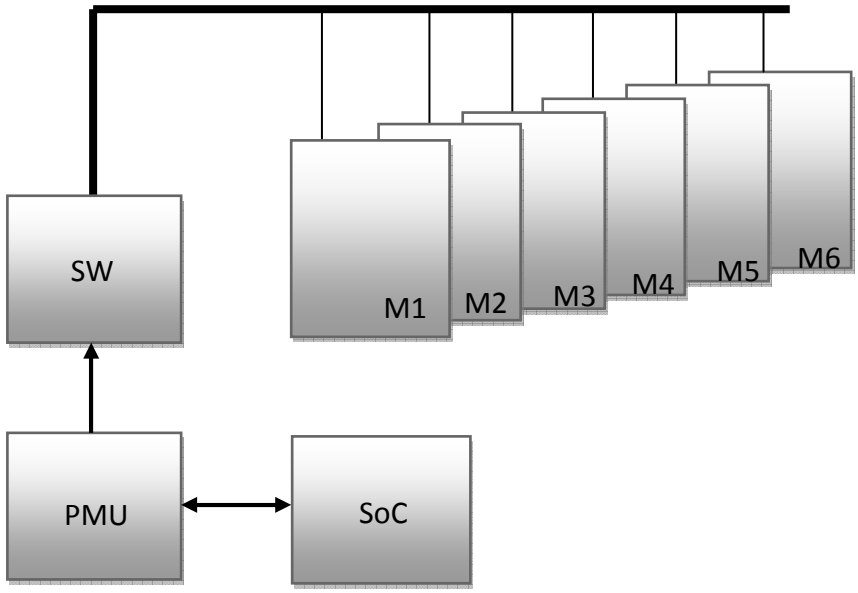


Fig 3.1 Block diagram of the system architecture

M1 to M6 are the six major modules that made up the system. Those are:

M1 => Cellular radio

M2 => Display module (LCD panel and backlight)

M3 => Wireless module (Wi-Fi and Bluetooth)

M4 => Audio module (codec and amplifier)

M5 => Graphics module

M6 => Navigation module (GPS, accelerometer and compass)

The modules are independently supplied through a set of active switches, labeled as SW in the figure. The power management unit (PMU) is responsible to control the active switches depending on the system activity. The system on chip (SoC) is built around the high performance microprocessor leveraging ARMv5 architecture and instruction set. The SoC used in this design is Freescale's i.MX233 which integrates the following main components: High performance low power ARM926EJ-S microprocessor core up to 456MHz, a graphics processing unit(GPU), 128bit AES decryption engine, DDR SDRAM controller, LCD controller, USB OTG PHY, 12-bit 16-channel analog to digital converter, audio codec, amplifier and power management. The ARM core inside the SoC supports core voltages from as low as 1V up to 1.55V simultaneously scaling the frequency from as low as 24MHz up to 456MHZ. the core voltage and the CPU frequency can be changed on the fly while the CPU is still up and running. The core also supports many operating states for minimizing power consumption. These modes are: RUN, IDLE, and STOP. At STOP mode the CPU clock is turned off and the core voltage is set at the lowest threshold, a state known as "wait for interrupt" which is entered when the ARM core is made to wait for an interrupt to occur to continue processing the next instruction. In this mode the core uses only 4.6mW. There is an active serial link between the SoC and the PMU such that the SoC can update the current power profile to the PMU. The SoC's power usage is in turn controlled by the PMU depending on the dynamic power scheduling algorithm and the application being executed.

3.3. System modeling

The approach is first to determine which part of the system has the longer active time and the higher power consumption in proportion to the other system components. Once this is identified, then the activity of this module can be modeled by using some mathematical/statistical equivalence depending on which an algorithm can be generated to optimize the power-performance tradeoff.

3.3.1. Background

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, text 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 communications processor along with the radio interface represents the cellular module.

In this work DPM is performed only on the active interaction between the main applications processor and the communications processor. Therefore a statistical model is developed to characterize the activity of the cellular module.

3.3.2. Goal and metrics

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 in this paper 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 control policy π, the ERP is given by:

$$ERP^{\pi} = E[P^{\pi}] \, . \, E[T^{\pi}] \tag{3.3}$$

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.

There are other performance metrics that also capture the tradeoff between response time and energy, for example, a weighted sum of the mean response time and mean power (ERWS). However, the ERWS metric implies that a reduction in mean response time from 1001 sec to 1000 sec is of the same value as a reduction from 2 sec to 1 sec. By contrast, the ERP implies that a reduction in mean response time from 2 sec to 1 sec is much better than a reduction from 1001 sec to 1000 sec, which is more realistic. One reason for the popularity of ERWS is that it is a nicer metric to handle analytically, being a single expectation, and hence additive over time. Therefore, one can optimize the ERWS metric via Markov Decision Processes, for example. From the point of view of worst case sample path based analysis, this metric allows comparing arbitrary policies to the optimal policy via potential function arguments. However, ERP, being a product of two expectations, does not allow a similar analysis. Other realistic metrics of interest include minimizing total energy given bounds on, say, the 95% of response times.

3.3.3. Policy definition

A specific set of management policies (defined in Table 3.1) is considered and prove that it contains the optimal policy for the activity between the two main processors, the applications processor and the communications processor.

The application running on the applications processor is the customer requesting a job to be done by the communications processor, like making calls, internet connectivity, etc.

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, S until there is no work to process, and begins to wake up as soon as work arrives.

Table 3.1 Summary of the policies considered in this paper, and their description.

To begin with, the communications modules are considered. The 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 the 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 ON to OFF. Therefore the passive modules: Audio module (codec and amplifier), display module (LCD panel and backlight) are managed by the INSTANTOFF policy.

3.3.4. Stochastic Analysis

Managing the job request to the network sub-section is very similar in flavor to the well studied problem in the stochastic analysis community: Inventory management. In inventory management, the problem of capacity provisioning takes the form: how much inventory should one maintain so as to minimize the total cost of unused inventory (holding cost, in our case idle power) and waiting cost experienced by orders when there is no inventory in stock (queuing delay of applications). Conceptually this problem is remarkably similar to the problem we consider, and the two common solution strategies employed, known as Make to Order and Make to Stock, are similar in flavor to what we call INSTANTOFF and NEVEROFF, respectively.

3.4. Process Model

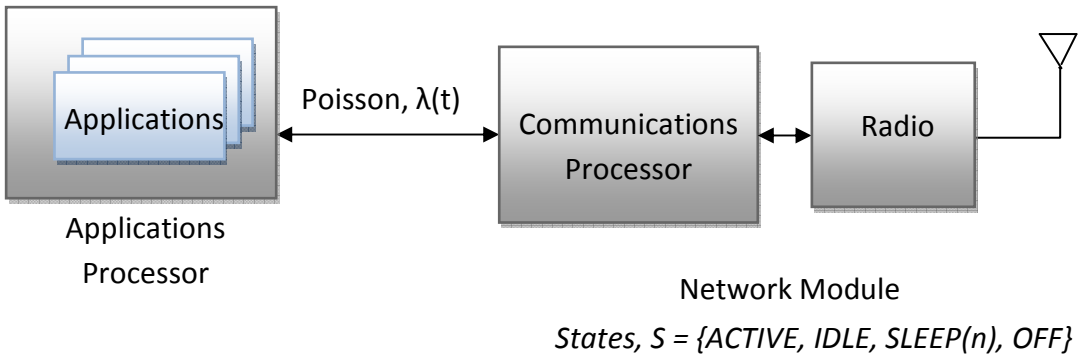


Fig 3.2. Stochastic Process Model

Jobs arrive from the application, to the communications processor, according to a Poisson process. A fixed arrival rate, λ is considered. However, a time-varying arrival rate, $\lambda(t)$ can also 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 $\rho(t) = \lambda(t) \cdot E[S]$ 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 $N - 1$ sleep states: $S_1, S_2, \ldots, S_{N-1}$. For convenience, the idle is sometimes referred to as S_0 and the off state as S_N. The associated power values are $P_{ACTIVE}, P_{IDLE}, P_{S0}, P_{S1}, \ldots, P_{SN} = P_{OFF}$. 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 $T_{si \rightarrow Sf}$ and it is a constant (not a random variable). Rather obviously, we assume $T_{ACTIVE \rightarrow IDLE} = T_{IDLE \rightarrow ACTIVE} = 0$. Further, the average power consumed while transitioning from state S_i to S_f is given by $P_{Si \rightarrow Sf}$.

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, $T_{ON \rightarrow OFF} = T_{Si \rightarrow OFF} = 0$, 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, $T_{OFF \rightarrow IDLE} = T_{OFF \rightarrow Si} = T_{OFF}$, 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 $P_{Si \rightarrow Sf} = P_{ACTIVE}$. The results of this paper are derived under the model assumptions.

3.5. Optimal Policies

The first step towards achieving the proposed goal is finding policies to effectively manage the performance and energy of the different modules in the system. It is pointed out in 3.3.3 that the policy INSTANTOFF is optimal for passive modules as their response time is not considerable.

However for the cellular modules the aim is to find the policy that minimizes ERP under a Poisson arrival process of known intensity. Theorem1 below states that the optimal policy is included in the set of NEVEROFF, INSTANTOFF, SLEEP(n) (as defined in table 3.1), and hence there is no need to consider any other capacity provisioning policy.

Theorem 1

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.

It should be noted that this is quite a non-intuitive result, and in general one should not expect it to hold for other metrics such as ERWS. The theorem rules out a large class of policies, for example those which may randomize between transitioning to different sleep states, or policies which move from one sleep state to another, or those which may wait for a few jobs to accumulate before transitioning to the ON state. While ERP, being a product of expectations, is a difficult metric to address analytically, for this case it is able to obtain tight optimality results by deriving explicit expressions for ERP.

Proof of Theorem 1:

A high-level sketch of the proof is given here in terms of five lemmas, whose proofs are referred to Appendix A. These lemmas successively narrow down the class of optimal policies, until we are left with only NEVEROFF, INSTANTOFF and SLEEP.

Definition 1

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.

Lemma 1
Under a Poisson arrival process and general *i.i.d*. job sizes, the optimal policy lies in the set Π_{mixed}.

Lemma 2

Consider a policy $\pi \in \Pi_{mixed}$ with parameters as in Definition 1. The mean response time for policy π under a Poisson (λ) arrival process with $Exp(\mu)$ job sizes is given by:

$$E[T] = \frac{\sum_{i=0}^{N} P_i \sum_{j=1}^{\infty} q_{ij} r_{ij}}{\sum_{i=0}^{N} P_i \sum_{j=1}^{\infty} q_{ij}\left(j + \lambda T_{S_i}\right)} \tag{3.4}$$

where,

$$r_{ij} = \frac{j + \lambda T_{S_i}}{\mu - \lambda} + \left[jT_{S_i} + \frac{j(j-1)}{2\lambda} + \frac{\lambda T_{S_i}^2}{2} \right] \tag{3.5}$$

and the average power for policy π is given by:

$$E[P] = \frac{\sum_{i=0}^{N} P_i \sum_{j=1}^{\infty} q_{ij}\left[j\left(\rho P_{ON} + (1-\rho) P_{S_i}\right) + \lambda T_{S_i} P_{ON} \right]}{\sum_{i=0}^{N} P_i \sum_{j=1}^{\infty} q_{ij}\left(j + \lambda T_{S_i}\right)} \tag{3.6}$$

Lemma 3

The optimal strategy for a single module must be pure. That is, $p_i = 1$ for some $i \in \{0,\ldots,N\}$ and $q_{in_i} = 1$ for some integer $n_i \geq 1$.

Lemma 4

The optimal pure strategy dictates that $n_i = 1$, if the optimal sleep state is S_i.

Lemma 1 is proved using a sample path argument and crucially depends on the Poisson arrival process and the model assumptions for the sleep states of the module.

Lemma 3 relies on the structure of ERP metric. While Lemma 3 also holds for the ERWS metric (with a much simpler proof), it does not necessarily hold for general metrics such as the product of the mean power and the square of the mean response time. Lemma 4 also relies on the structure of the ERP metric and does not hold for other metrics such as ERWS.

Lemma 5

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:

$$E[T] = \frac{1}{\mu - \lambda} + \frac{T_{S_i}\left(1 + \lambda T_{S_i}/2\right)}{1 + \lambda T_{S_i}} \tag{3.7}$$

$$E[P] = \frac{\rho P_{ON} + (1-\rho)P_{S_i} + \lambda T_{S_i} P_{ON}}{1 + \lambda T_{S_i}} \tag{3.8}$$

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.

Proof:

Follows by substituting $p_i = 1$ and $q_{i1} = 1$ in Lemma 2. The expressions in Lemma 5 allow us to determine regimes of load and mean job sizes for which each of NEVEROFF, INSTANTOFF and SLEEP policy is best with respect to ERP. Although not shown, it is found that NEVEROFF is typically superior to the other policies, unless the load is low and the mean job size is high, resulting in very long idle periods. In the latter case, INSTANTOFF or one of the SLEEP policies is superior, depending on the parameters of the sleep and off states. Eqs.(3.7) and (3.8) are also helpful for guiding embedded system architects towards designing useful sleep states by enabling the evaluation of ERP for each candidate sleep state.

3.6. Dynamic power management Policy

Dynamic power management (DPM) techniques are used to reduce power consumption when the processor is active and performing some tasks. Dynamic power management is tightly bound to optimize the two rather contradicting requirements – low energy and high performance. There is a lot of debate in the literature on the issue that DPM techniques don't actually save energy, but power. One argument is that a given task requires a certain amount of time at the given execution speed. The faster a task is executed (increased performance) the higher the power consumption will be, but the lesser the time it takes to finish. In other hand if the speed of execution is lower (compromising performance), the power consumption may be reduced, but it will take longer time to finish the given task. In either case, since energy is the product of power and time it will remain the same. So the argument is DPM doesn't save energy, even if it does the savings will not be worth going for.

In this research, however, it is shown that meaningful energy saving can be achieved if DPM is profile driven and implemented systematically on a modular design.

3.6.1. Profile Driven Dynamic power scheduling

As discussed above dynamic power management may not save energy if it's as simple as frequency and voltage scaling, but would yield significant energy savings if the profile and/or context of the running application(s) is provided dynamically.

Let's assume a single application running, let it be a web browser and call it X.

X needs for instance, the CPU, RAM, Display and graphics at start up, but may use other modules later on depending on the user's interaction.

Let's create a scenario, where the user want to browse the updates on YouTube, So first the device has to be connected to the internet for the browser to work. This can be done using the cell or Wi-Fi. Which to use depends on availability, data rate and power – this calls for some optimization given the user profile, the connection scenario, etc.

If extreme power saving is required the display can be turned OFF while connecting. This may be left to the user to decide, i.e. users can setup profiles and activate the one they want depending on the circumstances.

Till now we have four modules active, CPU and RAM, display, graphics and Wi-Fi/cell.

Now say the user opened a video on YouTube, the flash player application will be launched. Here additional modules will be enabled, specifically audio codec and amplifier. Also the CPU and graphics need to be clocked faster while forcing the network interface to a low power state in between subsequent buffering intervals.

The above scenario clearly shows that profile based DPM can save significant energy enabling only the required modules at the required time, with a little overhead to the processor. This also enforces a new trend in application development, since the application has to provide its changed state each time a new process is started.

3.6.2. On demand power gating

As discussed in chapter three, one of the things that make this work unique is the introduction of the concept of active power switch matrix to separately control the power flow to individual modules. This switch matrix is controlled by the power management unit (PMU). In the above usage scenario, the unused modules have their corresponding switch off in the matrix, ensuring they are switched off to achieve zero leakage power. If a module is requested by the running application, then the PMU will throw the corresponding switch ON and activate the module.

The biggest challenge with this method is the response time and the energy required to bring the said module from OFF state to the ACTIVE state. In some cases this approach might consume more energy than it saves. However, neither the PMU does a simple ON/OFF operation nor do the modules have binary states. The actual implementation takes the response time and the energy cost for a transition between states in to consideration.

In simple passive modules, such as the audio codec, amplifier, backlight,… there is no significant energy to switch them from OFF to ON or otherwise and the response time is negligible.

The main applications processor is not supplied through the switch matrix; it always stays powered unless the device is powered off deliberately. The processor, as mentioned in chapter three above, has multiple states. So, in which state should the processor be at a given time is determined from the profile of the currently running applications. STOP mode is activated only when the user presses the sleep key, and pressing the same key again will wake the processor and bring it up to IDLE state. Similarly, even if it is powered through the switch matrix, the cellular module is always powered unless the user switches off the phone. The cell can be switched off while keeping the device active to use it for multimedia, browsing or navigation.

The complete algorithm has two parts: Dynamic power scheduling algorithm and dynamic power and clock gating algorithm. The algorithm is given below.

3.6.3. Dynamic power management algorithms

The figure below shows a block diagram of the system organization

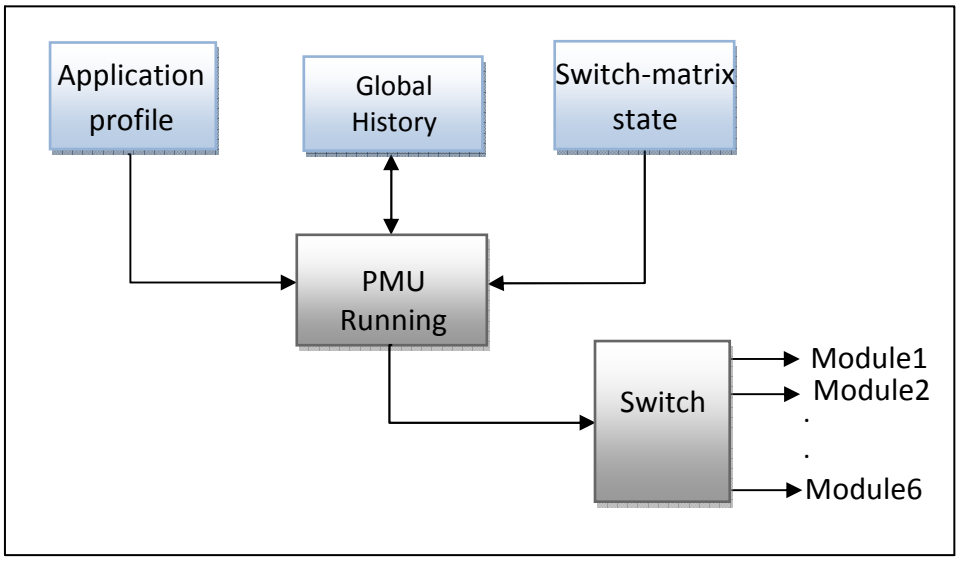


Fig 3.3 Elements of the dynamic power management scheme

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.

```
/* This loop runs as long as the app is running */
While(app_is_running)
{
  Get_profile(*app, *Pbuffer);  //get profile updates and
 //determine modules to be used
  refer_history(app_id, *Hbuffer, time_tag); //read history
 /*time_tag gives the option as recent, modified, or all.
   get power usage history (min, aver, max, std_dev) with
   corresponding duration*/
  Process::
   determine_modules_in_use {given *Pbuffer};
   get_performance_index {t, P};
   optimize_tradeoff {t,P,E};   //here using ERP as is theorem 1
   predict_module_usage_for_next_iteration;
  Update::
   update_switch_matrix_state;
   update_cpu_op_point {core_voltage, Core_freq, bus_freq};
  Save::
   Save_app_profile_for_next_iteration;
   Update_global_history {if_change_is_significant};
  /* wait for the update cycle to finish and continue */
 } //end loop
```

Chapter 4

Implementation

Implementation of these novel techniques and methods of power management requires a new device to be designed from the ground up. This new device is bound to the hardware architecture and should be able to run the dynamic power management algorithms discussed in the preceding chapters.

The XLP is a Smartphone that is designed to these specifications. The hardware is implemented such that the leakage power consumption is to the absolute minimum. This is achieved by using modular design approach and active switch power gates as discussed in details in chapter3.

To validate the functionality of the device, the standard Linux kernel (version 2.6.31) is ported to this custom platform. The kernel is edited, configured compiled and built using only open source tools (GNU collection): GNU GCC assembler, compiler, linker and also GNU C library. Various other packages are also built along with the kernel to enable all the power and performance tests to be performed: these include, drivers, test applications, low level utilities (especially CPU frequency scaling, terminal for data logging, etc.) and many more.

The test, measurement and benchmarking is done using a custom made measurement and data acquisition device. This device is designed with high precision measurement and data acquisition capabilities. This will ensure the accuracy of the measurements and benchmarks.

4.1 Hardware design

A high level block diagram of the hardware architecture and interconnect is shown in fig 4.1. The diagram shows how the different components are interconnected and how the flow of power and control looks like inside the XLP.

The major components of XLP and their specifications are given in Table 4.1.

Component	Specification
SoC	Freescale i.MX233
CPU	ARM926EJ-S 455MHz
RAM	128MiB DDR SDRAM
Flash	1GiB SLC NAND
Cellular modem	Telit GE864 quad band GSM+GPRS
GPS	Telit GE864-GPS
Accelerometer	ST LIS302, 3-axis
FM radio	Silicon Labs Si4701, FM receiver w/RDS
Graphics	Freescale Embedded
Display	Newhaven 3.5" 16M color QVGA (touch screen)
SD card	SanDisk 2GB
Bluetooth	Roving Networks RN-42
Wi-Fi	TI WiLink (not populated)
Audio codec	Freescale embedded
Audio amplifier	TI TPA6211
PMU	Freescale embedded: 3 DC/DC, 4 LDO
Battery	1200mAH 3.7V Li-ion
Power switches	Si2305 5V, 8A P-MOSFET, R_{on} = 0.075Ω

Table 4.1. Major components of XLP

4.1.1 CAE tools

The design of the XLP requires a series of high-tech design, analysis and simulation software to facilitate the Computer aided Engineering (CAE). There are two major elements in CAE; Computer aided design (CAD) and computer aided manufacturing (CAM). The CAD addresses issues such as, Schematic design; PCB placement, layout, and routing; Electromagnetic interference and compatibility (EMI & EMC) analysis; Mechanical form factor and thermal constraints.

In this work the Cadence OrCAD V16.3 CAE software package is used to design the electrical schematic, PCB layout, routing and simulation. The distortion and interference in high speed digital interconnections is analyzed by using the IBIS model for the SoC and Using the Cadence Signal explorer. Schematic design is done using the Cadence OrCAD Schematic capture. Capture is globally known to be one of the best schematic design tools for the simple intuitive user interface, huge component library and very advanced functionalities. The PCB layout, routing and simulation are done by using the Allegro Expert. The design files are included in a CDROM submitted with this report.

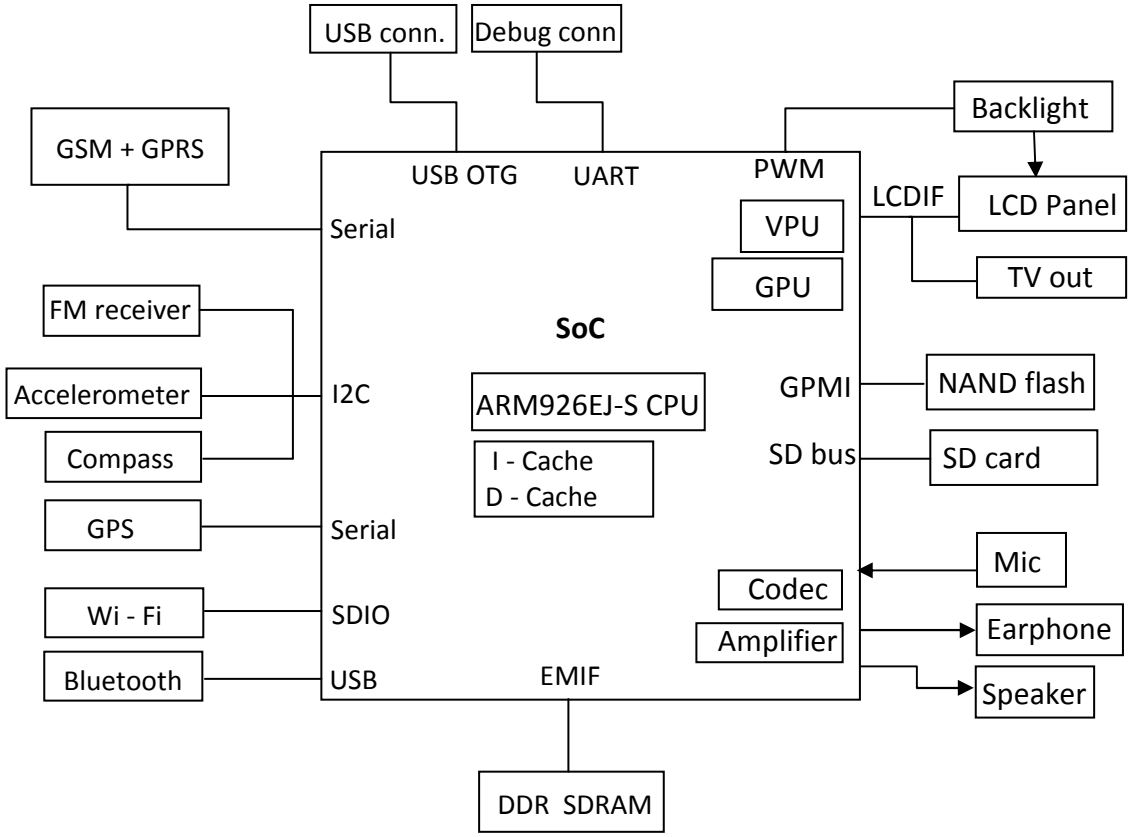


Fig. 4.1 Architecture of XLP, showing the important components and their interconnection

Note: The figure above shows only the functional block diagram, the power supply and management subsystem is not show.

4.1.2. Electrical Design

The electrical design has the following main constraints:

1. Routing dual data rate SDRAM high speed digital signals (data, address, clock and control)

2. Designing microstrip antenna for four different frequency bands. For the XLP, the following implementation was used:

   i) GSM/GPRS and GPS ---- surface microstrip, trace width 0.5mm, length = λ/4

   ii) Wi-Fi/Bluetooth – embedded microstrip, trace width 0.35mm, length = $\lambda/2$

   iii) FM radio ---- headphone cable + passive matching

The trace widths *(w)* are calculated according to the standard formula,

For surface microstrip: [33]

$$w = 7.475h\ e^{\left(-Z_0\sqrt{\varepsilon_r+1.41}\right)K} - 1.25t \qquad (4.1)$$

Restrictions: $0.1 < w/h < 3; \quad 1 < \varepsilon_r < 15$

where, $K = 87 \text{ for } 15 < w < 25; \quad k = 79 \text{ for } 5 < w < 15$

Z_0 *is* the characteristic impedance*;* ε_r is the relative permittivity of the PCB substrate and h is the height of the strip line above a ground plane.

For embedded microstrip:[33]

$$w = 7.475h\ e^{-x} - 1.25t \qquad (4.2)$$

Restrictions: $0.1 < w/h < 3; \ 1 < \varepsilon_r < 15$

where,

$$x = \frac{-Z_0\sqrt{\varepsilon_r + 1.41}}{87(1 - 10h)}$$

The PCB is completed in six layers with layer1(top) for placing components and signal routes; layer2(inner) complete ground plane; layer3(inner) split power plane; layer4(inner) signal routes; layer5 (inner) complete ground plane) and layer6 (bottom) some components and signal routes.

Figure 4.2 shows the PCB stack-up. Figures 4.3 and 4.4 show a snapshot of the microstrip antennae, figure 4.5 shows the DDR SDRAM interface routing. The complete PCB design is included in the CDROM including the PCB in OrCAD, pdf and Gerber formats.

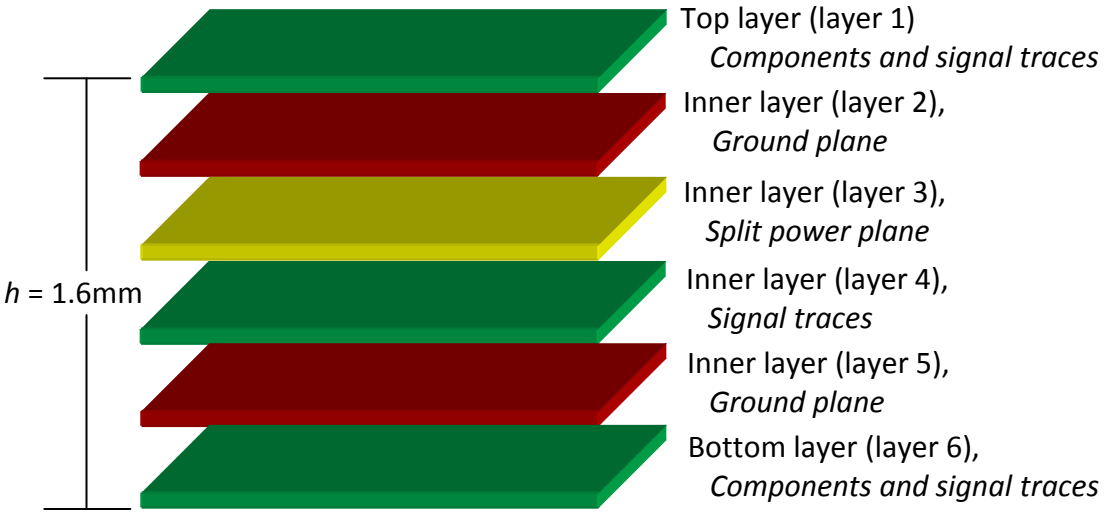


Fig. 4.2 PCB layer stack-up and assignment of each layer

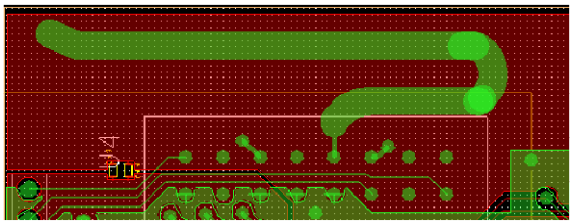


Fig. 4.3 (a) snapshot of the GSM/GPRS microstrip antenna

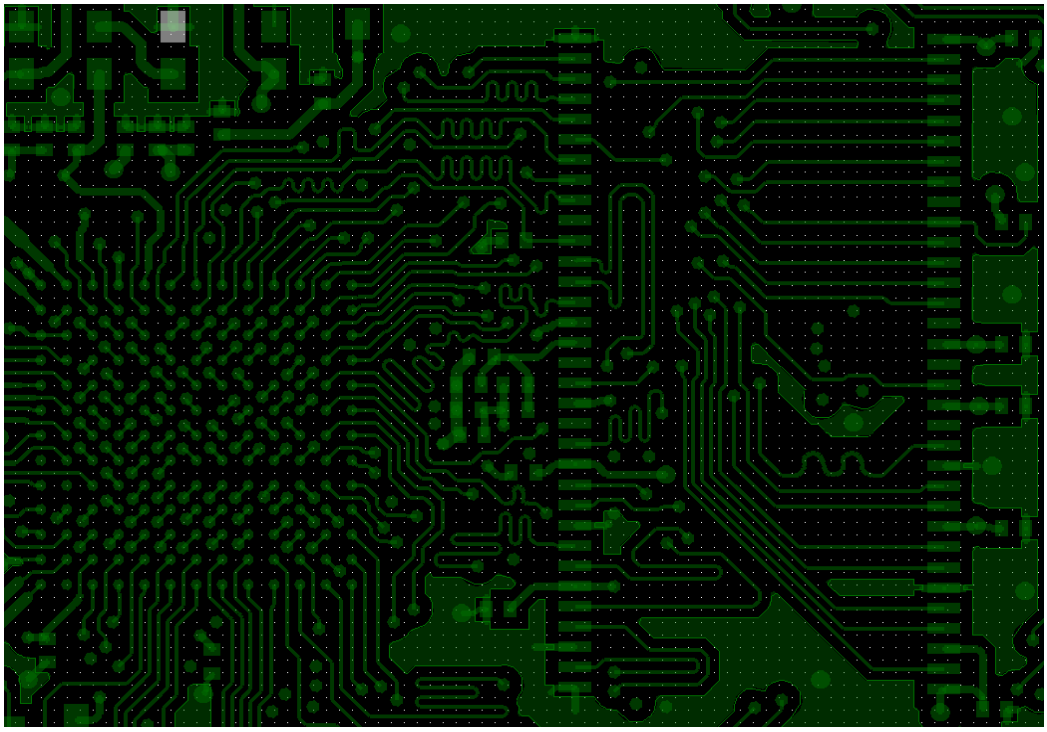


Fig. 4.4 (a) DDR SDRAM interface routing (top Side)

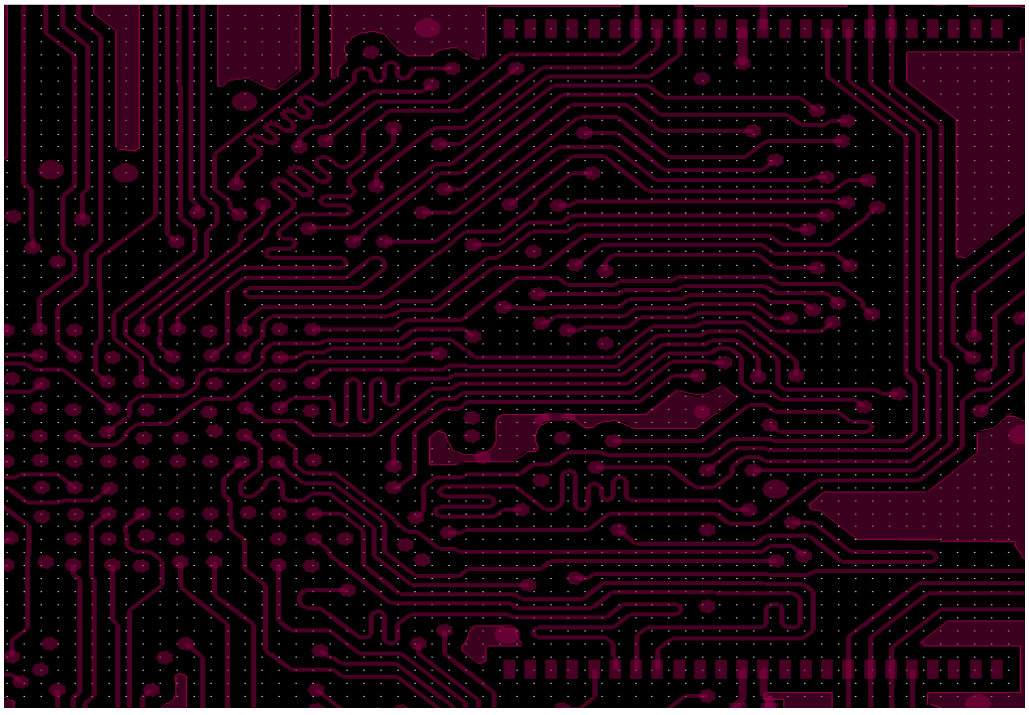


Fig. 4.4 (b) DDR SDRAM interface routing (bottom side)

4.1.3. Mechanical and thermal design

The following are the major constraints:

1. All the components and routing has to be accommodated in a board that measures (120mm long, 68mm wide and 2mm thick) to keep the mobile form factor.
2. The core PCB substrate should have a good thermal resistance and to support all operating frequencies and dielectric constant to give the required characteristics impedances.

The PCB substrate material in this design is the standard FR-4 laminate, with relative dielectric permeability, $\varepsilon = 4.5$. The surface copper is 1oz for both internal and external layers. Fig 4.5 shows a photo of the completed device.

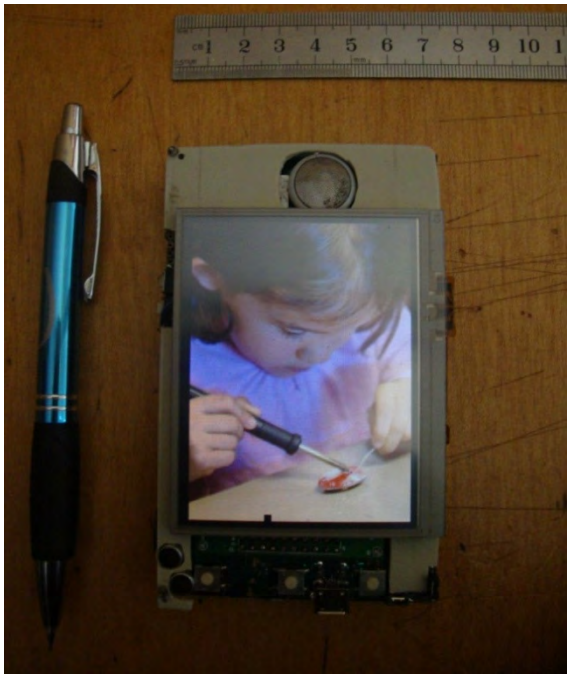


Fig. 4.5 (a) photo of XLP (front)

Fig. 4.5 (b) photo of XLP (Back)

4.2. Firmware and Software

The XLP hardware requires different firmware and software at different levels to be usable as a standalone mobile device. The Linux BSP is a collection of binary, code, and support files that can be used to create a Linux kernel image and a root file system for the XLP.

Fig 4.8 below shows a high level structure of the Linux BSP ported to the XLP platform. It consists of user space executables, standard kernel components that come from the Linux community, and hardware-specific drivers and functions provided by Freescale for the i.MX processors and device specific codes written by the author for the XLP.

4.2.1. Low level utilities

Boot stream

When the CPU comes out of reset, it begins executing from the on chip ROM. There is no alternative – no other code is permitted to handle the reset exception. The ROM reads the boot mode pins to discover the boot source (USB, SD/MMC, NAND Flash, etc.) and negotiates with that source in a device-dependent way to retrieve a "boot stream". A boot stream is a stream of bytes in Safe Boot (SB) format.

This boot stream starts with a "Load" command that instructs the ROM to copy the executable into memory. The final "Jump" command instructs the ROM to transfer control to the executable that it just loaded.

Another very important command is "Call." This command tells the ROM to make a function call to a given address and then continue processing the boot stream when control returns. A "Call" command is usually preceded by a "Load" command that copies into memory the function to be called. Collectively, the "Load" command, the associated executable and the "Call" command are referred to as a "bootlet".

Fig 4.6 is a schematic representation of a boot stream that contains two bootlets, followed by the main executable:

LOAD	Bootlet Executable #1	CALL	LOAD	Bootlet Executable #2	CALL	LOAD	Main Executable	JUMP

Fig 4.6 Boot Stream outline

Each bootlet is an executable that has been built separately, for a specific purpose, and may or may not know anything about the bootlets that precede or follow it.

The boot stream can instruct the ROM to "Call" any number of executables before the final "Jump," depending on the needs of the system. The XLP Linux BSP boot streams contain the following bootlets:

power_prep —bootlet to configure up the power supply for the different components of XLP.

boot_prep — This bootlet configures up the clocks, SDRAM and setup the bus speed.

Linux_prep — This bootlet prepares to boot Linux on to the RAM.

Here's a schematic representation of a boot stream constructed with the XLP Linux BSP:

LOAD	power_prep	CALL	LOAD	boot_prep	CALL	LOAD	Linux_prep	CALL	LOAD	zImage	JUMP

Fig 4.7 an example XLP Boot Stream loading the Linux Kernel

4.2.1 Kernel

The XLP Linux port is based on the standard Linux kernel. The kernel supports most of the features available in many modern embedded OSs such as:

• Process and thread management

• Memory management (memory mapping, allocation/deallocation, MMU, and cache control)

• Resource management (interrupts)

• Power management

• File systems (VFS, cramfs, ext2, ramfs, NFS, devfs, JFFS2, FAT, UBIFS)

• Linux Device Driver model

• Standardized APIs

• Networking stacks

ARM Linux Kernel customization to support any given platform includes a custom kernel configuration and machine specific layer (MSL) implementation. The MSL provides a machine-dependent implementation as required by the Linux kernel, such as memory map, interrupt, and timer. The complete MSL code can be referred to the attached CD-ROM at the following location.

<CD-ROM>/BUILD/linux/arch/arm/mach-XLP

4.2.2. Device drivers

The XLP has several different components that require a specific driver in order to communicate with the operating system. Many of these drivers are provided as standard components of the Linux kernel and are common across all platforms. But the specific components on the XLP require specific configuration, timing, voltage levels, etc. as specified by the manufacturers in the datasheet. Therefore, from as easy as reconfiguration to as hasty as writing from scratch is required to properly interface and use the different components of the device.

There are many drivers provided by Freescale that are specific to the peripherals on the i.MX family of processors.

These drivers are used for the on-chip peripherals and for those devices on-board, but off-chip, such as LCD panel, audio amplifier, baseband, PMU, etc. specific drivers are written and integrated to the kernel. Also, rather obviously, the power management part of the kernel is entirely re-written to use the DPM algorithm discussed in the preceding chapter. Most of the drivers can be compiled into the kernel or compiled as object modules which can be dynamically loaded from a file system through *insmod* or *modprobe* commands. Modules can be loaded automatically as required using the kernel auto-load feature. The BSP contains a `modules.dep` file and a `modprobe.conf` file that contain the dependency information for the modules.

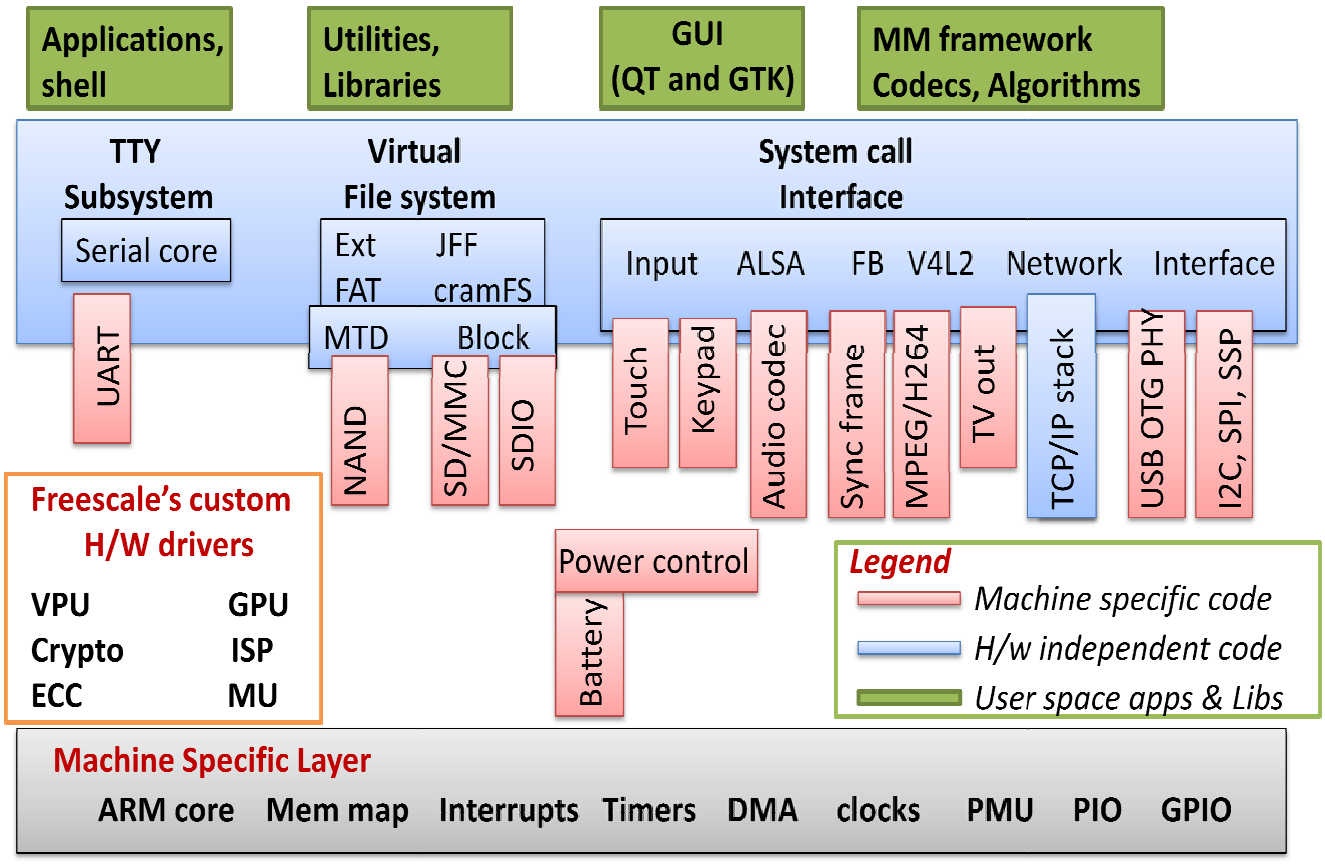


Fig 4.8 XLP's Linux BSP block diagram

These are the specific devices whose drivers are exhaustively edited and modified, some of them are even written from scratch.

• Enhanced power and battery management driver

• 24bpp parallel RGB display driver with frame buffer and backlight control

• Flash memory driver, for MTD, MMC, SD, and SDIO

• Wireless access driver, Wi-Fi and Bluetooth

• FM radio module driver

• Navigation and orientation sensors driver

• Cellular radio/baseband driver.

The complete driver code included in the attached CD-ROM at the following location.

<CD-ROM>/BUILD/linux/drivers

4.2.3. Kernel configuration

First of all, the host computer is setup with Ubuntu 9.04 (Jaunty Jackalope) which is an older version, but LTIB is known to be very stable on this version of Ubuntu. Once the host computer is running Ubuntu, the LTIB is installed and the Linux BSP for the XLP is configured as required.

Kernel configuration is done through the Linux Target Image Builder (LTIB), an open source tool provided by the LGPL license [39]. The configuration settings available on XLP that are different from the standard features are as follows:

• Embedded mode

• ARM9 support

• Supported file formats: ELF binaries, a.out, and ECOFF

• Block devices: Loopback, Ramdisk

• File systems: ext2, dev, proc, sysfs, cramfs, ramfs, JFFS2, FAT, pramfs

• Frame buffer

• Kernel debugging

• Power management

• Memory Technology Device (MTD) support

• USB Host/device multiplexing

• Unsorted block images (UBI) support

• Flash translation layer (FTL)

• CPU frequency scaling

The complete kernel source code is included in the attached CD-ROM at the following location.

<CD-ROM>/BUILD/linux

4.2.4. Test programs

After all the kernel configuration, driver development and deployment, the system has to be tested with the proposed power management philosophies. For this reason two test applications were developed to run on the kernel.

The first application performs power and frequency measurements and transmits the results to a host computer through the serial link. The host computer runs a measurement, analysis and graphing application. The application, periodically and in real time, measures and transmits the following parameters for a given test duration.

CPU core voltage and frequency, system bus frequencies, chip die temperature, CPU loading (%), current running processes and memory usage.

The program running on the host computer performs the following tasks:

i. Takes precise measurement of battery voltage, total current from battery, power in to the cellular module, wireless module, display module, backlight, audio module, and graphics module.
ii. receives real time data from XLP, make some analysis and processing
iii. display the result in the desired form (numeric, tabular or graphical)

The second application launches preconfigured usage scenarios on the user prompt. The usage scenarios configured are: minimal usage (80% idle), media player mode (video and audio), business mode (repeated voice calls, SMS, browsing), and gamming (excessive graphics rendering). The four scenarios are separately run and the power usages are recorded.

4.3 Measurement setup

To calculate the power consumed by any component, both the supply voltage and current must be determined. To measure current, sense resistors are inserted on the power supply rails of the relevant components. With a known resistance and measured voltage drop, current can be determined by Ohm's law. To measure the voltages, a precise measurement and acquisition device was made to which the sense resistors were connected via twisted-pair wiring. The key characteristics of measurement and acquisition hardware are summarized in Table 4.2.

Characteristics	value
Max sampling rate	500KS/s
Input range	±0.2V, ±1V, ±5V, ±10V
Resolution	16 bits
Accuracy	112μV @ ±0.2V; 1.6mV @ ±5V range
Sensitivity	5.2μV @ ±0.2; 48.8μV @±5V range
Input impedance	1GΩ

Table 4.2 Measurement system specification

4.4 Benchmarks

Two types of benchmarks are performed. First, a series of micro-benchmarks designed to independently characterize components of the system, particularly their peak and idle power consumption. Second, a series of macro-benchmarks based on real usage scenarios.

For low-interactivity applications (e.g. music playback), it is simply launched from the command line. For interactive applications, such as web browsing, a trace-based approach is used. A trace consisted of a sequence of input events, including a time-stamp, the name of the device providing the input (the touch screen or one of three pushbuttons), and for touch screen events, the coordinates of the touch. The Linux kernel provides this information by reading from the /dev/input/event device files. To collect the trace, the target application is used normally, while in the background storing the input events to file. Then, replayed the events under benchmarking conditions by writing the collected data to the /dev/input/event files at the correct time. Although this approach does bypass the hardware and interrupt paths that would usually be followed for a touch screen event, the measurements showed the additional power to be negligible. The vast majority of energy required to handle a touch screen event is consumed in delivering it from the kernel to software.

Chapter 5
Results

5.1. Base line cases

Prior to running any benchmarks, 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.

5.1.1. 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. Figure 5.1 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.

5.1.2. Idle mode

The device is in the idle state if it is fully awake (not suspended) but no applications are active. 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. Figure 5.2 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%.

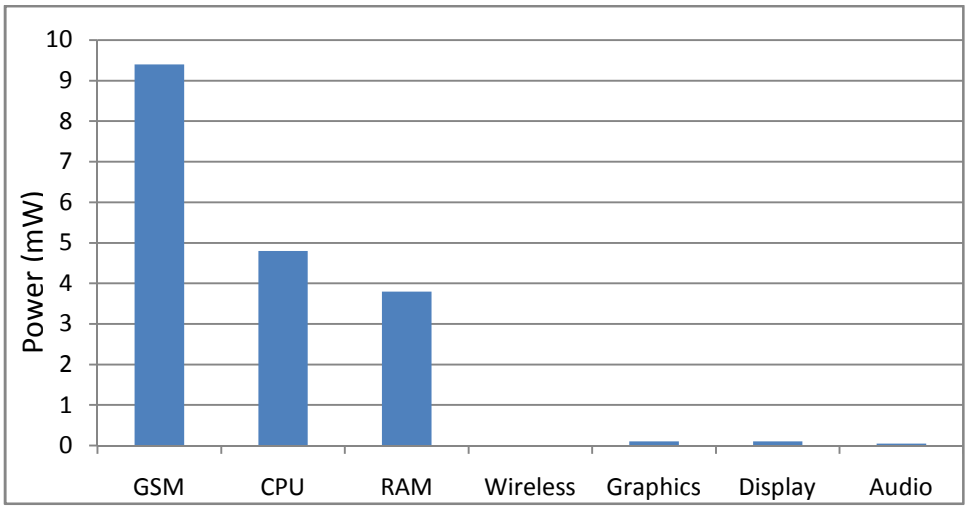


Fig 5.1 Suspended mode power breakdown. The aggregate power consumed is 18.3mW

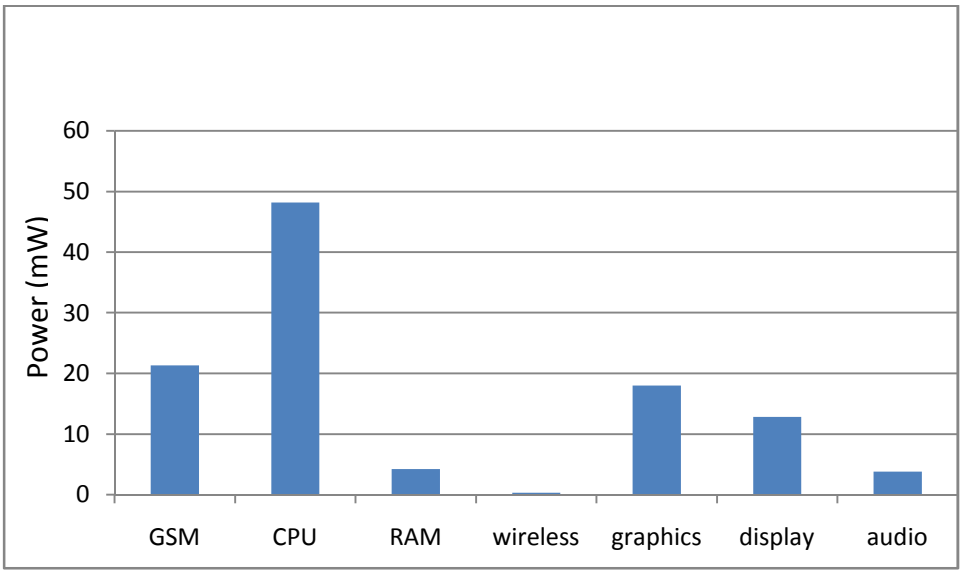


Fig 5.2 Idle mode power breakdown with backlight off. Aggregate power is 108.6mW

5.1.3. 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.

Figure5.3 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 Linux kernel provides a user-space utility that allows linear control of this value between 0 and 100. The minimum backlight power is approximately 0mW- at brightness level of 0 (off), the maximum 288mW, and the default corresponds to a brightness level of 50, consuming 48.2mW.

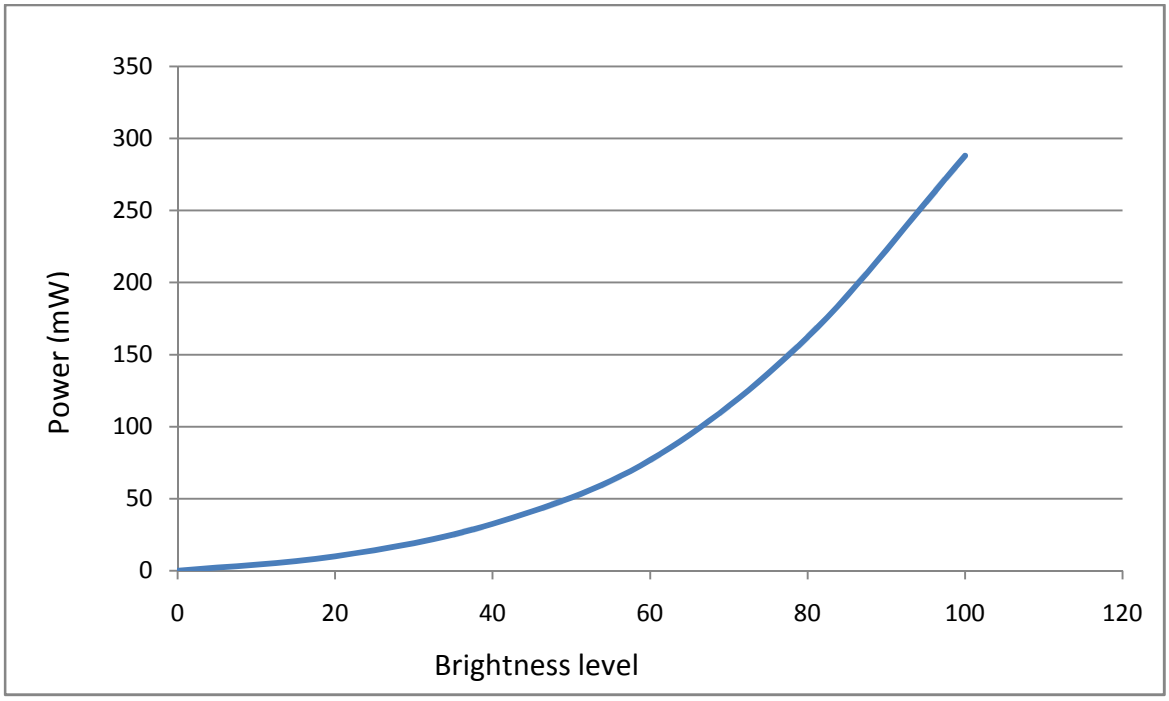


Fig 5.3 brightness level and corresponding power consumption of the backlight

5.2. Micro benchmarks

As mentioned above, 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

5.2.1. 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 5.1.

CPU frequency (MHz)	Core voltage (V)	EMI bus speed (MHz)
24	1.00	24
64	1.05	64
261.8	1.275	100
360	1.375	120
392.7	1.475	131
454.7	1.55	133

Table 5.1 voltage and frequency scaling values

5.2.2. Flash storage

Storage on the XLP device 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, 8 MiB of random data was written, with an fsync between successive 4 KiB blocks to ensure predictability of writes. Between each iteration, the page cache is flushed. Figure 5.4 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 5.2 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.

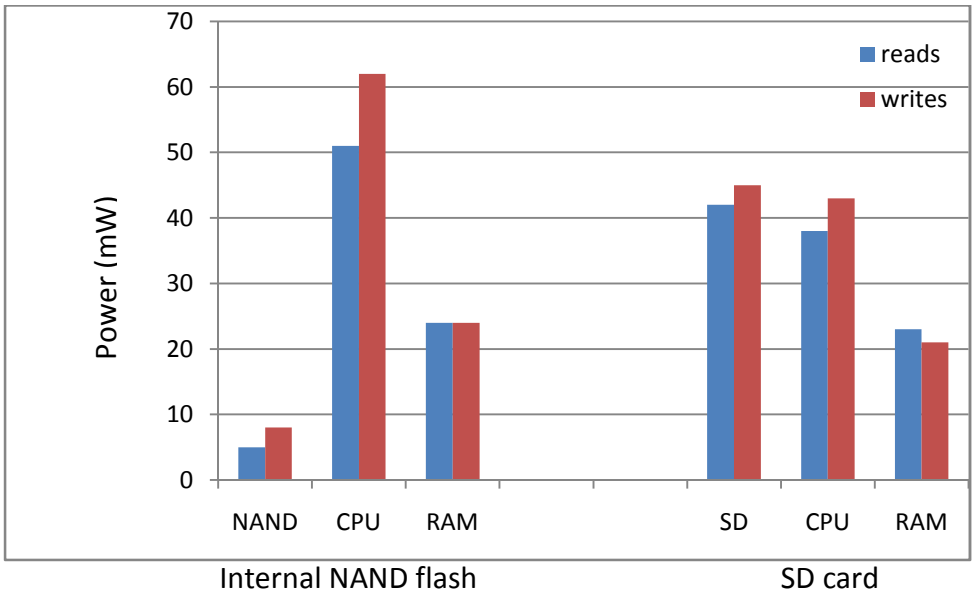


Fig 5.4. Power usage for flash read/write

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

Table 5.2 Flash storage power and performance

5.2.3. Network

In this benchmark 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 Figure 5.5. 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 %.

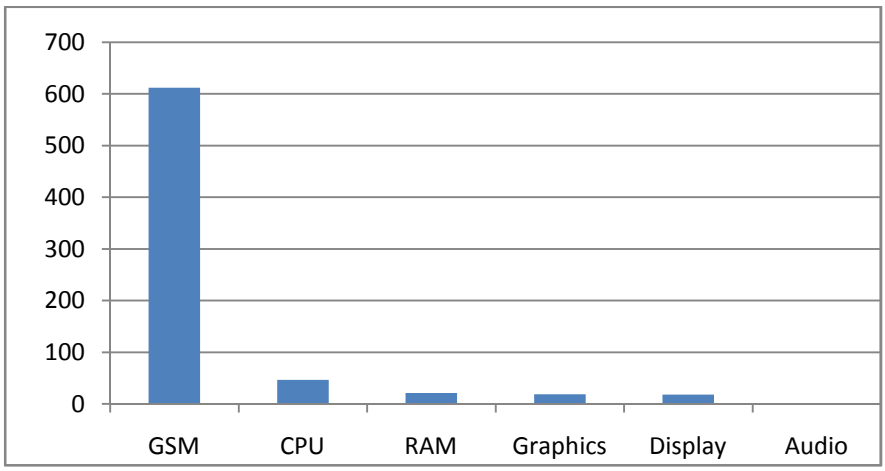


Figure 5.5 Network power breakdown. Aggregate power consumed is 715.1mW

5.3. Usage scenarios

The macro-benchmarks 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.

5.3.1. Audio playback

The system is being used as a portable media player in this benchmark. 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. Figure5.6. shows the power breakdown for this benchmark at maximum volume for 10 iterations.

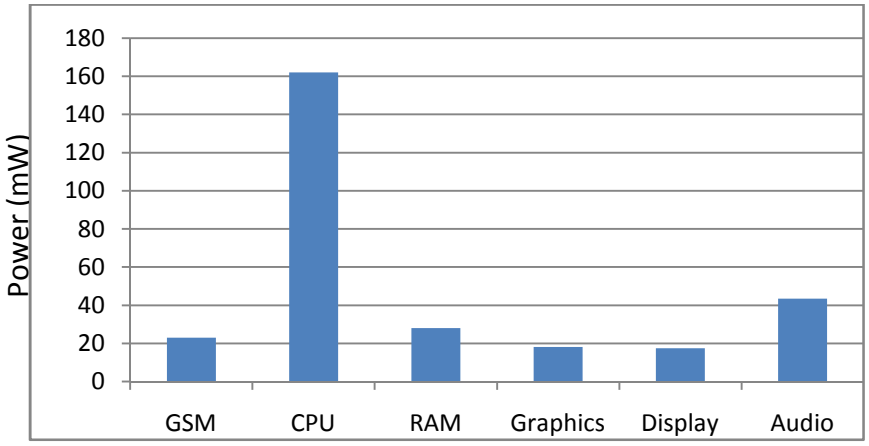


Figure 5.6 Audio playback power breakdown. Aggregate power consumed is 294.0mW

5.3.2 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 Figure 5.7. The backlight power is included in the results with for 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.

5.3.3. Voice call

The power consumption of the device when making a GSM phone call is shown in fig6.8. 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.

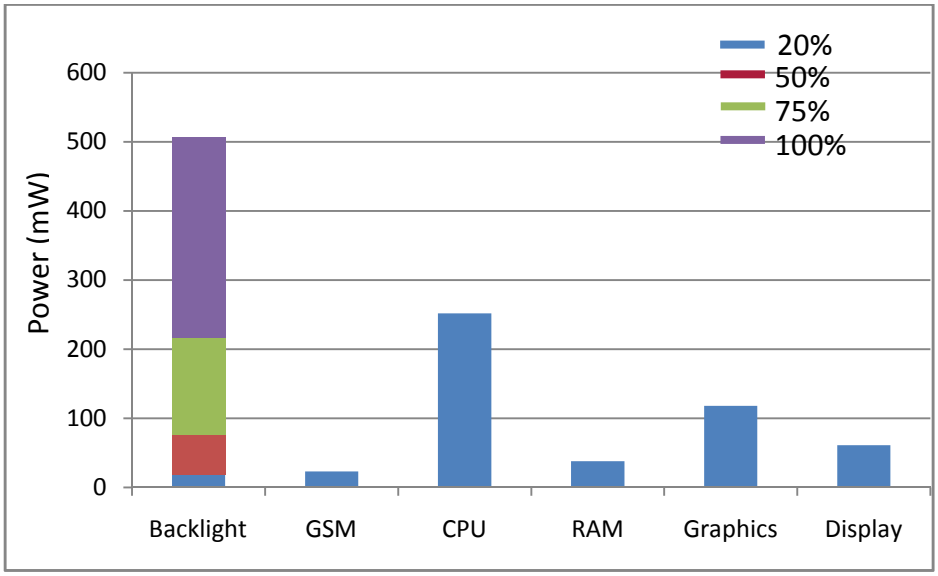


Fig 5.7. Video playback power breakdown. Aggregate power w/o backlight is 492mW

5.3.4. 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 Figure5.9. 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).

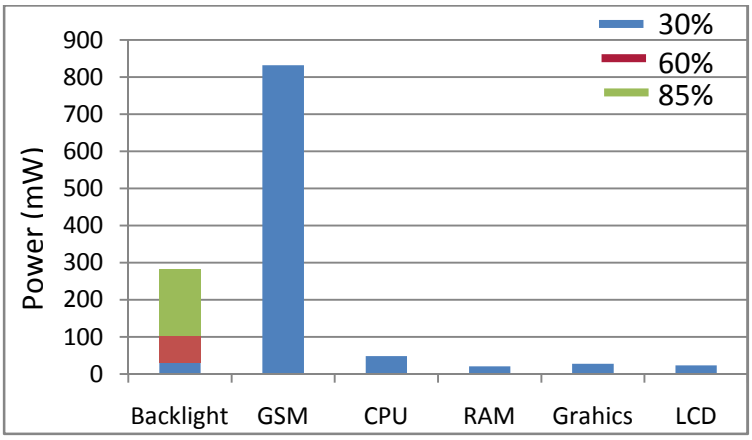


Fig 5.8 Power breakdown during voice call. Aggregate power consumed is 952mW

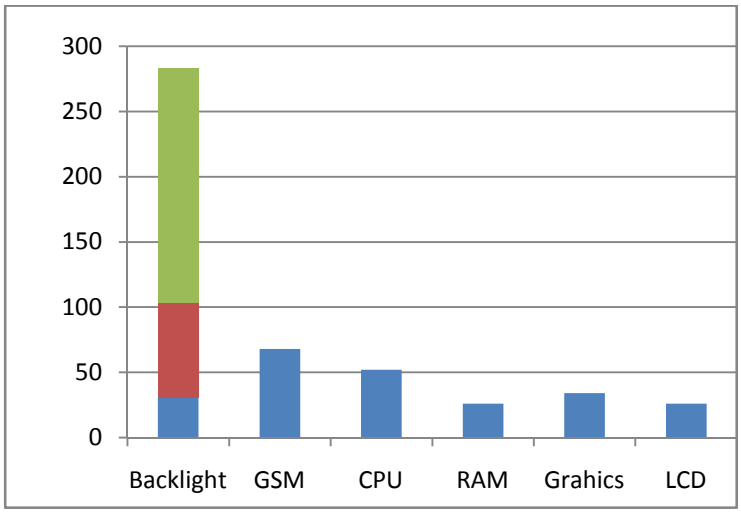


Fig 5.9 Power breakdown for using SMS. Aggregate power consumed is 206mW

Chapter 6
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.

Aaron Carroll and Gernoth Heiser [1] thoroughly studied and published detailed power consumption breakdown of three very popular Android smart phones; the openMoko Freerunner, the HTC dream (G1), and Google's Nexus one (N1). The former is a 2.5G Smartphone featuring a large, high-resolution touch screen display, and many of the peripherals typical of modern devices. The important thing about this device is it is its hardware and software sources are freely available such that detailed power measurements are possible. For the later two devices, Carroll and Heiser measure only the full-system power of these platforms at the battery. Per-component measurements are not possible because the necessary documentation (schematics, etc.) are not available. Moreover, there is no reason to expect these production devices would be capable of the type of instrumentation that can be performed on the Freerunner, since the additional components and PCB area would increase the per-unit cost. The only feature the freerunner and the XLP are missing compared to G1 and N1 is a 3G modem and camera. The comparisons, therefore, doesn't cover this features. Table 6.1 summarizes the features of these devices with the XLP alongside.

6.1. Comparison validations

Like the cases where the power consumption of the XLP was probed in the preceding chapter, the three devices under test are also subject to the same scenarios and their power usage is recorded [1].

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.

Component	**Freerunner**	**HTC dream (G1)**	**Nexus one (N1)**	***XLP***
SoC	Samsung S3C2442	Qualcomm MSM7201	Qualcomm QSD8250	Freescale i.MX233
CPU	ARM926T 400MHz	ARM11 528MHz	ARM Cortex_A8 1GHz	ARM926EJ-S 455MHz
RAM	128MiB	192MiB	512MiB	128MiB
Display	3.5” TFT LCD 480x640	3.2” TFT LCD 320x480	3.7” OLED 480x800	3.5” TFT LCD 240x320
Radio	GSM + GPRS	UMTS + HSPA	UMTS + HSPA	GSM + GPRS
Kernel	Linux 2.6.29	Linux 2.6.29	Linux 2.6.29	Linux 2.6.31
OS	Android 1.6	Android 1.6	Android 2.1	Custom*

Table 6.1 Summary of the specification of the devices under comparison

6.1.1. Display and backlight

In addition to the LCD display backlight, the G1 features a backlit physical keyboard and buttons which are not present on either the Freerunner or the N1. These backlights do not have any brightness control, and contribute 189mW when both enabled. The Nexus One features an OLED display, and as such does not require a separate backlight like the others. Furthermore, the effects of display content and brightness on power consumption are more tightly coupled.

For instance, the OLED power consumption for a black screen is fixed, regardless of the brightness setting. For a completely white screen at minimum brightness, an additional 194mW is consumed, and at maximum brightness, 1313mW!

6.1.2. CPU and RAM

Carroll and Heisel published the CPU power consumption and performance, refer to [1] for different micro benchmarks, different CPU frequencies and other test conditions. The N1 tend to consume more power, but finishes the job faster due to the 1GHz processor. The G1 is almost half slower than G1 and consumes, like 30% less power. Generally the N1 is more efficient when it comes to energy; the additional performance outweighs the increased power requirement.

6.1.3. Network

The cellular network along with the web browsing benchmark, are the only two where a more modern phone can be expected to show significantly different results. Voice call and text messaging is used as a comparison metric in this benchmark, since the N1 and G1 have a 3G modem, but the freerunner and XLP have 2.5, thus other scenarios like, web browsing using the mobile network are not considered valid for comparison. The much higher bandwidth supported by 3G protocols is likely to result in them being more power-hungry.

6.2. Benchmarks

The total system power consumption for the Freerunner, G1, Nexus One and XLP is shown in table 7.2 for a selection of benchmarks. The power consumption of the backlight (OLED for the N1) has been subtracted out, since it is highly dependent on the user's brightness setting. Table 6.3 shows the additional power consumption of the OLED display at minimum and maximum brightness levels. The lower power consumption of the G1 in the idle and web benchmarks can be attributed to the excellent low-power state of its SoC and effective use of it by software.

The power disparity for the phone call benchmark is likely due to power consumed by the non-radio components of the system. The G1 and N1 phones enter a suspended state during the call, offloading all functionality to the UMTS module. In contrast, the Freerunner remains in a fully-active state throughout. The power consumption of the GSM subsystem alone (832.4mW) is comparable to the G1 and N1 system consumption. Due to lack of freely-available documentation, it is not clear whether the Freerunner's GSM chipset lacks this feature, or if it is not supported in software. [1]

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	

Table 6.2 Average power consumption comparison (without backlight)

Benchmark	OLED power consumption (mW)	
	minimum	maximum
idle	38.0	257.3
video	15.1	102.0
Voice call	16.7	112.9
web	164.2	1111.7

Table 6.3 additional power consumption by N1 OLED at minimum and maximum brightness

Chapter 7
Conclusions and future work

7.1. Conclusion

The implementation of XLP, with the novel hardware architecture and the dynamic power scheduling, presents a platform to exercise on different suits of hardware and software experiments. Besides the basic objective of this thesis, Power management, the XLP can be used as a generic platform to develop standard user applications which can leverage on the performance and low power consumption of the device.

In this work a detailed analysis of energy consumption for various benchmarks has been given, based on actual measurements. It is shown how the different components of the device contribute to overall power consumption. A model of the energy consumption is provided for different usage scenarios, and showed how these translate into overall energy consumption and battery life under a number of usage patterns. The modularity of the XLP Smartphone allows one to perform such a detailed analysis and breakdown of its power consumption. This is not possible to the same degree on a typical commercial device.

The detailed measurements are then compared with a coarse-grained analysis of modern phones, and shown the results to be far superior. A minimum of 28% saving on the suspend state implies a tremendous amount of energy saving on the long run since a typical device will spend more than 65% of the time in this state. Around 300% lower power consumption was recorded during the idle state and more than 30% in all the usage scenarios which mainly is due to the stochastic power management employed on the cellular modem. More saving will be gained if the XLP had used a 3G modem because of the better power management feature they provide. The slightly higher power usage on the voice call benchmark can also be traced to the specific modem used in this device. Last but not least, the enhancements made on the chip by Freescale are also major factors in the low power capabilities of the SoC used in the XLP.

7. 2. Limitations

The XLP has several limitations, albeit not major ones, which need to be kept in mind when using the results. Most, if not all, of this limitations are due to the limited budget allocated to research projects by the Institute. The overall expense to implement the XLP was estimated to be around 15000 ETB, but the budget allocated for a master's thesis was a maximum of ETB 7000. Therefore, the XLP was forced to drop some features that would have made it a complete Smartphone. The major ones that are left due to budget deficit are: 3G network, Wi-Fi, AMOLED display, Compass, and camera.

The biggest limitation is that the XLP may not be seen as a latest generation mobile phone, but a few years outdated. The main feature it is lacking is a 3G cellular interface, which supports much higher data rates than the 2.5G GPRS interface. The validation results, however, show that this higher data rate does not appreciably affect power consumption in practical situations. Further, the application processor is based on a relatively dated ARMv5 architecture; however it is clocked at a rate consistent with 2010-vintage smart phones. The difference in power consumption compared with more modern processors can be traced largely to idle power; in other respects, the age of the CPU is not a substantial limitation. More over the Wi-Fi, camera, compass and AMOLED display that are meant to be included in the XLP are not populated on the PCB even if footprints are already in place for them and the software is written to support them. Again the results shown exclude these missing modules and thus the comparison doesn't cover the usage of the aforementioned components.

7.3. Future Work

The ultimate aim of this work is to enable a systematic approach to improving power management in mobile devices. Since power management will remain a major concern in the design of battery powered devices, we hope that by presenting this data, we will enable such future research, both in our lab as well as by others. The limitations of the XLP can be addressed without changing the core concepts introduced in this paper and a more elaborate and advanced implementation can be done thereby achieving even a better result.

References

Note: Not all the references used are not listed down here, but the ones extensively used**.**

[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]. MAHESRI, 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, Dec. 2004.

[3]. Sagahyroon, A. "Power consumption in handheld computers." In Proceedings of the International Symposium on Circuits and Systems (Dec. 2006), pp. 1721–1724.

[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), pp. 327–338.

[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), IEEE Computer Society, pp. 158–168.

[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. (From Elsevier, Science direct - *www. sceincedirect.com)*

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

[8]. Turbo Boost Technology in Intel® Core™ micro architecture (Nehalem) Based Processors, white paper, Intel *corp. (www.intel.com)* 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]. Padmanabhan Pillai and Kang G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," 2001.

[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]. C. 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 *Cpd* Calculation" Applications report, Texas Instruments Inc., June 1997.

[16]. 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

[17]. D. Peterman "The birth of a standard: Unified Power Format (UPF) For low power design intent" in Low Power ICs Workshop, IEEE Design Automation Conference, 2007.

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

[19]. Ravindra Jejurikar, Cristiano Pereira, and Rajesh K. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," *IEEE Proceedings of the Design Automation Conference*. 2004, ACM Press.

[20]. 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*

[21]. 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

[22]. Pedram M. et al. "Dynamic Voltage and Frequency Scheduling for Embedded Processors Considering Power/Performance Tradeoffs." IEEE transactions on VLSI systems, 08 August 2010.

[23]. Dave Reed, "Keeping leakage current under control," http://www. eetimes.com/ story /OEG 20030207S0026, 2003.

[24]. Doug Abbott," Linux for Embedded and Real-time Applications", Elsevier Science, New York, 2003

[25]. Neil Matthews, Richard Stones, "Beginning Linux Programming", 4th edition, Wiley Publishing Inc, Indianapolis, Indiana, 2008

[26]. M. Tim Jones, "GNU/Linux Application Programming", 2nd edition, Course technology, Boston, MA 2008

[27]. Ori Pomerantz, "Linux Kernel Module Programming Guide", a free e-book under GPL. Can be downloaded from the GNU/Linux website.

[28]. "Linux kernel module programming by example" a free e-book under GPL. Can be downloaded from the GNU/Linux website.

[29]. Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, "Linux Device Drivers", 3rd edition, O'Reilly Media Inc, Sebastopol, CA, 2005

[30]. Robert Love, "Linux System Programming", O'Reilly Media Inc, Sebastopol, CA, 2005

[31]. Steve Heath, "Embedded Systems Design", 2nd edition, Newnes (Elsevier Science), Oxford England, 2003

[32]. Sloss, Andrew N. "ARM system developer's guide: designing and optimizing system software ", Morgan Kaufmann Publishers, an imprint of Elsevier, San Francisco, CA, 2004

[33]. Kraig Mitzner "Complete PCB design using OrCAD Capture and PCB editor", Elsevier Inc, Boston, MA, 2009

[34]. i.MX23 Layout and Industrial Design Guidelines, rev 0, Application report, Freescale Semiconductor Inc, Networking and Multimedia group, Austin, TX. Nov, 2009

[35]. ARM926EJ-S Technical Reference Manual, ARM Limited *(http://www.arm.com)*, Rev. r0p5 June 2008

[36]. i.MX23 Applications Processor Reference Manual, Freescale semiconductor Inc., *(http://www. frescale.com)*, Rev 0.1, October 2009.

[37]. i.MX23 Linux BSP user's guide, version 10.05.03, Freescale Semiconductor Inc. May, 2010.

[38]. i.MX23 EVK Linux 10.05.03 Reference manual, Freescale Semiconductor Inc. May, 2010.

[39]. The Linux Target Image Builder (LTIB)*: http://www.bitshrine.org/ltib/*

[40]. Telit product documentation. *http://www.telit.com/GE864-quad*

[41]. Si47xx antenna, schematic, layout, and design guidelines, Silicon Labs, rev 0.4, June 2008

[42]. Si47xx Programming guide, Silicon Labs, rev 0.3, March 2008

[43]. The iMX wiki page *http://www.imxwiki.com* , last accessed June 24, 2011

[44]. The iMX developers community, *http://imxcommunity.org*, last accessed June 24, 2011

[45]. The GNU web site for open source books, software tools, etc: *http://www.gnu.org*.

[46]. For the Linux kernel, patches and updates: *http://www.kernel.org*

Appendix A

Proof for theorem1

Proof of Lemma 1:

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,

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

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.

Proof of Lemma 2

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:

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

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 j^{th} 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. $\mathrm{Exp}(\lambda)$ random variables, and $T_\lambda(k)$ denotes the time between the $(k-1)^{st}$ and k^{th} 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 the term1 as shown below 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.

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

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

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 $\frac{n_i+E[X_i]}{1-\rho}$.

$$E[T] = \frac{E[total\ response\ time\ per\ cycle]}{E[number\ of\ jobs\ per\ cycle]} = \frac{\sum_{i=0}^{N} P_i \sum_{n_i=1}^{\infty} q_{in_i}\frac{r_{in_i}}{1-\rho}}{\sum_{i=o}^{N} P_i \sum_{n_i=1}^{\infty} q_{in_i}\frac{n_i+\lambda T_{S_i}}{1-\rho}}$$

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

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:

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

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

Proof of Lemma 3

To prove that the optimal strategy is pure, we only need to note that the expressions for both the mean response time and average power are of the form

$$E[T] = \frac{q_1 t_1 + \cdots + q_n t_n}{q_1 m_1 + \cdots + q_n m_n}; \quad E[P] = \frac{q_1 u_1 + \cdots + q_n u_n}{q_1 m_1 + \cdots + q_n m_n}$$

Where n is the number of pure strategies that the optimal strategy is randomizing over. For some discrete probability distribution ($q_1 . . ., q_n$). It can be show that when n = 2, the optimal strategy is pure, and the proof will follow by induction on n. For n = 2, we consider E[T] and E[P] as a function of q_1 over the extended domain $q_1 \in (-\infty, \infty)$, and show that there is no local minima of E[T].E[P] in $q_1 \in (0, 1)$. Further, note that both E[T] and E[P] are of the form $a + \frac{b}{c+dq_1}$ for some constants a, b, c, d. The lemma trivially follow if the product of E[T] and E[P] are concave function of q.