Optimization of System Controller Unit

By

Maharshi Shah

Roll No: 10MCEC14

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2012

Optimization of System Controller Unit

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Maharshi Shah

Roll No: 10MCEC14

Guide

Prof. Vibha Patel

Mr. Chandramouli Srinivasan

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2012

Declaration

I, **Maharshi Shah**, **10MCEC14**, give undertaking that the Major Project entitled **"Optimization of System Controller Unit"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in Institute of Technology of Nirma University, Ahmedabad, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Maharshi Shah

Certificate

This is to certify that the Project entitled "Optimization of System Controller Unit" submitted by Maharshi Shah (10MCEC14), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Ms. Vibha Patel
Internal Project Guide,
Institute of Technology,
Nirma University, Ahmedabad

Mr. Chandramouli Srinivasan
External Project Guide and Manager,
Intel Technology India Pvt. Ltd.
Bengaluru

Dr. S.N. Pradhan
Professor and PG-Coordinator,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J. Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

Today low power devices like Netbooks and Tablets have started appearing in the market and they are being used in day to day activities. The End-User has seen the need for a faster and more capable system to work on. So, the race to increase the performance of the system has already started. Among the parts which can be optimized, one is System Controller Unit also known as SCU. SCU is an ARC Microcontroller. It plays major role during the booting of the system and also while the system is running. Current implementation of SCU is not optimized. The main objective of this thesis work is to optimize the current SCU model in such a way that it can fully utilize all the resources available to it and hence it can provide better performance in terms of time and power. This thesis work is mainly concentrated on the firmware part of the SCU.

Acknowledgements

It gives me great pleasure in expressing thanks and profound gratitude to my external guide and manager **Mr. Chandramouli Srinivasan ,** Intel Technology India Pvt. Ltd. and Internal Guide **Ms. Vibha Patel,** Nirma University, for their constatnt guidance. I would also like to thank **Mr. R. Sriram**, Power Management Architect, **Mr Panner Kumar**, SCU Firmware Developer and **Mr S. Siddhartha**, Win8 Driver Developer for their time to time suggestions and the clarity of the concepts of the topic that helped me a lot during this project.

I would like to extend my gratitude to **Dr. S. N. Pradhan**, Professor and M.Tech Coordinator, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad for fruitful discussions and valuable suggestions during meetings and for their encouragement. I would also thank my college, all my faculty members in Department of Computer Science and my colleagues without whom this project would have been a distant reality. Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the first part of my dissertation work successfully.

- Maharshi Shah

10MCEC14

Abbreviations

SCU	System Controller Unit
ACPI	Advanced Configuration and Power Interface
GPIO	General Purpose Input and Output
SoC	System on Chip
FW	Firmware
SRAM	Shared Random Access Memory
PnP	Plug and Play
IPC	Inter Processor Communication
RTC	Real Time Clock
OSPM	Operating System Power Management
PMU	Power Management Unit
PMIC	Power Management Integrated Circuit
UEFI	Unified Extended Firmware Interface
PM	Power Management
APM	Audio Playback Mode
OSM	Optimized Standby Mode
ACU	Audio Controller Unit
DMA	Direct Memory Access
SPI	Serial Parallel Interface
ACK	Acknowledgement
ROM	Read Only Memory
RAM	Random Access Memory
T0	Timer-0
T1	Timer-1
DWORD	Data Word
IVT	Interrupt Vector Table
OEM	Object Equipment Manufacturer
KBD	Keyboard
PMU	Power Management Unit

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today low power devices like Netbooks and Tablets have started appearing in the market. Different companies are coming in market with their own designs. Each device offers different features and gives different performance. System Controller Unit also known as SCU is one of the core parts of Netbook and Tablet which is responsible for the performance of the overall system.

SCU is an ARC Microcontroller. I have worked on the firmware part of SCU. Firmware part of SCU has two main modules. One is BootROM Code module and another is Runtime module. BootROM Code module is used during booting process while runtime module comes in to the picture once booting is completed. During boot process SCU is responsible for initializing ROM and storage devices, initiating boot flow, reading Firmware image and verifying the headers. During runtime, SCU is responsible for IPC, interrupt handling and timer services.

During my thesis, I have gone through the basics of firmware architecture. Then I have studied the firmware part of the SCU and from that I had created a work flow model of SCU. I have studied that model in detail and then I have tried to improve it. The two major areas which I have looked upon are time and power. I have looked in to the possibilities where SCU can be used extensively in Power Management activities.

1.1 Objective of the Work

The objective of this research is to provide a better utilization of the resources and to obtain better performance.

1.2 Scope of the Work

The scope of this work is to optimize the performance of SCU and hence optimize the performance of the device in which SCU is used.

1.3 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2, *Detail study of the Firmware Architecture*

Chapter 3, *Architecture of SCU*

Chapter 4, *Operating System Power Management*

Chapter 5, *Proposed Implementation*

Chapter 6, *Implementation and Results*

Chapter 7, *Conclusion and Future Work*

Chapter 2

Detail study of the Firmware Architecture[1]

2.1 Firmware Organization

Firmware Organization includes many modules. Three main modules are BootROM, Firmware Image and Firmware Recovery. BootROM Code is executed by the SCU microcontroller and resides in the SoC. Firmware Image is located in the storage device. The Firmware Recovery modules are downloaded from an external peripheral source (such as USB or wireless) and executed by the SCU.

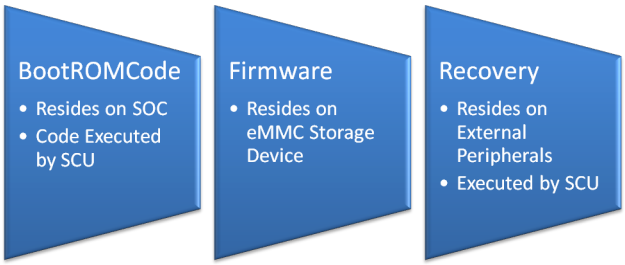

Figure 2.1: Firmware Organization

2.2 Storage Device

Storage device is logically divided into two areas. (1) Boot Partition/ Protected Partition (2) User Partition/ Unprotected Partition. The Protected Partition or Boot Partition contains the header, followed by the Runtime FW. The Unprotected Partition or User Partition contains the OS image, OS level drivers and Root File system. The Unprotected Partition can be accessed by processor using the storage drivers. Access to the storage device is controlled by the mutex controller. The processor can only access the User partition. The SCU can access the boot partition as well as the user partition. It uses a special driver service to access those partitions.

2.3 Firmware Bootflow

Firmware Bootflow is very complex in nature and it includes many steps and different cases. Before the bootflow begins, it is assumed that image of the OS is already loaded in the boot partition of storage device. When system is powered on, BootROM code is executed. SCU takes care of this execution. Then the firmware image is loaded in to the SRAM. Then the Bootflow is handed to processor. Afterwards OS will take the control of the boot flow. When any error occurres during the bootflow then the system goes to OS recovery. FW recovery module is called for this OS recovery. FW Recovery resides on some external USB device. When this module is called, it first initializes the USB device and will start the data transfer from USB to SRAM. Then SCU executes that module.

2.4 Interrupt Routing and Handling

Interrupt Routing and Handling is very flexible. It supports Thirty Two interrupts. Some interrupts are serviced by SCU while others are serviced by processor. To handle SCU interrupts deffered interrupt handling is called whereas to handle processor related interrupts, no-deffered interrupt handling is called. These interrupts are two level interrupts. The first level interrupt tells which subsystem has generated the interrupt while the second level indicates the actual interrupt that has occurred.

2.5 GPIO

Different types of GPIO are available out of which some support the PnP capability while others don't support PnP capability. Device with PnP capability are easily recognize by the system and easily integrated in to the system while non PnP compatible device needs special types of operation and information. These non-PnP compatible devices are integrated using static information which is provided to the OS using some static tables.

2.6 Inter-Processor Communication

When a piece of firmware needs to get information from, or issue instructions to, a different processor, inter-processor communication mechanisms have to be created. In general, there are two groups of messages, downstream and upstream. Downstream messages can originate from either processor or Power-unit. Additionally, there are messages between ACU and SCU. ACU uses its own IPC handling mechanisms for these messages.

2.7 Timer Services

There are two 32-bit internal timers and eight external timers in the SCU . SCU can access the two internal timers. Processor cannot directly access two internal timers. Processor has access to the eight external timers. The platform has two real-time clocks (RTCs). Both are functionally identical, but they may or may not be in sync with each other at all times.

Chapter 3

Overview of SCU

3.1 Introduction

System Controller Unit (SCU) has to take care of many responsibilities from boot time to runtime. Main tasks of SCU during booting includes of loading different firmware, waking up different systems, checking different security related issues. SCU thus plays an important role in preparing the system for use. Main responsibilities of SCU during Runtime includes PM activities; interrupt handling, IPC and handling timer events.

3.2 Architecture of SCU [5]

SCU is an ARC micro controller. It is the subsystem which takes control after reset. It is expected to be on all the time and hence is designed to consume very low power. The SCU subsystem consists of System Controller Unit core (ARC 600 core), on-chip memories (ROM, RAM) and peripherals. The SCU has on-chip code ROM, code RAM and data RAM. The System Controller Unit has bus mastering capability. It has access to SRAM. It implements IPC mechanism to communicate with processor.

The instruction ROM is used to store the bootstrap code that is responsible for bootstrapping SCU after power up. The instruction RAM is used as local RAM for the System Controller Unit. It holds latency critical code of the SCU. It is responsible for boot strapping the processor. The data RAM is used as local RAM for the System Controller Unit. It has 36 different general purpose and special purpose registers.

One Non-Maskable Interrupt pin is used to interrupt the SCU. The interrupt vector table is fixed inside ROM. It is up to the programmer to define a new interrupt vector table location inside SRAM once the execution switches from ROM to SRAM. Both Processor and SCU have ability to interrupt each other using doorbell mechanism. Besides IPC, the interrupt system also captures the interrupt from various devices and processes it or transmits to the processor in a message format using front side bus delivery method. The interrupt unit is configurable and can support up to 32 interrupts. These 32 interrupts are categorized in over two sets of maskable interrupts: level2 (mid priority) and level 1 (low priority). A third level is reserved for exceptions, which has the highest priority. The core does not implement interrupt vectors as such, but rather a table of jumps. When an interrupt occurs, the processor jumps to fixed address in memory, which contains a jump instruction to the interrupt handling code.

The processor timers are two independent 32-bit timers. Timer 0 and timer 1 are identical in operation. The only difference between them is that they are connected to different interrupts. The processor timers are connected to a system clock signal that operates even when the SCU is in sleep mode. The timers can be used to generate interrupt signals that will wake the processor from sleep mode. The processor timers automatically reset and restart their operation after reaching the limit value. The processor timers can be programmed to count only the clock cycles when the processor is not halted. The processor timers can also be programmed to generate an interrupt or to generate a system reset upon reaching the limit value.

The SCU FW implements logic for IPC. These registers will be implemented in a separate hardware block and will be primarily used for communication between host processor and SCU. Apart from processor, other subsystems are also expected to send message transactions to the SCU using this mechanism. The IPC hardware along with IPC firmware would be responsible for supporting all IPC functionality. The IPC unit is designed to asynchronously pass control messages and data between the processor and SCU. The message could be as simple as a write or more complex where the processor can request the SCU to perform some task, collect some data and make the SCU write that data in a specific SRAM location.

The SCU DMA serves two purposes: One is to provide a way for the SCU to move data

to and from DDR since it has no way of doing that using load/store instructions. This includes the ability to move data directly from DDR into the instruction RAM of the SCU. And another is to provide handshaking interface that enable SPI controllers to do DMA operations.

3.3 Bootstrap Workflow [2]

When the system is switched on, SCU performs following steps.

- Initialize bootstrap stuff
  - **Initialize T0:** In this step, T0 is allowed to generate delays when system needs it.
  - **Read from Flash:** In this step the top DWORD is read to obtain starting address of the images or headers.
  - **Copy data from SPI to execution RAM:** This is done to keep a seperate copy of the bootsrap code which can be used if the original copy gets corrupted.
  - **Jump to execution RAM:** Actual execution starts from the RAM.
  - **Enable security engine:** This initializes the security engine which plays a major part in checking the images or headers for verification.
  - **Initialize the SRAM:** This step allows SCU to copy anything needed for execution to SRAM.
- Initialize Storage Device: This allows reading from storage devices images and headers which are required for actual runtime execution.
- Initiate cold boot process
  - **Read FW Image:** This is the first step of cold boot process. When this image is read different headers are also read along with it.
  - **Verify Different Headers:** Different headers are checked for security related issues and also checked if any of them are corrupted. In that case, firmware recovery gets initialized.

  - **Verify Different Firmware:** Different firmware like SCU Runtime, PM Unit and Processor firmware are checked if any of them are corrupted..
  - **Disable all interrupts:** All the interrupts are disabled so that if the user provides some input through different interface, they will be ignored.

3.4 Runtime Workflow [2]

- **Configure GPIOs:** GPIOs are configured from the tables that are available to SCU FW.
- **Setup Runtime Interrupt Vector Table:** IVT is initialized and used later to handle different interrupts .
- **Subsystem initialization:** All the subsystems are brought back from their reset state and initialized to their default state.
- **Enable IPC interrupts:** Interrupts between different subsystems are enabled. Different subsystems like processor and PMU can communicate with each other using IPC interrupts.
- **Enable timer for interrupt:** Timers can be used by different subsystems in case they need timer interrupts.
- **Initialize Clock:** Clock is now initialized and synchronized with the master clock which is real time clock.
- **Enable master interrupt control:** All the interrupts which were disabled during last step of bootstrap are now enabled and now the system can generate the interrupts which are handled by SCU.
- **SCU Idle loop:** SCU goes into the waiting state for some event to occur.

Chapter 4

Operating System Power Management [3]

4.1 Overview

OSPM's (Operating System Power Management) primary function is to efficiently manage power by controlling platform and subsystem power states. OSPM uses the concept of modes in order to determine the most power efficient state for the platform at any given point in time. A mode represents a comprehensive power model of the platform which provides access to all the resources required to support a specific usage model. Subsystems which are not explicitly required for a given usage model and the associated mode will be placed in a low power mode. The SCU has inherent knowledge of the subsystem PM capabilities, constraints, and implementation, will direct subsystem specific actions to implement a specific state.

4.2 OSPM Architecture

Power Manager is a Control Panel application that manages devices independent of the base PM model. The Power Manager interface provides flexibility to OEMs and device driver developers without sacrificing compatibility with the base model. In the base power model, devices receive notification that the OS is suspending and resuming. This notification occurs in an interrupt context, so devices are restricted regarding what they can do during

a suspend state and how long they can take to do it. The following illustration describes the PM architecture.

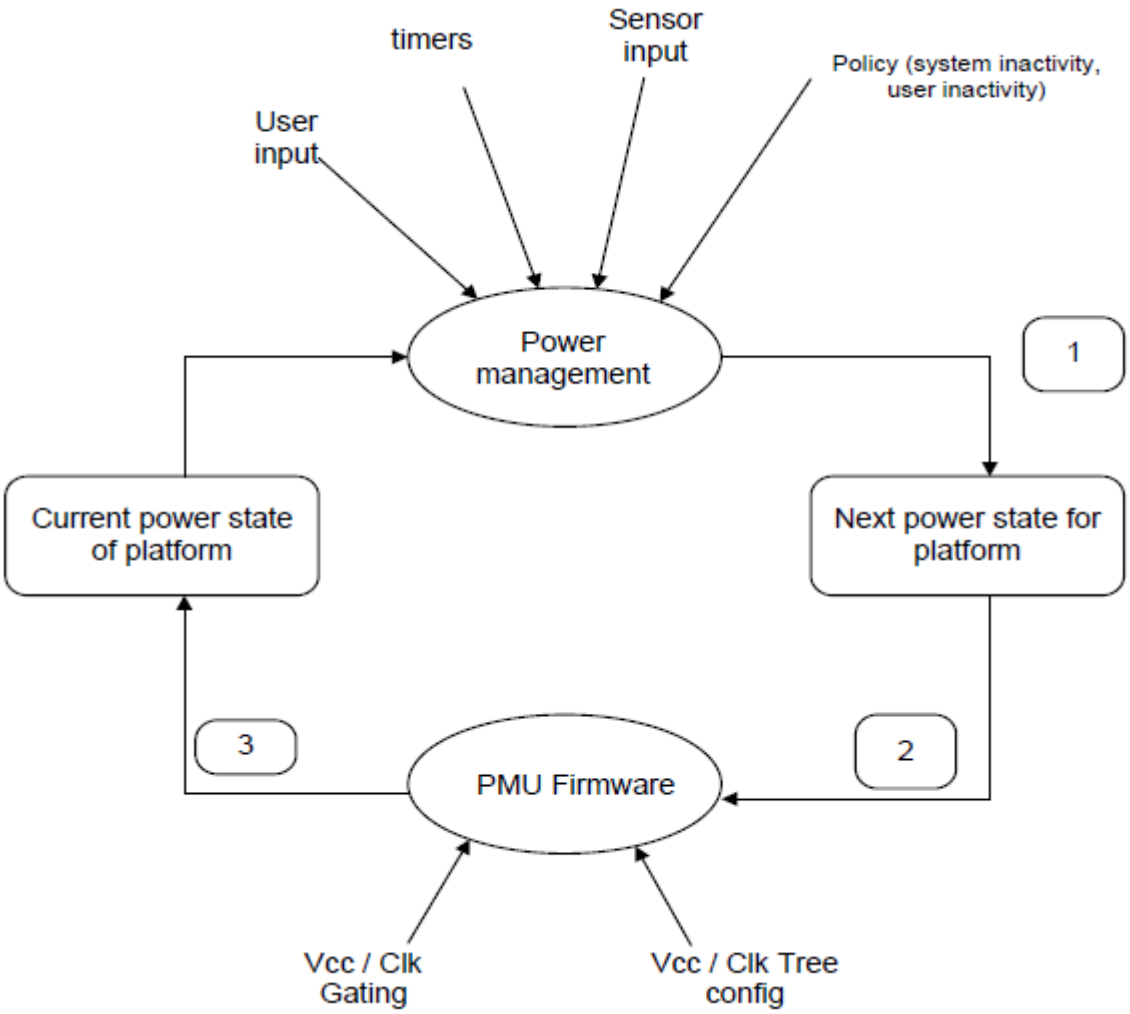

Figure 4.1: OSPM Architecture

Using Power Manager, devices receive power state change notifications as I/O controls. Because I/O controls run in a thread context, driver developers have much more flexibility in how they implement the power state change. Using I/O controls to manage power also enables separation of device power state from overall OS power state. Thus, some devices can be turned off while the OS is running, and others can be left on while most of the OS is suspended. In addition to managing device power, Power Manager notifies applications about power-related events. For example, Power Manager informs interested applications

when the OS resumes from a suspend state.

- OS power states impose maximum power consumption limits on all devices.
- Application imposes minimum power consumption limits on specific devices to obtain minimum performance levels.
- Power Manager allows devices to intelligently manage their own power, as long as their power levels are between the maximum and minimum limits.
- If the minimum power consumption limit is set higher than the maximum, the power of the device remains elevated for as long as the application requires the device.
- Devices can implement one or more device power states. Device power states are limited in number.
- If the OS transitions to the suspended state, application-imposed minimum power limits are set aside while the OS is in a suspend state.

a. OSPM Framework: A message Framework for applications, platform services and other OEM policy managers, User interfaces, etc to plug in.

b. Policy Manager: A pluggable Policy Engine, which can be smart enough to make key mode transition/Policy decision for a given application usage as well the environment the application running in

c. Event Manager: An Environment scan module which monitors various events that happens in the platform, Sensor inputs, and System profiles. This module is responsible for influencing the PM to make run time mode transitions.

d. Platform Power Manager: A Platform Power Manager, which will implement the policy taken by the Policy engine.

e. Activity Timers: For each timer, Power Manager checks for a timeout and an optional list of wake sources.

4.2.1 OSPM Framework

The OSPM framework is a host for required OSPM components such as the system profiler, power policy engine, platform power manager, and sensor manager. The framework should run as a separate process dedicated to the functions specified in this architecture specification. The framework also exposes a set of interfaces available to other user level components. These interfaces provide the ability for Software components to convey mode usage, performance requirements, and resource usage. This information is then provided to OSPM components to allow them to project mode changes on the platform. The framework also provides an interface which allows OSPM components / plug-ins to be dynamically created / deleted. This action would typically be initiated by the event handler. The OSPM framework provides external interfaces for applications, services, and drivers to communicate with OSPM components. The framework encapsulates all the other OSPM components within the process space of OSPM. The framework also supplies a set of API's which provide for inter-module communication within OSPM. The API's will abstract the actual communication mechanism from the components so that any messaging interface can be used for this purpose.

4.2.2 Policy Manager

Policy manager is responsible for picking the correct policy for a given usage. On init, a static user defined policy of a given platform is loaded, and as the application launched, depending on various runtime events, the policy manager actively chooses the correct policy and notifies the power manager to implement that policy. It also notifies the user appropriately on any change in the policy. The policy engine or the logic that chooses the correct policy is designed as modular, so that user can plug in their own policy Manager. The policy selection algorithm can be a very simple static look up to a very complex logic. The idea is the OEM can add value differentiation through the policy engine

4.2.3 Event Manager

Event manager is responsible for doing the environment and system scan and provides input to the Policy manager for any change in environment. It's the policy manager which

eventually decides what the correct policy for a given change in environment event.

4.2.4 Platform Power Manager

Power manager is the core execution piece of OSPM framework. It includes

a. PMU Driver: A platform specific PMU Driver, who drives the System PMU

b. Power Manager Engine: Responsible for coordinating device drivers, PM and CPU coordination of sending PM events

c. CPU and Device driver interface

d. Event and Policy manager interface

e. PM Event handler: It is responsible for handling PM events and optional device, CPU event.

Power manager expects all managed devices to support one or more device power states. There are limited numbers of device power states, and the device must inform power manager of its power consumption characteristics. Device power states generally trade off performance for low power consumption. Some applications may require that a device be maintained at a certain device power level. For example, a streaming audio application might require that its network card and audio codec stay powered at a high level while music is playing. A streaming video application might need network and audio, and it might want to keep the display from going into screen-saver mode and keep the backlight on. Applications can request that Power Manager set minimum device power-state requirements.

4.3 Clock and Voltage Control

Subsystem power state change by PM logic in SCU involves controlling the voltage and clocks to subsystems. Each subsystem is supplied current by a set of power rails. A voltage and clock tree infrastructure is defined to control the voltage and clocks for any particular subsystem of platform. The Voltage regulation mechanism provides two knobs to control subsystem power state:

a. Voltage Regulators in the PMIC that supply the power rails to platform. The power rails are subdivided to form a tree structure. The Voltage Regulators in PMIC can be controlled by System Controller using the SPI bus interface through PMIC registers.

b. Power Gates that provides an isolation mechanism for each subsystems in order to achieve independent voltage islands. These Power Gates can be controlled by SCU firmware through PMU register.

4.4 Logical to Physical Subsystem Mapping

The logical subsystem definition is used by OSPM to communicate to SCU. Physically, each logical subsystem consists of one or more physical subsystem. A physical subsystem has a dedicated clock supply with independent power/clock gate. SCU firmware controls the power and clock gates of the physical subsystems as guided by the OSPM driver. SCU firmware contains a table that maps logical subsystems to constituent physical subsystems. This logical to physical subsystem mapping table is hard coded in the SCU Runtime firmware.

4.5 Activity Timers

Upon initialization, Power Manager reads the registry to acquire a list of activity timer names. For each timer, Power Manager checks for a timeout and an optional list of wake sources. It then creates the following named events:

a. A timer reset event

b. An active status manual-reset event

c. A manual-reset event

If the timeout associated with the timer expires without a reset event, Power Manager signals the active event and sets the inactive event. If the reset event is signaled, power manager signals the inactive event and sets the active event. The first event is an auto-reset event that any driver can signal to indicate system activity. Drivers that support resetting

activity timers read the name of the timer reset event from the registry. The other events are manual reset events, of which only one is signaled at a time. Drivers, applications, or power manager itself can open handles to these events to determine whether the timer has expired. These events indicate the status of the activity timer. The reset event is an input to the activity timer system, and the status events are the corresponding outputs. Any number of drivers can open handles to the reset event of an activity timer. A driver should read the name of the event from the registry to obtain OEMs customizations set in the registry. By customizing the registry, the OEMs decide how the activity of a driver is interpreted by power manager. Potential sources of activity include all drivers that open handles to the reset event of the activity timer. When the system is suspended, power manager signals the active manual-reset events associated with activity timers. On a resume event, it scans the timers to see if any are associated with the wake source that caused the system to resume. If it finds a match, that activity event is signaled. This enables power manager to resume in the system power state associated with that activity timer. A keypad press or touch-panel tap can reset the activity timer.

4.6 Wake Events

A wake event is defined as any asynchronous event that takes the platform out of standby mode. One example is a wake event generated by a communication device. Typically, all wake events have to be communicated to the host in the form of an interrupt. Upon receiving a wake event, the waking sub-system may need to be woken up at a minimum in order to receive the in-band data. In addition, the SCU also wakes up subsystems that have been identified by the OSPM in the Wake Config/Status registers. From an implementation standpoint, wake is essentially another interrupt to the ARC core. For example, when a KBD key is pressed in standby mode, the KBD controller generates an interrupt to the SCU. Since SCU is aware that the platform is in standby mode, this interrupt now is treated as a wake event. If the platform were not in standby, this would be a regular interrupt from the host and no wake sequence is initiated. There are sub-systems that can generate a wake event when the controller is either power or clock gated. These situations are handled on a case-by-case basis. In some scenarios, the regular interrupt line cannot initiate a wake

sequence; instead a separate routing from the wake capable buffer is used from the pins directly, or an entirely separate GPIO outside of the interface must be used.

4.7 Advanced Configuration and Power Interface

4.7.1 Overview

ACPI aims to consolidate, check and improve upon existing power and configuration standards for hardware devices. It provides a transition from existing standards to entirely ACPI-compliant hardware, with some ACPI operating systems already removing support for legacy hardware.

4.7.2 OSPM responsibilities

ACPI requires that, once an OSPM-compatible operating system has activated ACPI on a computer, it then takes over and has exclusive control of all aspects of PM and device configuration. The OSPM implementation must expose an ACPI-compatible environment to device drivers, which exposes certain system, device and processor states.

Power states [6]

The ACPI specification defines the following states for an ACPI-compliant computer-system.

- System States

  To the user, the system appears to be either on or off. There are no other detectable states. However, the system supports multiple power states that correspond to the power states defined in the Advanced Configuration and Power Interface (ACPI) specification. The following table lists the power states from highest to lowest power consumption.

- Device States: D0, D1, D2, D3, D4

Device power state definitions are statically predefined. Power Manager passes a device state to a driver, and the driver is responsible for mapping the state to its device capabilities, and then performing the applicable state transition on the device. The following table describes device power states.

Power state	ACPI state	Description
Working	S0	The system is fully usable.
Sleep	S1,S2,S3	The system appears to be off.
Hibernation	S4	The system appears to be off.
Mechanical Off	G3	The system is completely off and consumes no power.

Table I: System Power States

Device power state	Registry key	Description
Full on	D0	On and running.
Low on	D1	Fully functional at a lower power
Standby	D2	Partially powered, with automatic wakeup on request.
Sleep	D3	Partially powered, with device-initiated wakeup
Off	D4	Device has no power.

Table II: Device Power States

A physical device does not have to support all of the device power states. The only device power state that all devices must support is the full on state, D0. A driver that is issued a request to enter a power state not supported by its device enters the next available power state supported. For example, if power manager requests that it enter D2 and it does not support D2, the device can enter D3 or D4 instead. This can be done if power manager supports one of these states. If a device is requested to enter D3 and cannot wake up the system, it should enter D4 and power off, rather than staying in standby. These rules are intended to simplify driver implementation.

Power manager appropriately maps system power states to the corresponding device power states. For example, if a device only supports device power states D0 and D4, power manager does not immediately request that the device enter the D4 power state when it transitions from the full on power state. Power manager waits until the system enters a system power state in which D3 or D4 is configured as the maximum device power state for that device. If D0, D1, or D2 is configured as the maximum power state, power manager keeps the device at D0.

Chapter 5

Proposed Implementation

5.1 Current Implementation

Currently power management is handled by operating system. SCU and PMU help operating system in managing the power for the system. We have only three possible power modes: Normal Mode, Sleep Mode and Hibernate Mode. User can select any of these modes. OSPM is notified when user selects a power mode. Then OSPM notifies SCU for the current power mode and then SCU executes that power mode with the help of PMU.

5.2 Drawbacks of Current Implementation

Currently there are 64 subsystems in the system. After power on, all the subsystem are in active mode if the system is running in normal mode. According to user operations, these subsystems are utilized but generally all systems are not in active mode at same time. This means that most of the time, many subsystems are wasting the power that they are getting as they do not have any work to do. So, PM is not yet optimize specially in Tablets where battery power plays an important role.

5.3 Proposal of new Implementation

As discussed earlier, we are not utilizing battery power at its best. What we can do is, we can put some of the subsystems in sleep mode while they are not used and powered on

them when they are needed. For this purpose, we must know which subsystems are used at what time. Then we can create a list which defines which subsystem are active during which activities. We can create our user defined power modes in which we can optimize our system for a specific task. For example, documentation. Many people use their tablets or laptops for documentation work. Half of the subsystems remain inactive during this task. So, it is safe to turn off those subsystems which are inactive. When user wants to do some other work, the system can be placed in normal mode. Same way we can do for audio/video play back or internet surfing. Even we can combine them when and where ever it is possible. The basic idea behind this whole implementation is simple. Consider an example of a home: Assume home is a system and each room is a subsystem. At some moment of time, we will be using only some rooms of the home. So, its better to switch of the lights of those rooms. Same thing applies to any system. While system is running, all subsystem will not be used. So, its better to power off those subsystems. Following figure shows the system in three different states. One is totally turned off while another is totally turned on and the last one is partially turned on. Total turn off means when the system is switched off and it is not consuming any power. Total turn on means system is fully running with each subsystem is turned on and system consumes the most power. Partially turned on system is what I am talking about here when the unused subsystems will be turned off and only used subsystems will be turned on. Based on the above idea, I have decided two power modes

- Audio Playback Mode (APM)
- Optimized Standby Mode (OSM)

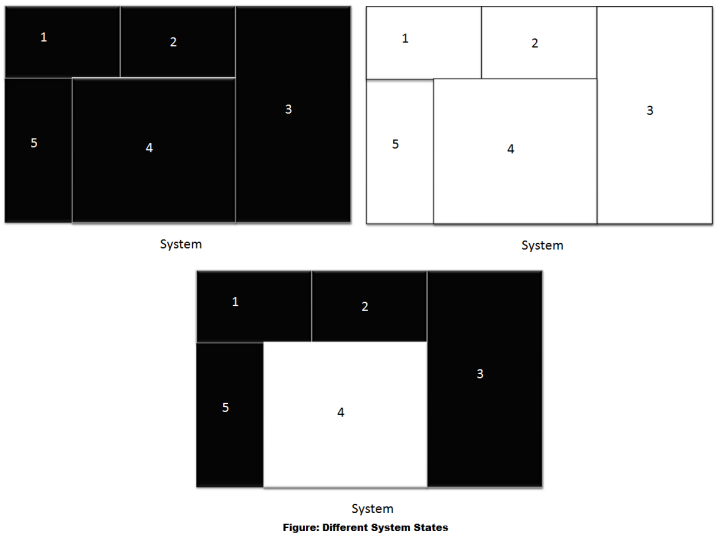

Figure 5.1: Overview of Optimization

5.4 Audio Playback Mode

For Audio Playback mode I have created a mapping of the subsystems which indicates which subsystems will be active and which subsystems will be in sleep state. For this mode I have decided the binary value which will be written by OSPM in the SCU register. When the user selects such power mode it will be communicated to OSPM. OSPM policy manager will write that encoded value in the SCU register. Hence SCU will be notified for this mode change. SCU will refer to the mode-system mapping table in the memory. Then according to the table it will put the subsystem in to respected states. The SCU maintains two types of tables. One table contains physical to logical subsystem mapping and another table to maintain voltage and power rails for each subsystem. We will add another table which will have mapping for each mode and the subsystems related to it.

My first level of implementation did not show big improvement in power saving. Reason behind that was that small subsystems do not consume much power. So even if we switch off the unused systems, the power consumption of the overall system would not be affected much. The entity which makes the biggest difference here is the processor. It is one of the most power consuming entity in whole system. So, we can keep the processor in the sleep

mode and let audio engine handle the audio playback. Audio engine supports all kind of audio playback. So once the user selects this power mode, the system will automatically turn off the unused systems including processor and then playback is handled by the audio engine. Theoretically it really saves a lot of power and so as practically.

5.5 Optimized Standby Mode

The system enters standby mode based on a number of criteria, including user or application activity and preferences that the user sets in the power options application in Control Panel. By default, the system uses the lowest-powered sleep state supported by all enabled wake-up devices. Before the system enters standby mode, it determines the appropriate sleep state, notifies applications and drivers of the pending transition, and then transitions the system to the sleep state. In the case of a critical transition, such as when the critical battery threshold is reached, the system does not notify applications and drivers. Applications need to be prepared for this and take the appropriate action when the system returns to the working state. The system wakes from standby mode in response to user activity or a wake-up event defined by an application. The amount of time it takes the system to wake depends on the sleep state it is waking from. The system takes more time to wake from a lower-powered state than from a higher-powered state because of the extra work the hardware may have to do (stabilize the power supply, re-initialize the processor, and so forth). The system takes the most time to wake from hibernation (S4) because it must read the hibernation file.

Current standby mode also known as S3 has some limitations like only CPU memory is persistent while other memories are flushed once system goes in to sleep mode. Also resume from standby mode takes around 5-7 seconds. Once the system goes into standby mode it becomes offline meaning if we want to receive some updates from internet, it is not possible. We have to turn on the system and then manually update the emails or RSS feeds. Here arise the requirement of new optimized standby mode. In OSM system will always be connected to internet.

This implementation becomes challenging as we need to update the system depending on the user or architecture preference. We as a developer needs to decide which systems will be

on and which events or interrupts can wake the system. For simplicity I have started with timers and USB as wake events meaning once the system enters in to the sleep/standby mode either timers or USB interrupts can wake the system. Here the devices which can wake the events are configured using wake register of PM block in SCU. Before the system goes in to standby mode, these registers are needs to be configured. So, when the system enters into standby mode, these subsystems does not go to sleep mode instead they will be in normal D0 device state. So, for example, if USB is a wake event and once the system is in standby mode, if we generate any USB interrupt for example moving or clicking USB mouse or pressing a key on keyboard or entering a USB device in the system, system will wake from standby mode. One thing we assume here is once the display goes off, user is not going to communicate with the system.

Chapter 6

Implementation and Results

6.1 Overview

The user will select a power mode from OS interface. From that interface, OSPM will come to know about the mode user has selected. OSPM will communicate this mode change to SCU by writing to some register. SCU will in turn execute that power mode with the help of PMU and PMIC.

6.1.1 OSPM State Management

OSPM supports multiple system idle states. System idle states can be directly co-related with a specific mode. These states require a wake event to bring the system back to the system active state (S0i0). The wake event can range from a timer event, to a network related event. The OSPM must perform a series of actions to enter and exit and OSM state. Based on the target platform state, OSPM writes the wake configuration registers to direct the PMU's to return to a specific configuration upon the detection of an enabled wake event. OSPM must also program the wake enable registers to define what subsystem wake events will be promoted to system PM wake events. Once these registers are properly configured, the OSPM issues a command to the PMU's to trigger on an asynchronous event which will be determined by the mode table entry. OSPM will then generate a request to the PMU to enter OSM state. An example of this sequence is shown below:

a. OSPM determines that a particular mode of operation should be entered (MP3 play-

back, etc)

b. OSPM refers to the mode entry table to determine what state should be entered upon resume and how to program the wake configuration register, PM wake control register.

c. OSPM writes wake configuration registers such that subsystem will enter state after a PM wake event.

d. OSPM writes PM wake control registers such that a wake event from subsystem will generate a PM wake sequence

e. Writing of register will allow a signal to be sent to the PMU that the PMU will trigger on the wake event from subsystem and generate a PM wake sequence

f. OSPM sends a command to the PMU to enter the state by writing to command register.

6.1.2 System State Entry/Exit

System State Entry

The steps for power down are as follow.

a. OSPM decides to place the system into OSM and programs OSPM registers.

b. OSPM sets the command type and writes the command register.

c. SCU informs Power Unit of state using Go-OSM message.

d. Power Unit places north subsystems into proper states.

e. SCU receives Ack-SLEEP and Ack-OSM messages.

f. SCU saves the GPIO value programming for all GPIO's that will lose power.

g. SCU takes over the flow, and arms keepers using the signal in each family.

h. SCU walks through south subsystems.

i. SCU continues to shut down system rails for the required system state.

System State Exit

The steps for power up are as follows.

a. SCU detects a wake event, takes any special action based on source.

b. SCU enables the main-GPIO subsystem, and restores values it saved on entry.

c. SCU disarms the keepers by asserting signal.

d. SCU walks through PMU registers and brings required subsystems to their wake state

e. SCU sends GoC0 if full wake required.

f. OSPM restores alternate function mapping as needed for controllers that lost power.

6.2 Challenges in new Implementation

The biggest challenge in implementing this idea is that it is not one level task. User can choose the power mode at OS level, in turn OSPM comes to know about the mode change and then OSPM notifies SCU about this mode change. Then SCU works with PMU to implement the mode change request. So, we need to make changes at many levels and communication needs to happen between all the layers of the system.

6.3 Workflow

Audio Playback

In this mode, the ACU can download the audio media into SRAM. Frequent CPU interaction is not required once the media playback has started; the CPU may need to awaken only when the ACU media buffers are exhausted. As a result, OSPM can put the platform into APM such that only the audio subsystem is on and rest of the system state is as per APM. When ACU runs out of the audio data in SRAM and needs to access the System Memory, it sends Link-Up IPC message to SCU. SCU powers up the platform to partial sleep state. SCU then sends back a "Link-Up-Ack" message to ACU. ACU can now DMA the audio data from system memory(DDR) to SRAM. After the ACU DMA is complete, ACU sends

a "Link-Down" message to SCU. SCU transitions the platform to APM state. If ACU requires device driver services; it can trigger an interrupt to processor. SCU will wake the platform to S0 state. Detail Implementation is as follow:

- OSPM checks the "Busy" bit in the PMU status register to ensure that it can initiate a mode change
- OSPM enables subsystems to generate wake event by writing into the OSPM Wake Configuration register
- OSPM enables interrupts to be propagated by setting the interrupt enable bit in the Interrupt Control register
- OSPM configures the target state of each subsystem in platform standby state by writing to the subsystem configuration register
- OSPM configures the Exit state of each subsystem from a wake event by writing into the PM-WSSC register
- OSPM issues the set configuration command.
- PMU hardware sets the 'Busy' bit in the PMU Status register
- PMU hardware generates an interrupt to SCU when OSPM writes to the PMU CMD register
- Determines the cause of interrupt as an OSPM configuration change by reading the PMU Interrupt register
- Registers the interrupt as an OSPM initiated one - sets a flag. SCU firmware should prevent propagation of MSI. Set the MSI-DISABLE Bit. Servicing is deferred to the main loop of the SCU Firmware
- Reads PMU Command register to determine the platform target state.
- Reads PMU Command register to determine if there is a trigger to start processing. Valid trigger is a PME event from ACU. Any other trigger - log as invalid and exit

- The config registers are read and exit config is prepared for optimization of the flow
- Waits for Ack-SLEEP message from Power Unit.
- IPC2 hardware receives the Ack-SLEEP from the Power Unit (SCU firmware services the IPC interrupt and receives the SLEEP trigger.)
- If platform is entering Low power audio playback mode, then SCU firmware waits for "link down" message from ACU. "Link down" message indicates that there are not ongoing DMA transfers by ACU, and that it safe to transition the platform to APM Background state(LPMP3 mode)
- SCU receives IPC message from ACU, and decodes the message code to "link down".
- SCU Firmware sends GO-APM message to System
- SCU firmware waits for Ack-OSM message from Power Unit to be received via IPC
- IPC2 hardware receives the Ack-OSM from the Power Unit. ARC firmware services the IPC interrupt and receives the Ack-OSM message.
- If Ack is for APM continue else abort
- SCU transitions the Power down/up subsytems per PM-SSC config
- Turn off the respected rails
- ACU has exhausted the audio media in SRAM and needs to access the System memory
- ACU send a "link-up" request to SCU by trigger ACU-to-SCU IPC interrupts.
- ACU waits for an "acknowledgement" from SCU
- SCU decodes the ACU message to be a "Link-Up" request
- SCU checks the current platform state. If the platform is in Low power audio mode, as indicated by the platform state, SCU initiates the partial wake sequence.
- SCU power up the rails

- SCU compares the current sub-system power state with the target state for partial S0 wake.
- SCU powers up required sub-systems
- The sequence of power modes is through M7-M6-M4
- Go to Power mode M5
- Wait for Ack APM message from Power Unit
- If security engine power up is required then power up the same
- SCU sends "Link-Up-Ack", acknowledgment for the "Link-Up" request, to ACU
- On receiving the Ack, ACU proceeds to download audio media from System memory to SRAM using ACU DMA engine
- On completion, ACU send a "Link-Down" message to SCU
- On receiving the "Link-Down" messages, SCU initiates APM Background stand-by entry sequence.
- The CPU is already in SLEEP state. Link-down message is already received, so SCU can skip these steps during re-entry.
- SCU Firmware sends GO-APM message to Power Unit

Optimized Standby

In this mode, system waits for all the system to become Idle. When the system becomes idles, SCU start putting the subsystems in to D3 state. System provides each device the capability to wake the system. The devices which request for wake event will be kept in D0i1 while all other devices will be kept in D3. SCU and processor also goes into Sleep mode. Only devices which had requested for the wake event will be active in low power mode. If any interrupt comes, it first comes to SCU. SCU will notify OSPM about that interrupt. Then OSPM determines whether that interrupt has requested for the wake event or not. If it has not registered as a wake source it will be ignored. If it has requested for the

wake event, OSPM replies back to SCU that it is a wake event and make system available to user by keeping all the subsystems in D0 state. SCU gets the message and does the task and then sends the end of interrupt to OSPM. Hence system again becomes live for the user to use.

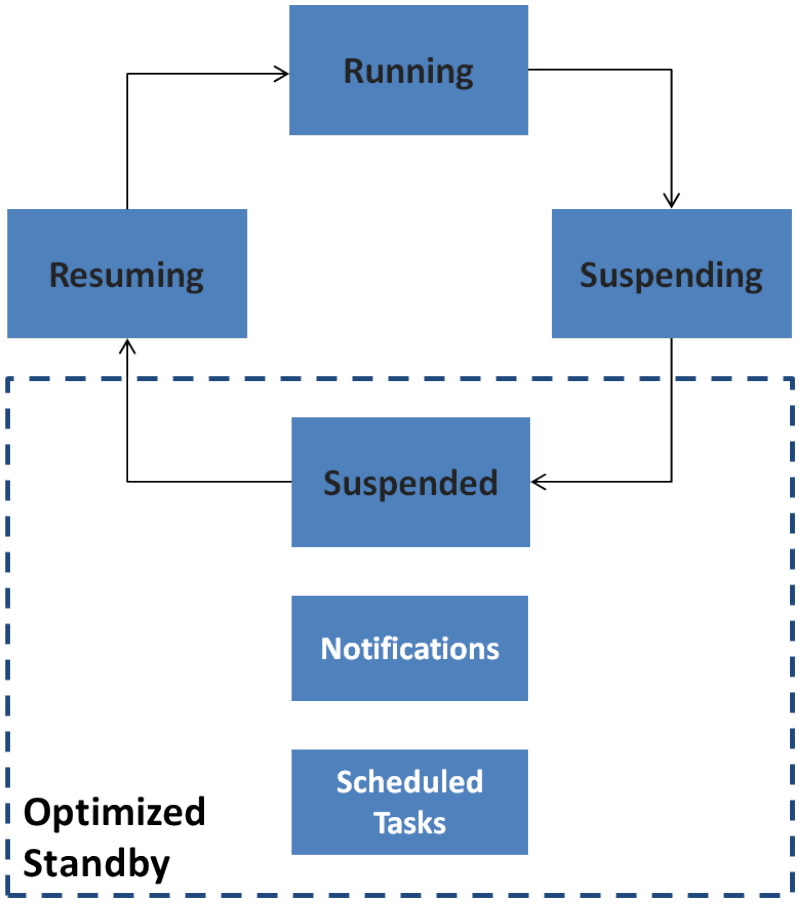

Figure 6.1: Optimized Standby

Detail Implementation is as follow:

- OSPM checks the "Busy" bit in the PMU status register to ensure that it can initiate a mode change
- OSPM enables subsystems to generate wake event by writing into the OSPM Wake Configuration register
- OSPM enables interrupts to be propagated by setting the interrupt enable bit in the Interrupt Control register

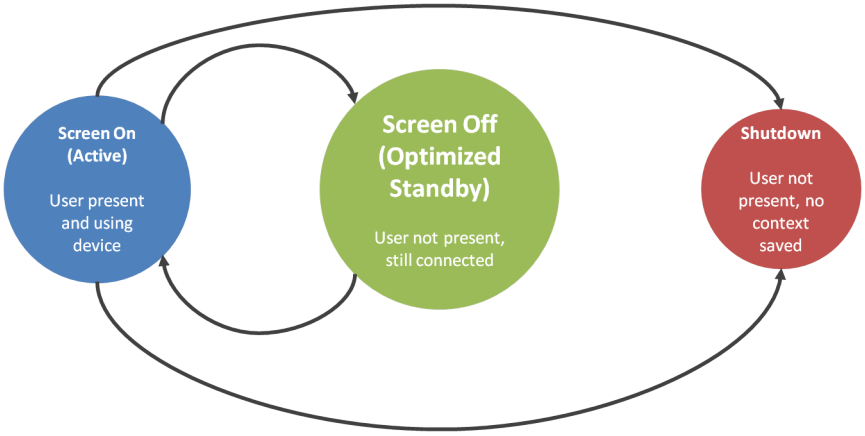

Figure 6.2: Optimized Standby

- OSPM configures the target state of each subsystem in platform standby state by writing to the subsystem configuration register
- OSPM configures the Exit state of each subsystem from a wake event by writing into the PM-WSSC register
- OSPM issues the set configuration command.
- PMU hardware sets the 'Busy' bit in the PMU Status register
- PMU hardware generates an interrupt to SCU when OSPM writes to the PMU CMD register
- Determines the cause of interrupt as an OSPM configuration change by reading the PMU Interrupt register
- Registers the interrupt as an OSPM initiated one - sets a flag.
- Reads PMU Command register to determine the platform target state.
- Reads PMU Command register to determine if there is a trigger to start processing.
- The config registers are read and exit config is prepared for optimization of the flow
- Waits for Ack-SLEEP message from Power Unit.
- IPC2 hardware receives the Ack-SLEEP from the Power Unit

- SCU Firmware sends GO-OSM message to System
- SCU firmware waits for Ack-OSM message from Power Unit to be received via IPC
- IPC2 hardware receives the Ack-OSM from the Power Unit. ARC firmware services the IPC interrupt and receives the Ack-OSM message.
- If Ack is for OSM continue else abort
- SCU transitions the Power down/up subsytems per PM-SSC config
- Turn off the respected rails
- SCU checks the current platform state. If the platform is in OSM, as indicated by the platform state, SCU initiates the partial wake sequence.
- SCU power up the rails
- SCU compares the current sub-system power state with the target state for partial S0 wake.
- SCU powers up required sub-systems
- Go to Power mode M5
- Wait for Ack Sx message from Power Unit

6.4 Results

After implementation, I carried out different experiments on the systems available to me. I compared the power consumption with the legacy implementation and found that my implementation improves the results and shows less power consumption.

Audio Playback

The following graph shows the power consumption of overall system when mp3 music is being played in the legacy mode for the duration of 60 seconds and observations were recorded each second.

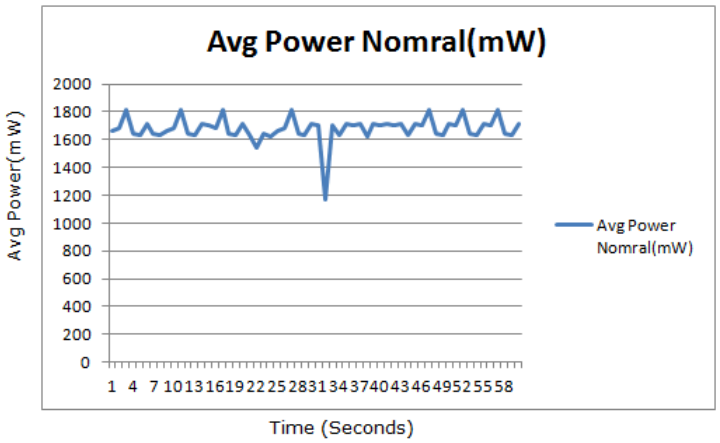

Figure 6.3: Normal Playback

The following graph shows the power consumption of overall system with optimization 1 when mp3 music is being played with APM mode and processor is not in sleep mode for the duration of 60 seconds and observations were recorded each second.

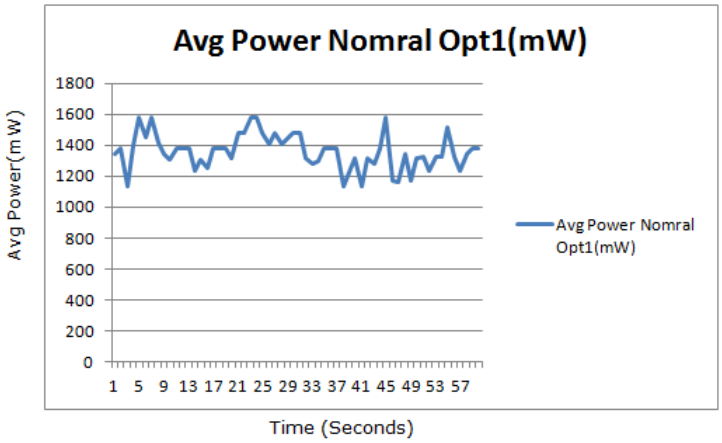

Figure 6.4: Audio Playback: Implementation 1

The following graph shows the power consumption of overall system with optimization 2 when mp3 music is being played with APM mode and processor is in sleep mode for the duration of 60 seconds and observations were recorded each second.

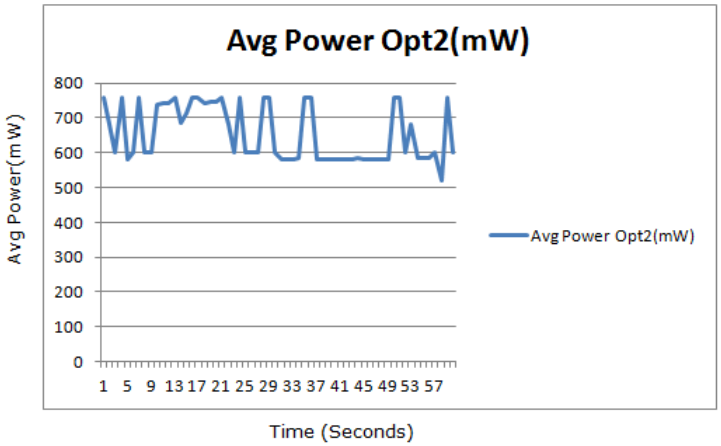

Figure 6.5: Audio Playback: Implementation 2

The following graph shows the comparison of all three scenarios.

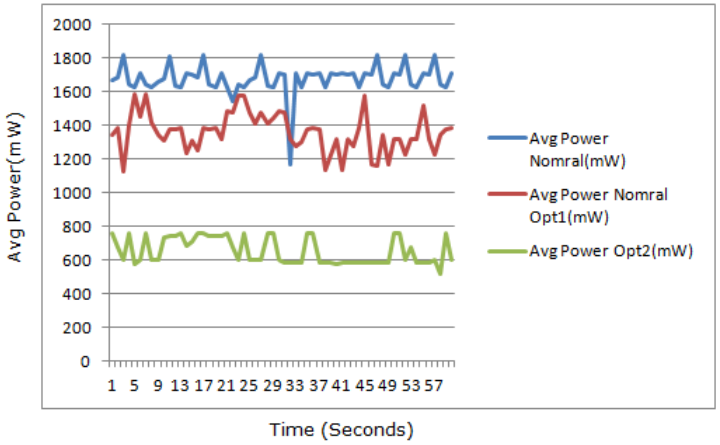

Figure 6.6: Comparison of Normal Playback and Audio Playback Mode

Optimized Standby

The following graph shows the power consumption of overall system when the system is idle for the duration of 60 seconds and observations were recorded each second.

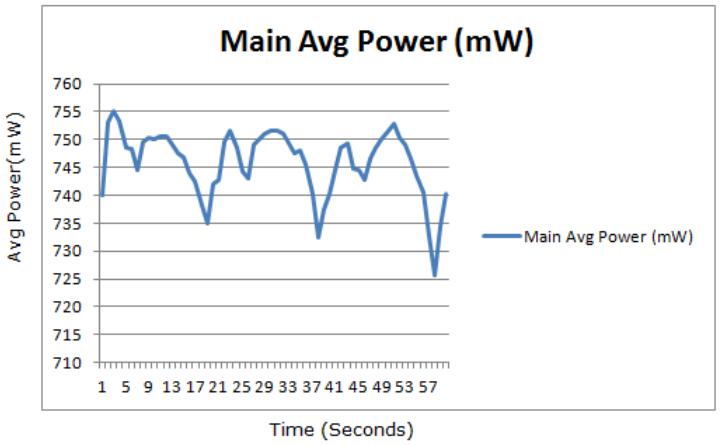

Figure 6.7: Normal Standby Mode

The following graph shows the power consumption of overall system when the system is in optimized standby mode for the duration of 60 seconds and observations were recorded each second.

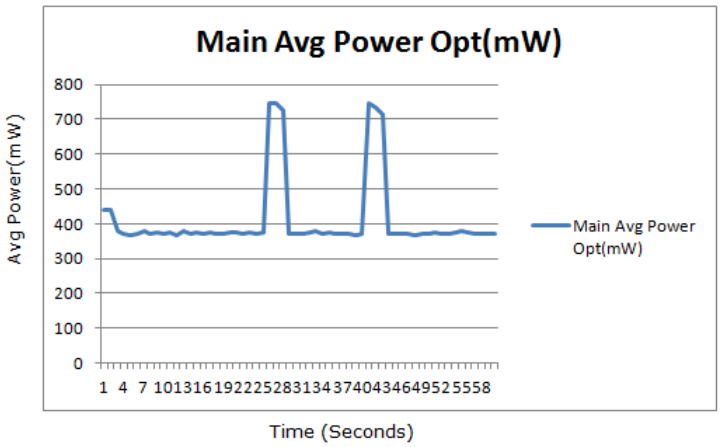

Figure 6.8: Optimized Standby Mode

The following graph shows the comparison of above two scenarios.

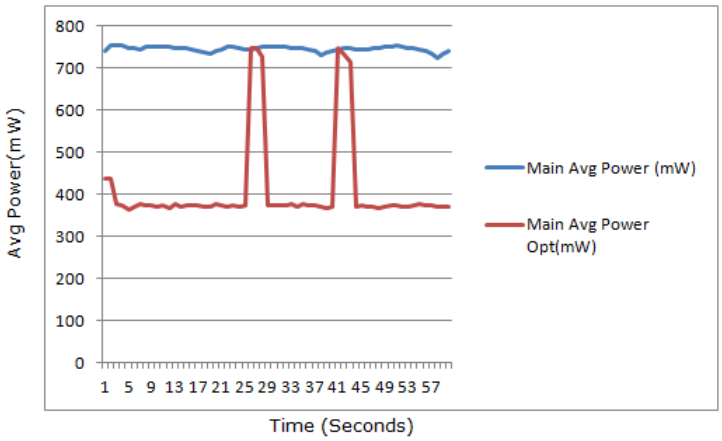

Figure 6.9: Comparison of stand by and optimized standby mode

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

In this dissertation, I proposed two different PM modes to save the power and increased battery life. First I proposed Audio Playback mode in which only required subsystems runs while others are turned off and in other mode when the system is idle, we turn off the systems and save the power and then turn on them when the user becomes active. From the results we can see that for audio playback mode's optimization 2, the overall power consumption becomes almost half then the original power consumption. Original average power consumption for the system per second was 1679mW while with optimization 2 the average power consumption per second becomes 653mW. Same way when system is in idle condition without any optimization the average power consumption per second was 745mW while with optimized standby mode it becomes 411mW per second. Hence I can conclude that if we utilize our system efficiently we can reduce the power consumption of our system which leads to longer battery life without affecting the system performance.

7.2 Future Scope

Current implementation is very specific for the platform and operating system. The next challenge in this work is to make it OS and platform independent. Also some other modes can be created depending on the operations like video playback, internet browsing or text editing.

References

[1] Firmware Architecture and Specification, Intel

[2] SCU Code Study, Intel

[3] Power Management Hardware Archiecture Specification, Intel

[4] Mixed Signal Integrated Circuit Specification, Intel

[5] System Controller Unit HAS, Intel

[6] http://msdn.microsoft.com/en-us/library/aa923906.aspx