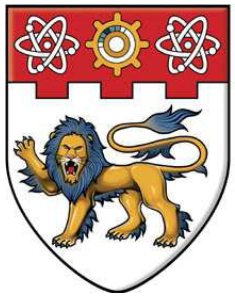


NANYANG TECHNOLOGICAL UNIVERSITY

# RTOS-BASED POWER MANAGEMENT IN EMBEDDED SYSTEMS

**MUHAMED FAUZI BIN ABBAS**

**SCHOOL OF COMPUTER ENGINEERING**

**2011**

# RTOS-BASED POWER MANAGEMENT IN EMBEDDED SYSTEMS

**MUHAMED FAUZI BIN ABBAS**

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

**2011**

# Acknowledgements

I would like to express my gratitude to Professor T. Srikanthan, my supervisor, for his support, guidance and extraordinary patience. Without his understanding approach, I would not have been able to complete this work.

I would like to thank Mr. Mohit Sindhwani, Project Officer, CHiPES, for his regular guidance, patience, and support throughout the duration of the project. I would also like to thank him for his insightful ideas and suggestions regarding the project execution.

I would also like to express my utmost appreciation to Infineon Singapore for sponsoring my tuition fees for this research project.

Last but not least, I would like to thank my wife and family, who have helped me with the difficulties I met, no matter in life or study. They have always supported my pursuit of knowledge, and encouraged me to follow my dreams wherever they take me.

Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **ACPI** | Advanced Configuration and Power Interface specification |
| **OSPM** | OS directed Power Management |
| **OS** | Operating System |
| **RTOS** | Real Time Operating System |
| **PC** | Personal Computer |
| **APM** | Advanced Power Management |
| **BIOS** | Basic Input/Output System |
| **DPM** | Dynamic Power Management |
| **CPU** | Central Processing Unit |
| **ASL** | ACPI Source Language |
| **AML** | ACPI Machine Language |
| **ROM** | Read Only Memory |
| **ACPICA** | ACPI Component Architecture |
| **NRE** | Non Recurring Engineering Costs |
| **API** | Application Programming interface |

# Abstract

Power Management has increasingly become one of the most important features in modern embedded systems. This is in addition to other important constraints such as Non Recurring Engineering (NRE) costs and shorter Time to Market (TTM) due to shrinking product life cycles.

In this research work, a systematic approach to incorporating power management feature into RTOS has been proposed. Comprehensive literature reviews of existing power management techniques that rely on operating systems were performed in an attempt to establish suitable techniques for constraint-aware embedded systems. This led to the detailed examination of the popular method of OS based power management technique known as Advanced Configuration and Power Interface (ACPI).

ACPI was adapted to cater for RTOS based embedded systems and the proposed power management framework was implemented on the Renesas SH7727 using T-Kernel as the target RTOS. It was shown that a centralised power management could be realized using this framework so as to facilitate the rapid incorporation of new devices and integration of new power management policies without necessitating substantial engineering effort. In addition, the added overhead to RTOS is minimal and it can be easily adapted to suit a variety of RTOS.

A systematic technique to deploy power policy algorithms along with RTOS-ACPI has been proposed as part of this work. The proposed technique ensures that it is capable of allowing power policy algorithms to quickly access and to manage the platform specific power features. A technique to improve application-awareness was also evaluated to show that application specific information can be relied upon to identify power-switch points that RTOS-ACPI can depend on to manage power.

The proposed methods pave way for incorporating a comprehensive and scalable RTOS centric power management support into resource constraint-aware embedded systems.

# Chapter 1
# Introduction

## 1.1. Motivation

In a time where energy costs are rising, it is no surprise that industries are clamouring to find ways to reduce cost incurred from energy by researching technology which could aid in its cause. The embedded systems arena has always lagged the desktop computing arena by at least one generation of products. It is popularly said that today's desktop technology are tomorrow's embedded technology.

However, the industry has realized the chance and has caught up with the computer system domain. It is now possible to find technology that would facilitate energy conservation for embedded system. However, this rapid growth in the industry has brought with it a new set of challenges.

In addition, battery technology is progressing at a slow pace; improving by just a few percent each year. Portable devices form factor are also shrinking, implying that the amount of space for batteries is also decreasing, thus the need for the device to consume less power. Unfortunately, consumers are less forgiving and do not accept short battery lifetime of their devices.

Embedded systems are also increasingly getting more complex in an attempt to tackle the different challenges. Nonetheless, as embedded systems became more complicated, the complexity of the underlying hardware has typically been abstracted at higher level by the presence of a software layer called the Real Time Operating System (RTOS). The RTOS abstracts the details of the hardware and presents a standardised interface to the programmers. With the increase in complexity in modern embedded systems, the complexity of the embedded RTOS has increased at the same pace to keep up with the advances.

Another important key observation is that system-wide energy consumption can be effectively managed by an RTOS. Although a large amount of work has

been done in the area of RTOS-based power management, little has been done to take a holistic view to address its pragmatism in embedded systems.

## 1.2. Aim of the Project

The main aim of this research work is to identify and develop novel techniques to effectively manage power while minimizing RTOS overheads in modern embedded systems. The proposed methods target embedded platforms with RTOS support, which enables the framework to monitor and configure the platform with the most effective power strategies at runtime to minimize the power utilization of the embedded platform. To these ends, it is envisioned that a framework for RTOS based power management developed for real time embedded systems can lead to a quick and efficient means of incorporating power strategies that is capable of adapting to the workload characteristics of the application, while meeting the constraints of embedded systems.

## 1.3. Main Contribution of this Thesis

In this thesis, we have proposed a flexible framework for managing power for embedded systems running an RTOS. The following lists the main contributions that have been made during the course of this research, which are documented in this thesis:

1. Refining the ACPI specification and adapting it for embedded systems. The use of ACPI specification are motivated by our investigations which reveal that the specification is comprehensive for typical computing systems but not for embedded systems.

2. Efficient strategies were introduced and incorporated to the framework to reduce overhead on the RTOS. Unlike existing ACPI implementations that required the RTOS to retrieve the information it seeks from various memory sources, our implementation reduces the memory utilization by consolidating all the data into a single database that is managed by the framework.

3. Alternative design and implementation were proposed for the framework. This would allow the proposed framework to be implemented onto various RTOS.

4. The proposed framework abstracts the power services of the embedded platform to the software developer. This feature brings flexibility to the framework that would allow generic power savings algorithms and application-specific power strategies to be implemented onto the proposed framework quickly.

5. A novel method to describe the power features of the embedded platform in script language that would be translated into a hierarchical structured database by the proposed framework. This database is managed by the framework and contains the entire data for the platform's power features.

## 1.4. Organization of this Thesis

The next chapter of the report presents the background to the project. The current technologies in managing power for embedded systems is identified and presented. The chapter identifies prior work that has been done in power management and briefly looks at the current power strategies being embedded in RTOS research and looks at previous efforts to introduce certain power management features to RTOS. It shall also look into existing power management techniques and strategies in some of the popular RTOS, like VxWorks, Android and Linux. This chapter shall propose the motivation of this research.

Chapter 3 would review ACPICA and extract salient design that could be implemented in the RTOS-ACPI. It shall also discuss various design consideration for an embedded system environment and identify the extent of its context.

Chapter 4 shall investigate power management policies and propose power management algorithms that would augment the power management framework. A special power management strategy is discussed that does offline application profiling which extract salient information that would be fed into the power policy to manage the system's power.

Chapter 5 concludes the report and highlights what have been discussed in the report.

# Chapter 2
# Literature Review

## 2.1 Power Management

Power consumption of electronic devices has become a serious concern in recent years. Power management that can lengthen the battery time in the portable systems while delivering acceptable performance is crucial to embedded application domain. The ability to extend the running time of battery powered device through software has been a prevalent topic that has led to many branches of research and implementations. In addition, power management are essential to reducing operational cost and lowering heat dissipation, which increases system stability and makes the end product more affordable. The demand for better battery life of new portable devices has led to novel approaches in tackling power management. The definition of power management has expanded to include power distribution, power delivery and power consumption. Currently, there are two common techniques of power reduction, namely, static and dynamic.

Power management basically fall into 2 main category; static power management and dynamic power management. Each has its advantages over the other and has the capability to provide the best performance depending on the type of application. Although static techniques can save significant energy, they are relatively inflexible to adapt to changes in the environment. Dynamic power management at the OS level, on the other hand, is flexible, easy to use, and compatible with different applications.

***Dynamic Power Management***

Dynamic power management (DPM) is a powerful methodology for reducing power consumption in electronic systems. The techniques leverage on the runtime characteristics of the application to reduce power when systems are serving light workloads or are idle. The primary focus of DPM is to reduce power consumption without incurring loss of performance. The power manager

monitors the workload of the system and dynamically makes power-saving decision in real-time to best minimize the system power consumption. DPM research efforts have typically been targeted to investigate a particular strategy or optimization. However, the term 'dynamic power management' have been abused to describe any features that reduce the power consumption. This report will define DPM as the ability for the system to adapt to the behaviours of the application and take intelligent steps to reduce the power consumption.

***Static Power Management***

Static techniques such as synthesis and compilation for low power are applied at design time. This technique could tailor solution for application on specific situations which would not work well for other situations. This is due to its computation is done offline which would not be able to adapt to changes during runtime. Applications are analyzed by an offline profiler, which will extract the application's intrinsic characteristics. A power manager will make the decision to manage the device power state according to the information obtained.

## 2.2 Current Research Trends on Managing Power

The broad range of research on power management can extend many engineering domains as mentioned earlier. It would not be possible to list down a exhaustive list of research in this area as new ideas adds on to the list. However, listed below are the researches done by distinguish academia and have attained acceptance status.

### 2.2.1 Circuit Level Power Management

***Interface Power Minimization***

The power dissipation at the off-chip bus is a significant part of the overall power dissipation in digital systems. Therefore, the minimization of the switching activity at the I/O interfaces can provide significant savings on the overall power consumption. An innovative encoding technique used to minimize the switching activity of system-level address buses. The schemes target the reduction of the average number of bus line transitions per clock cycle [Beni98a]. Another approach is for the efficient generation of an encoder to

minimize switching activity on the high-capacity lines of a communication bus. This approach is a static one in the sense that the encoder is realized ad hoc according to the traffic on the bus.

Typically, an embedded system executes the same application throughout its lifetime and so it is possible to have detailed knowledge of the trace of the patterns transmitted on a bus following execution of a specific application. The approach is compared with the most efficient encoding schemes proposed in the literature on both multiplexed and separate buses [Gius03].

***Optimising Flip-Flops for Low Power***

Flip-flops that use new gating techniques that reduce power dissipation deactivating the clock signal. Presented circuits overcome the clock duty-cycle limitation of previously reported gated flip-flops. Circuit simulations with the inclusion of parasitic show that sensible power dissipation reduction is possible if input signal has reduced switching activity [Stro00]. Another technique make use of double edge triggered flip-flops which results in a power reduction of 50% in the clock net, and in a reduction of up to 45% inside the flip-flops. The investigation considered other flip-flop parameters, like setup and hold times, propagation delay and testability [Rafa96].

### 2.2.2 Multi-Processors

***Multi-core Processors***

Power management of multi-core processors is extremely important because it allows power/energy savings when all cores are not used. While operating systems are capable of power management, heuristics for effectively managing the power are still evolving. The granularity at which the cores are slowed down/turned off should be designed considering the phase behaviour of the workloads. The effects of the idle core frequency on the performance and power of the active cores are investigated and identifying the idle core frequency to have the least detrimental effect on the active core performance [Birc08].

***Heterogeneous Multiprocessors***

As single-chip systems are increasingly composed of heterogeneous multi-processors; an opportunity exists to explore new levels of low-power design. When a system executes a variety of different tasks sets, the problem becomes one of establishing the cost and benefit of matching task types to processor types under anticipated task loads on the system. This includes not only static task mapping, but dynamic scheduling decisions as well as the selection of the most appropriate set of processors for the systems. Different models were considered to find an appropriate system-level power-performance trade-offs and introduce some design strategies to tackle the problem [Paul07].

***Multithreaded Processor Cores***

A new hardware-based power management technique that is made possible by a multithreaded processor core. A processor-internal scheduler manages frequency and voltage scaling based on the current processor utilization given in percentage of the total performance [Uhri04].

### 2.2.3 Application Level Power Management

***Synthesis of Instruction Sets***

*A*n energy-efficient instruction set synthesis that can comprehensively reduce the energy-delay product of Application-Specific Instruction set Processors through optimal instruction encoding, considering both the instruction bit width and the dynamic instruction fetch count. A typical embedded RISC processor shows that the proposed energy-efficient instruction set synthesis technique can generate application-specific instruction sets that are more energy-efficient over the native instruction set for several application benchmarks [Jong07].

***Code Compression for Embedded Systems***

Compression has been utilized in electronic systems to improve performance, reduce transmission costs, and to minimize code size. Two techniques to compress instructions are discussed. The first technique compresses instruction traces, so that the compressed trace can be used to explore the best cache configuration to be used in an embedded system. Trace compression enables rapid cache space exploration. The second technique uses compressed

instruction in memory, to be expanded just before execution in the processor. This enables a smaller code footprint, and reduced power consumption. Benefits of the two orthogonal approaches to the design of an embedded system are looked at [Para07].

***Application controlled power management***

Some techniques [Heal04] [Luyh00c] has much better potential for reducing energy consumption. However, it places the burden of inserting power management directives on the programmers and requires the existing applications to be modified.

***I/O Operations***

Technique that look at correlating I/O operations to program behaviour that is employed in [Chri06]. At the register level, Program Counter Access Predictor is a technique that dynamically learns the access patterns of applications using the program counter and predicts when a peripheral can be switched to a low-power mode to save energy. When long idle period is detected, the program counters following the last I/O operation are recorded to be used to identify future occurrences of these program behaviours.

***Application Specificity***

There is various techniques and policies that are tailored to multimedia applications running on such battery-operated portable devices [Sama08]. While other targeted at network domain and are scalable as the system grows. They tend to be flexible to video parameters and network characteristics [Nich07].

### 2.2.4 Compiler Level Power Management

***Divide-and-Conquer Techniques***

A novel divide-and-conquer compilation technique to minimize the number of operations for general computations. Investigate coordinated impact of compilation techniques on the number of processors that provide optimal trade-off between cost and power. It is demonstrated that proper compilation techniques can significantly reduce power with bounded hardware cost [Hong99].

***Energy Estimation Compilation***

A novel Energy-Aware Compilation (EAC) framework that estimates and optimizes energy consumption of a given code, taking as input the architectural and technological parameters, energy models, and energy/performance/code size constraints. The EAC allows compiler writers and system designers to investigate power-performance tradeoffs of traditional compiler optimizations and to develop energy-conscious high-level code transformations [Kada05].

***Memory Optimisation Techniques***

A technique for reducing memory energy consumption is memory banking. The idea is to divide the memory space into multiple banks and place currently idle banks into a low-power mode. However, these techniques do not take the data cache behaviours explicitly into consideration. As a consequence, the energy savings achieved by these techniques can be unpredictable due to dynamic cache behaviour at runtime. Introducing a bank-aware cache miss clustering compiler optimization that increases idle durations of memory banks. This would enable better utilisation of available low-power capabilities supported by the memory system. This is to take advantage of the clustering cache misses which helps to cluster cache hits as well which in turn increases bank idleness [Oztu06].

***Instruction-Level Power Optimization***

The power consumption of an embedded application can be reduced by moving as much computation as possible from runtime to compile time. The compiler can look at the entire program and hence has a much larger observation window than what can be achieve by the developer. Each instruction has a corresponding power cost and by using the compiler to choose minimum-power instruction mix would lead to power savings. However, there might be a trade-off of slower performance.

### 2.2.5 Low Power Operating System Strategies

***Power-Aware Scheduling***

Next generation of embedded systems must be power-aware and not just low-power. They should be able to track their power sources and the changing power and performance constraints imposed on the system and schedule the task to get the best performance-power trade-off. [Jinf01].

***Dynamic Variation and Frequency Scaling (DVFS) Scheduling***

Power-aware task scheduling algorithm for DVS-enabled real-time multiprocessor systems, unlike the existing algorithms, can handle conditional task graphs which model more complex precedence constraints [Dong 03].

***Low Power Idle-time Scheduling***

This algorithm saves power by shutting down idle devices which often serve requests from concurrently running tasks. Ordering task execution can adjust the lengths of idle periods and exploit better opportunities for power management [Yung00].

*Process based (user-defined) power management* which makes distinction on the sources of requests has also been proposed [Luyh00b].

***A low power scheduler using game theory***

A new methodology based on game theory for minimizing the average power of a circuit during scheduling in behavioural synthesis. The problem of scheduling in data-path synthesis is formulated as an auction based non-cooperative finite game, for which solutions are developed based on the Nash equilibrium function. Each operation in the data-path is modelled as a player bidding for executing an operation in the given control cycle, with the estimated power consumption as the bid. Also, a combined scheduling and binding algorithm is developed using a similar approach in which the two tasks are modelled together such that the Nash equilibrium function needs to be applied only once to accomplish both the scheduling and binding tasks together. The combined algorithm yields further power reduction due to additional savings during binding. The proposed algorithms yield better power reduction than ILP-based methods with comparable run times and no increase in area overhead.

### 2.2.6 Power Policy Algorithms

Power management policy is an important part of power management framework, as the performance depends on the ability of the policy in managing power for that particular application running on the system. DPM algorithms can be broadly classified into three major categories: *timeout*, *predictive*, and *stochastic*.

***Timeout policy***

This policy is the most widely used policy in industry. The time-out policy maintains a set of suitable time-out values based on earlier analysis. Because of its simplicity, this policy is implemented in many areas, although its power saving performance is not satisfactory, and delay penalty will be introduced. In most real-world systems, the workload is not stable, that means using fixed timeout value is unreasonable and insufficient. However, dynamic variations of this policy allow the time-out interval to be dynamically adjusted. Predictive policy calculates the estimated length of an idle period based on earlier runtime information. Applying non-linear regression equations and adaptive learning tree are some of the popular techniques for this policy.

***Predictive policy***

*Predictive* policies are developed to dynamically adjust the timeout value depending on previous idle period history. Usually the workload's variety has some regular patterns. A good policy can find these patterns and making intelligent decision according to the patterns. To decrease the delay penalty, people also propose predictive wake up policy, try to hold high real-time capability.

In most real-world systems there is little knowledge of future input events and DPM decisions have to be taken based on uncertain predictions. The rationale in all predictive techniques is that of exploiting the correlation between the past history of the workload and it's near future in order to make reliable predictions about future events. Good predictors should minimize the number of wrong predictions. The most common predictive PM policy is the *fixed timeout*, which uses the elapsed idle time as observed event to predict the total duration of the

current idle period. The policy can be summarized as follows: when an idle period begins, a timer is started with duration $T_{to}$. If after $T_{to}$ the system is still idle, then the PM forces the transition to the 'off' state. The system remains off until it receives a request from the environment that signals the end of the idle period. Timeouts have two main advantages: they are general (their applicability slightly depends on the workload) and their safety can be improved simply by increasing the timeout values). Unfortunately, they trade-off efficiency for safety: large timeouts cause a large number of under-predictions, that represent missed opportunity of saving power, and a sizeable amount of power is wasted waiting for the timeout to expire.

*Predictive shut-down Policies* [Gold96] [Sriv96] improve upon timeouts by taking decisions as soon as a new idle period starts, based on the observation of past idle and busy periods. The DPM strategy proposed by Hwang et al. [Hwan97] addresses another limitation of timeout policies, namely the performance penalty that is always paid on wakeup. To reduce this cost, the power manager performs predictive wakeup when the predicted idle time expires, even if no new requests have arrived. This choice may increase power dissipation if idle time has been under-predicted, but decreases the delay for servicing the first incoming request after an idle period.

Since the optimality of DPM strategies depends on the workload statistics, static predictive techniques are all ineffective when the workload is either unknown, or non-stationary. Hence, some form of adaptation is required. While for timeouts the only parameter to be adjusted is the timer duration, for history-based predictors even the type of observed events could in principle be adapted to the workload. Several *adaptive predictive techniques* have been proposed to deal with non-stationary workloads. In the work by Krishnan et al [Kris95] a set of timeout values is maintained and each timeout is associated with an index indicating how successful it would have been. The policy chooses, at each idle time, the timeout that would have performed best among the set of available ones. Another policy, presented by Helmbold et al. [Helm96], also keeps a list of candidate timeouts, and assigns a weight to each timeout based on how well it would have performed relatively to an optimum off-line strategy for past

requests. The actual timeout is obtained as a weighted average of all candidates with their weights. Another approach introduced by Douglis et al. [Doug95] is to keep only one timeout value and to increase it when it is causing too many shutdowns. The timeout is decreased when more shutdowns can be tolerated. Several predictive policies are surveyed and classified in Douglis' paper.

***Stochastic policy***

A common strategy to system-wide approach is to model the entire system into mathematical equations and solve them using *stochastic* process. Stochastic policy model the arrival of requests and device power-state changes as states. These approaches formulate policies as optimization problems under uncertainty rather than trying to eliminate uncertainty by predictions.

Those policies usually model the general systems and user request as Markov chains. The Markov model enables a rigorous formulation of the search for optimal power management policies. Minimizing power consumption is then a stochastic optimization problem and obtaining the optimal probability to shut down an unused peripheral can be solved using discrete-time Markov processes [Beni99]. There are recent variations that extend this method and considered non-stationary accesses [Govi95]. Event-triggering was made possible by employing continuous-time Markov models [Qiup99]. Another approach would be Time-indexed semi-Markov decision process [Pute94].

## 2.3 OS-based Power Management Strategies

In this section, we look at some of other frameworks that are coupled to an OS. Reviewing these strategies would help get a broad view of the available methodologies, tools, and philosophies currently available in power management. OS is regarded as the best entity to manage power as it is informed of the system's state, such as device usage, CPU load, etc. The following are some of power management strategies for computer system OS and embedded system RTOS.

### 2.3.1 QNX Neutrino's Power Management Architecture

The QNX Neutrino is an open source microkernel-based OS which has the framework allows developers to fully exploit the power features of any embedded hardware. The source release includes the code to microkernel, the base C library, and a variety of board support packages for popular embedded and computing hardware.

QNX Neutrino's power management architecture has been designed to allow developers to build a fully customized power management solution specifically for their target hardware. The architecture makes no assumptions regarding any specific power management standards, but instead allows developers to have full control over the power resources of their system. The figure below shows an overview of the architecture, consisting of a server library, client library, and a power callout.

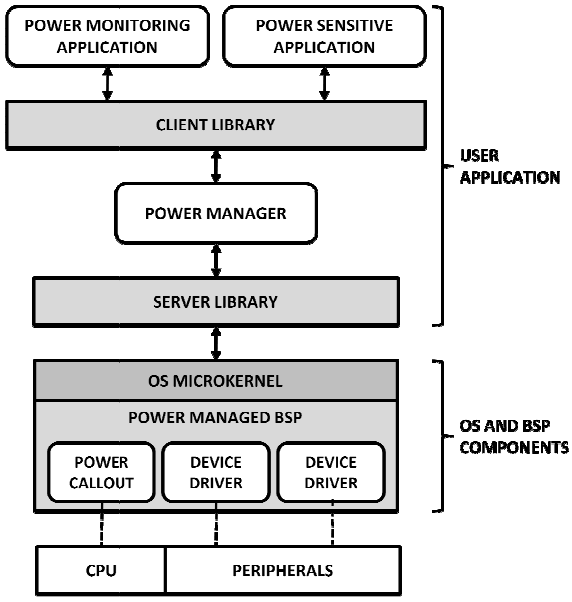



*Figure 1: QNX power management framework*

The server library provides the building blocks used to create a custom power manager. The power manager runs as a standard application in user space and manages all power aware devices in the system including device drivers and the CPU. The server library allows developers to create a state machine in their power manager in order to implement a custom power policy for the target system.

The client library provides the mechanisms to allow drivers and other applications to interface to the system's power manager. It is through this client interface that the power manager can control the power mode of individual device drivers. The client library interface can also allow monitoring applications the ability to pass system information to the power manager. In addition, applications that may be sensitive to power state changes can receive notifications from the power manager.

QNX architecture would also allow developers to write their own power callout to control the power state of the CPU itself. This custom callout is accessible via a standard kernel function call, allowing the power manager to take advantage of the CPU's specific power modes or voltage/frequency scaling capabilities. [SEPM09]

### 2.3.2 VxWorks Power Management

VxWorks is a real time OS, developed by Wind River Systems until recently, when Intel bought Wind River Systems in June 2009. It is a popular RTOS that is being used in safety critical, hard real time systems such as space missions and aircrafts.

The power management in VxWorks is very much aligned to the CPU being used. VxWorks allows harnessing the power management capabilities offered by the specific CPUs. Hence the power management interface is not uniform.

The power management framework of the VxWorks RTOS is tightly woven around the CPU used. The OS, by itself does not provide power management capabilities, it provides libraries which can be used to access the CPU’s power management features. The system components other than CPU do not have a mechanism to be controlled in a standard manner, so power management in VxWorks is very simplistic in this sense. Any more sophisticated the power management tends to be, heavy additional custom support needs to be added in.

### 2.3.3 Google Android Power Management

Android is an open source embedded OS developed by Google for use in mobile products. It was first released in 2007, and since been adopted in various intelligent mobile devices. Power Management in Android is a module in its Linux Kernel Layer which augments the power management capability of Linux [Goog08].

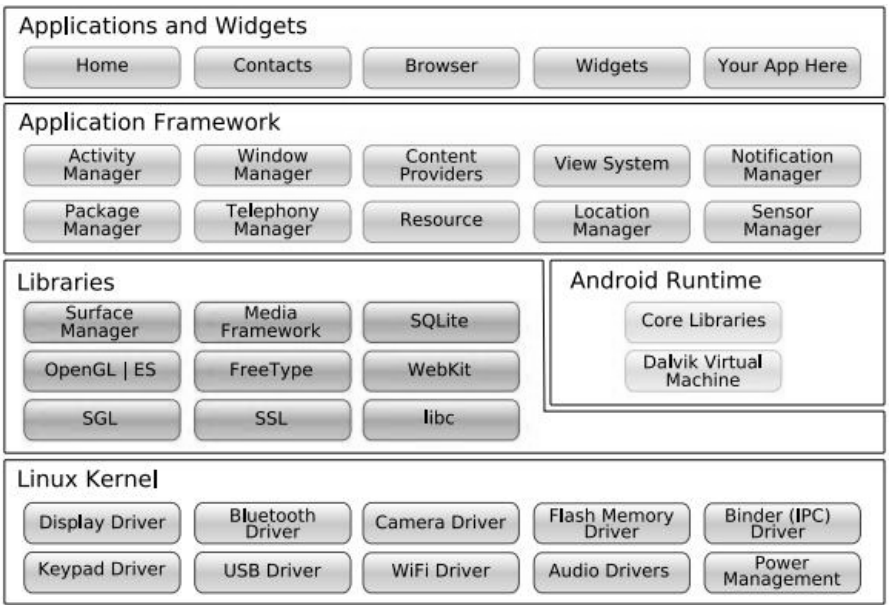



*Figure 2: Android System Architecture*

Android implements a very simple power management mechanism. It uses wake locks which exposes the platform's power management to the developer. Wake locks are used by applications and services to request CPU resources. As long as a resource is on active wake lock, Android keeps the resource powered up. When the application releases the wake lock on the resource, and there are no pending wake lock holding the device, Android would powers down the device. Even the CPU is recognized as a resource and requires wake lock request through the Android application framework and native Linux libraries. If there are no active wake locks, Android will shut down the CPU.

Alternative way of implementing wake is user activity which keeps the application processing part of android alive for as long as the user activity is detected within its timer's expiry. It could use more than one timer value for transitioning through various states.

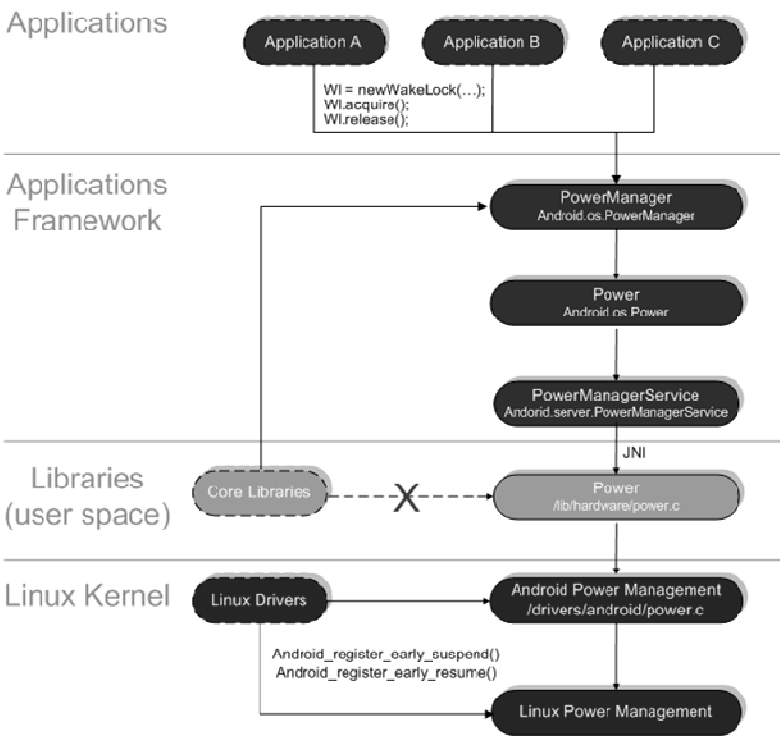



*Figure 3: Android Power Management Anatomy*

Linux power management code is driver-specific. There are two models for device power management:

1. System sleep: a system-wide low power state.
2. Runtime power management: various drivers can enter individual low-power states while everything else is still running.

The actual implementation of these states is system or device specific. So, there is an open end for the developers who require that the system be rigorously power managed.

### 2.3.4 Advanced Power Management (APM)

APM technology was first introduced in 1992. It was started by Intel, but was joined by Microsoft and IBM to bring up a unified approach to power management in the computer industry.

The basic working principle of APM is to control the power consumption of the system by monitoring the system activity and adjusting the power states or the power resources accordingly. As system activity decreases, APM reduces power

to unused system resources until the system is brought into a suspend state. [INMI96]

APM uses a layered approach to manage devices. APM-aware applications and APKM-aware device drivers interact with the OS-specific APM driver. This driver communicates to the APM-aware BIOS, which controls the hardware. APM can be said to be an OS-participative, BIOS-centered approach to power management. The APM architecture as illustrated by the APM specification can be found below [INMI96]:

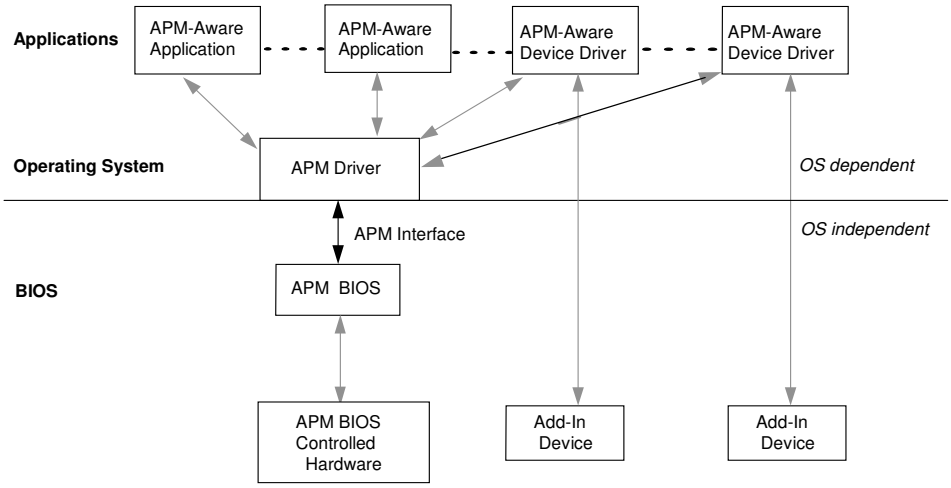



*Figure 4: Advanced Power Management Architecture*

***Disadvantages of APM***

APM started out and evolved as the de-facto power management standard that is adopted by most major industries. However, as it evolved many limitations surfaced and led to cropped up and led to difficulties. The next generation of OS power management (ACPI), came about due to the following challenges:

1. As stated by [PGSS02], the principle limitation of APM is that the OS has no knowledge of the APM actions. The power management functions were executed by the APM BIOS in the real time BIOS calls which can have an effect on the stability of the system.
2. Every platform has its own BIOS, specific to its hardware. This resulted in varied user experience for the same version of APM on different platforms.

3. APM was inflexible and lack support for sophisticated user requirements. As stated by [GROV03], APM's battery status interface could only aggregate the battery information, possibly from multiple batteries and report a single parameter, "minutes remaining".

### 2.3.5 Advanced Configuration and Power Interfaces (ACPI)

ACPI stands for Advanced Configuration and Power Interfaces, and is a popular power management framework used in computer systems. ACPI evolved from APM as a result for the need of a better power management technique.

Notable enhancement of ACPI:

1. OS has control of power management as opposed to the BIOS in APM. This leads to less frequent calls to BIOS which improves system stability.
2. Standardises the description of power management capabilities which facilitates the OS to manage power of the entire system.
3. Enable all computer systems to implement motherboard configuration and power management functions, using appropriate cost/function tradeoffs.
   This goal is commensurate with the purpose of ACPI itself. An important feature in this scheme is that ACPI is responsible for device configurations as well, in addition to power management.
4. Enhance power management functionality and robustness.
   BIOS code was limited in many aspects, one of which is memory. BIOS' lack of a wholesome overview of the system information, especially those that are not hardware-related, like the status of the application software, User preference or the OS design, does have disadvantages. Hence, ACPI denigrates the role of BIOS in power management.
5. Facilitate and accelerate industry-wide implementation of power management.
   Avoiding redundant research on power management across the computer industry, ACPI is developed and maintained by the collective contribution from the industry. This has helped in OS and the platform hardware evolving independent of each other.
6. Create a robust interface for configuring motherboard devices.

### *2.3.5.1 Power States in ACPI*

Before looking at the specifics of ACPI, it is beneficial to be aware of the basics of ACPI. Power states are various levels of power consumption defined in various granularities, for the system or for the individual devices of the system.

With a whole gamut of power states to choose from, the onus is on the power management policy to determine the optimal power state for the system. The following figure illustrates the various power states that are defined in the ACPI specifications.

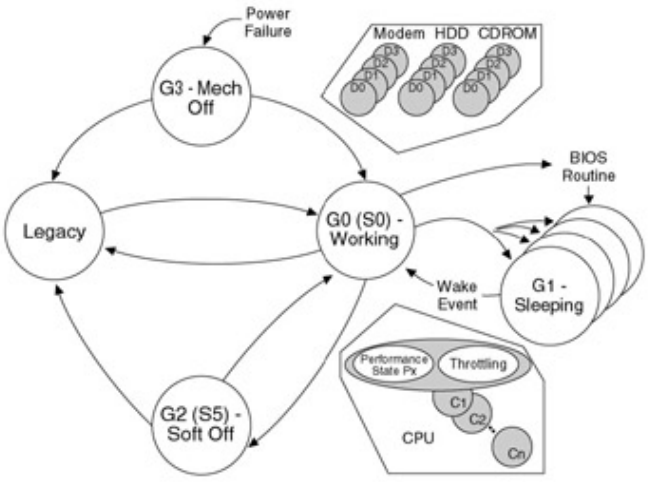



*Figure 5: ACPI Power States*

***Global System States***

Global system states are the power states that correspond to the entire system. The global system states ranges from state G0 to G3 [HIMP06]. The intermediate states G1 and G2 include various granularities of sleep states. G2 state consumes lesser power than G1 state. Hence the wake-up latency has a reverse relationship that increases from G2 to G1 state. Power consumption at G3 state is zero, excluding only the real-time clock.

*Table 1: Global System States*

| Global system state | Software runs | Latency | Power consumption | OS restart required | Safe to disassemble computer | Exit state electronically |
|---|---|---|---|---|---|---|
| G0 Working | Yes | 0 | Large | No | No | Yes |
| G1 Sleeping | No | > 0, varies with sleep state | Smaller | No | No | Yes |
| G2/S5 Soft Off | No | Long | Very near 0 | Yes | No | Yes |
| G3 Mechanical Off | No | Long | RTC battery | Yes | Yes | No |

***Device Power States***

Device power states are used to define the 4 different power modes for every peripheral in the system. Device power consumption, retention of device context, device driver responsibility and restore latency are criteria for defining the various device power states.

*Table 2: Device Power States*

| Device state | Power consumption | Device context retained | Driver restoration |
|---|---|---|---|
| D0 - Fully On | As needed for operation | All | None |
| D1 | D0 > D1 > D2 > D3 | > D2 | < D2 |
| D2 | D0 > D1 > D2 > D3 | < D1 | > D1 |
| D3 - Off | 0 | None | Full initialization and load |

***System Sleep States***

Sleep states are progressive levels of sleep defined in the Global system state G1. The following summarises the 5 levels of sleep states ranging from S1 to S5.

*Table 3: System Sleep States*

| Sleep States | Wake latency | System context retained | Note |
|---|---|---|---|
| S1 | Short | All | Hardware maintains the entire system context |
| S2 | Short | All, *except CPU & system cache* | OS is responsible for maintaining caches & CPU context. Control starts from the processor's reset vector after the wake event. |
| S3 | Short | System memory, some CPU and L2 configuration only | Hardware maintains memory context and restores some CPU and L2 configuration context. Control starts from the processor's reset vector after the wake event. |
| S4 | Longest | All | Minimum power consumption. Hardware platform powered off all devices. |
| S5 | Reboot | None | Used for initial boot operations to distinguish from S4. |

***Processor Power States***

These are processor power consumption states that are defined with the G0 global system state that ranges from C0 to C3. C0 state is the working state of the CPU when it executes instructions while C1 through C2 states progressively increases in wake-up latency, processor's disability to execute instructions and power savings. There can be various combinations of operating voltage and frequency which give rise to a new set of processor performance levels.

#### *2.3.5.2 ACPI Architecture*

ACPI is primarily an interface specification that defines the hardware and software interfaces and data structures of a system in order to implement OS directed power management in an efficient manner. As shown in the figure below, the ACPI architecture comprises of 3 main components; ACPI BIOS, ACPI Tables and ACPI Registers.

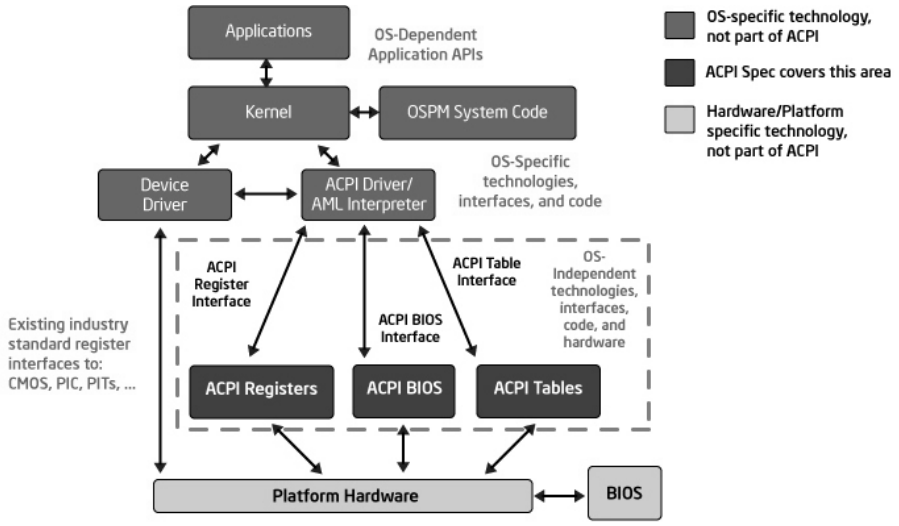



*Figure 6: ACPI Global System Architecture*

The application is the user code that runs on top of the kernel which is at the heart of the OS, supported by various subsystems and modules that make up the system. OSPM module is the intelligent component of power management which defines power policies, monitors the system, and calculates the optimal power management decisions. ACPI driver/AML interpreter located in the OS interacts with the ACPI defined components; ACPI BIOS, tables and registers. The ACPI specification does not dictate on the implementation of these components, they are left to the system developers.

***ACPI BIOS***

BIOS is a boot firmware, designed to be the initial function run by a computer to identify, test, and initialize system hardware when powered on. This process is known as booting and is to prepare the machine into a known state, so that software stored on compatible media can be loaded, executed, and given control of the computer. Apart from the BIOS, it is common to have BIOS extensions for specific device classes like video display adapters and hard-driver controllers. Similarly, ACPI specifies an ACPI-BIOS whose functionality is to boot the system as well as support OSPM. Its primary function is to supply the platform information to the independent OS that is unaware of the particular system's hardware. It is the platform hardware developer's responsibility to provide the hardware details to the OS through the ACPI-BIOS in the form of

ACPI tables. In contrast to the APM, the ACPI-BIOS is not called frequently and its executive capability is minimum and essential.

***ACPI Tables***

ACPI tables are at the heart of ACPI implementation, which provide information about the platform hardware to the ACPI subsystem. They are organised in tables and stored at various locations in the ACPI-BIOS.

ACPI tables are a vital components of an ACPI implementation as they describe the interfaces to the hardware. These descriptions allow the hardware to be built in various ways and can describe arbitrary operation sequences needed to make the hardware function.

ACPI Tables containing "*Definition Blocks*" can make use of a pseudo code type of language, the interpretation of which is performed by the ACPI driver part of the OS. That is to say that OSPM uses an interpreter that executes procedures encoded in the pseudo-code language and stored in the ACPI tables containing "*Definition Blocks*". The pseudo code language, known as ACPI Machine Language (AML), is a compact, tokenized, abstract type of machine language.[Himp06]

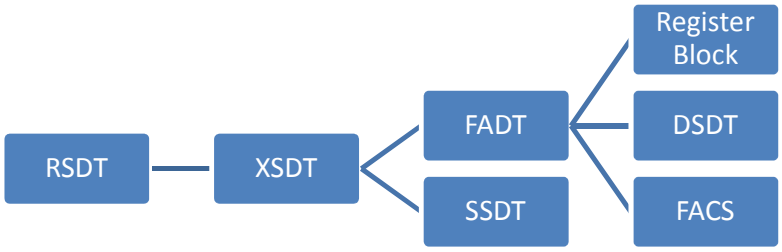



*Figure 7: ACPI Table Structure*

The following describes how the relevant information is extracted from the ACPI tables:

- The OS provides the tables to the ACPI subsystem. It provides the Root System Description Pointer (RSDP) to the ACPI subsystem, from where all the tables are loaded.

- RSDP points to Extended System Description Table (XSDT). This table in turn points to other definition tables.
- Typically XSDT points to Fixed ACPI Description Table (FADT).The FADT defines system information related to configuration and power management, such as the hardware register blocks on the platform.
- FADT point to Fixed ACPI Control Structure (FACS), which contains critical data like Global lock for synchronized access of shared resources between ACPI and external systems.
- FADT also points to Differentiated system Description Table (DSDT). This table contains a Differentiated definition block which contains information that OSPM can utilize to perform power management. It contains implementation details in the form of data as well as control methods in a hierarchical form.
- DSDT may be supplemented by Secondary System Description Tables (SSDT), which contain addition information and pointed to by XSDT. This is the table where all the devices will be enumerated and almost all relevant information for the OSPM will have to be implemented.

***ACPI Registers***

This is the part of the hardware interfaces as specified by ACPI. In order to make hardware platforms compatible with ACPI, they need to be built using these specs as the design guidelines. They are described in the ACPI tables for the OSPM to access them and perform power management functions.

The details are provided by the ACPI-BIOS in the form of tables which are coded in ACPI Source Language (ASL), and compiled into ACPI Machine Language (AML) code. ASL and AML are significant concepts in ACPI that reduced the OS's reliance on the firmware and the firmware to communicate to the OS the necessary steps to perform actions on the platform, but gives OS the responsible for executing them

ASL provides constructs to describe registers and the methods to access them. ASL is a comprehensible language that includes arithmetic, logical and branching instructs. A compiler needs to be developed for each machine to

convert ASL to AML. AML code which is machine-specific is then interpreted by the OS at run-time when necessary.

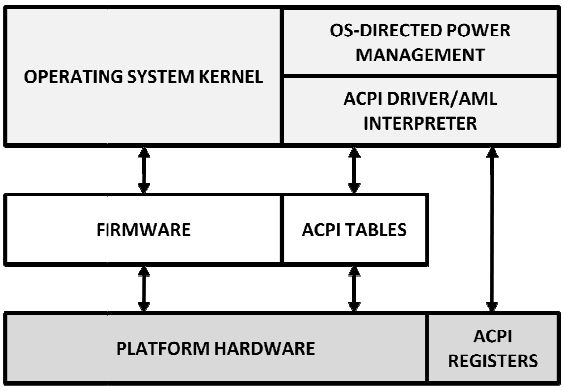



*Figure 8: AML interaction with the OSPM and Platform Hardware*

#### *2.3.5.3 ACPI Reference Implementation - ACPICA*

ACPI has been implemented for many desktop-based operating systems and their derivatives, including Windows, Linux, BSD, etc. In addition, there is an OS independent reference design called ACPI Component Architecture (ACPICA) which would be used to extract possible design for embedded systems implementation. It implements a set of software components that make up an implementation of ACPI. The complexity of ACPI specification makes it difficult to implement in an OS. The purpose of the ACPICA is to simplify ACPI implementations for operating system vendors (OSV) by providing major portions of an ACPI implementation in OS-independent ACPI modules that can be easily integrated into any OS. [Inte03]

***ACPICA Architecture***

ACPICA can be easily adapted to any host OS and is meant to be directly integrated into the host OS. However, it requires a small OS-specific interface layer (OSL), which must be written specifically for each host OS. OSL is the OS dependent portion which interfaces between the ACPI core subsystem and the OS. The abstraction it provides makes it possible for the ACPI Core subsystem to be OS independent.

The following figure illustrates the interactions among components in the ACPICA architecture.

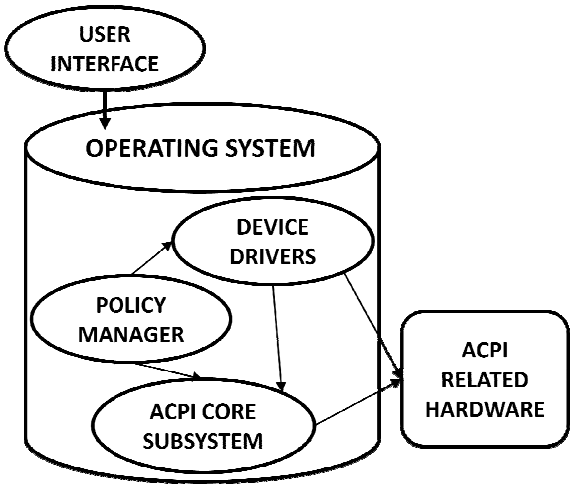



*Figure 9: ACPICA Architecture*

As seen in the figure below [INTE03], ACPI core subsystem provides the core ACPI services which include *ACPI table management*, *namespace management*, *resource management*, *ACPI hardware management* and *event management*.

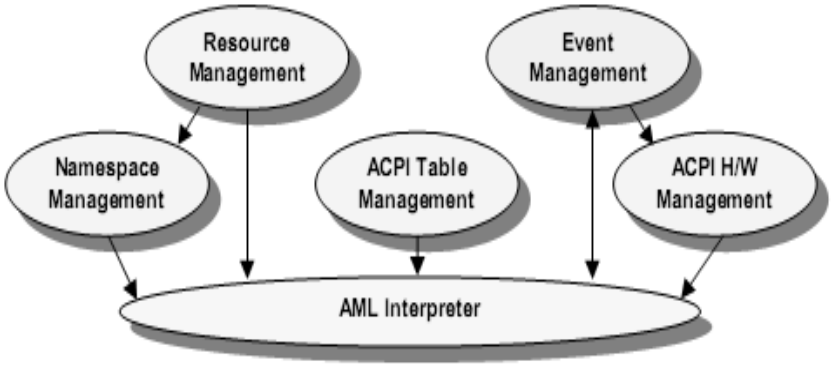



*Figure 10: Internal Modules of the ACPICA Core Subsystem*

***ACPI Table Management***

This component manages all ACPI tables such as the RSDT/XSDT, FADT, FACS, DSDT, SSDT, etc. The tables may be loaded from the firmware or directly from a buffer (typically from BIOS) provided by the host operating system. Services include *ACPI table verification*, *ACPI table installation & removal* and *access to all available ACPI tables*.

***AML Interpreter***

The AML interpreter is responsible for the parsing and execution of the AML byte code that is provided by the computer system vendor. Most of the other services are built upon the AML interpreter. Therefore, there are no direct external interfaces to the interpreter. The services that the interpreter provides to the other services include *ACPI table parsing*, *AML control method execution* and *evaluation of namespace objects*.

***Namespace Management***

The Namespace component provides ACPI namespace services on top of the AML interpreter. It builds and manages the internal ACPI namespace. Services include *namespace initialization from ACPI tables*, *device enumeration*, *namespace access* and *access to ACPI data & tables*.

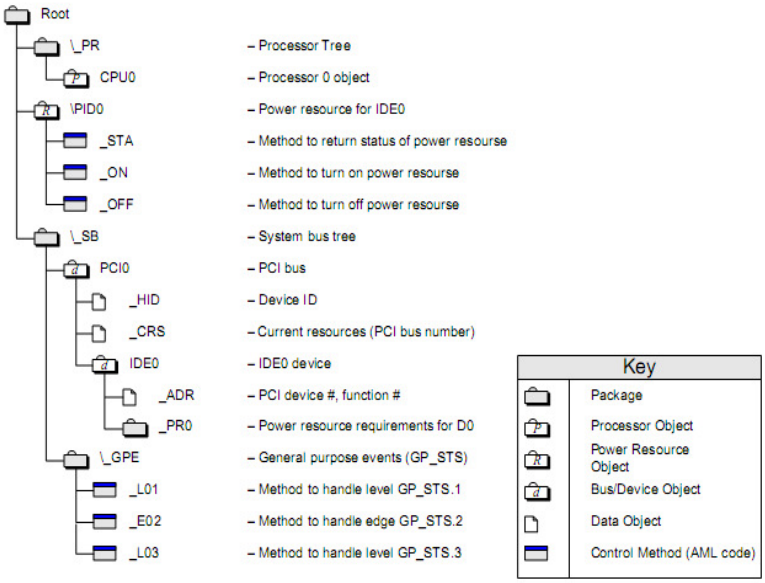



*Figure 11: Example of ACPI Namespace*

***Resource Management***

The Resource component provides resource query and configuration services on top of the Namespace manager and AML interpreter. Services include *obtaining possible resources*, *IRQ routing tables* and *power dependencies*. It also provides services to *obtain and set current resources*.

***ACPI Hardware Management***

The hardware manager controls access to the ACPI registers, timers, and other ACPI-related hardware. Services include *ACPI status register & enable register access*, *ACPI register access (generic read and write)*, *power management timer access*, *ACPI mode enable/disable*, *global lock support* and *sleep transitions support (S-states).*

***Event Management***

The component manages the ACPI system control interrupt (SCI). The single SCI multiplexes the ACPI timer, Fixed Events, and General Purpose Events (GPEs). This component provides *fixed event handlers*, *GPE handlers*, *notify handlers* and *address space & operation region handlers*, installation, removal and dispatch services.

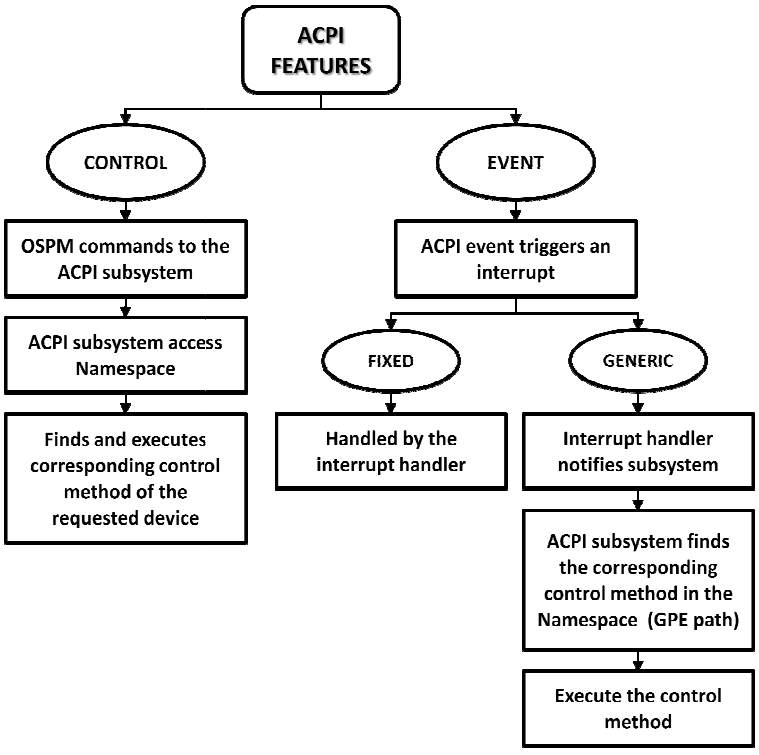



*Figure 12: ACPI Control Flow & Event Flow*

Fixed events are those which are the static part of the ACPI implementation necessary for the working of the OSPM as specified. ACPI defines power management timer, console button (power and sleep button), sleep and wake

control, real time clock alarm, ACPI select & SCI interrupt and processor control as fixed events.

Generic events are noncore features and only bring additions value to the OSPM system. For a computer system, some generic features are lid switch, device insertion & removal, battery events and plug-and-play configuration.

## 2.4 Platform & RTOS Consideration

To experiment with and validate the ideas that emerge from the work, it is necessary to identify suitable platforms that can be used. As discussed earlier, the project looks at modern and next generation embedded systems. Therefore, it is crucial to consider systems and options that are likely to become increasingly more relevant in the future. In this section, we identify the hardware and software solutions that are likely to be used in the course of the project.

### 2.4.1 Embedded Hardware

The following types of hardware are becoming more and more relevant in the embedded systems landscape:

1. Multi-core Processors - more than one processing element on the same chip.
2. Reconfigurable systems, incorporating a CPU and an FPGA.
3. Configurable system-on-chip and system-on-chip solutions

Given the timeframe, aims and constraints of the project, it is impractical to work on all the different hardware. However, working on a few platforms is a viable alternative that would highlight the efficacy and challenges of the system.

#### *2.4.1.1 Infineon's TriCore TriBoard*

With the explosion in the number of transistors that can be integrated into a single chip, modern microcontrollers and microprocessors are becoming increasingly more complex. There is a trend to incorporate multiple

programmable processing elements onto the same chip. Various silicon vendors have taken up this trend and there are now solutions available from various providers. For this project, the Infineon TriCore family has been identified as a suitable set of processors.

The TriCore architecture from Infineon Technologies unifies the best features of microcontroller architectures (such as fast context switching, I/O capability), RISC CPU architectures (such as load/ store design, pipelined CPUs, superscalar design) and DSP architectures (such as zero-overhead loops, DSP addressing modes, hardware multiply-and-accumulate units). This unified core is then fused with application-domain specific peripherals. The bus architecture allows multiple processing elements to be embedded onto the same chip. Also, certain architectural features allow the RTOS kernel to be very "thin" in the TriCore processors [Infi98a].

Typical peripherals [Infi01c] in TriCore processors include synchronous and asynchronous serial channels, parallel ports, timer units, watchdog timers and field buses. Given the nature and number of the peripherals available on the TriCore, combined with the complex unified CPU, it is fairly representative of complex modern microcontrollers.

In addition to being representative of complex embedded microcontrollers, the Infineon TriCore is an extremely attractive processor for this work because it incorporates a second, less powerful CPU on the same chip. This processor, called the Peripheral Control Processor (PCP), is a programmable processor that is designed to be used as an intelligent DMA controller [Infi00e, Infi01a].

Depending on the specific implementation, the peripheral control processor may have up to 64K-word of local code memory on chip and some amount of local data memory on chip. By using the on-chip bus, the PCP can access internal and external memory and memory-mapped peripherals.

The TriCore interrupt system [Infi00b, Infi00d] allows an interrupting source (such as a timer, serial port or an external interrupt pin) to be programmed to interrupt either the main CPU or any other interrupt service provider that has been implemented in the specific implementation. In the case of the currently

available TriCore processors, both the CPU and the PCP are implemented as interrupt service providers. However, the architecture supports up to four service providers on the same chip, and it is expected that an extra CPU and/ or extra PCP are likely contenders to be implemented as additional service providers in future implementations.

The TriCore microcontrollers are representative of modern complex embedded processors that provide a large set of application-domain-specific peripherals, coupled closely with a powerful CPU and independent co-processor.

***Processors in the TriCore Family***

The TriCore architecture has been instantiated into a number of processors. The TriCore architecture v1.3 integrates basic microcontroller peripherals that include general purpose timers, asynchronous serial channels, synchronous serial channels, parallel ports, system timer and a watchdog timer. The system block diagram of the TC1797 is shown in Figure 20 below. In addition, the platform is aimed at the automotive industry, and feature a host of peripherals well suited for that segment of embedded systems. There are a few good reasons why the TriCore architecture processors are well suited for use in this project. These are:

1. Multiple processors on the same chip: The TriCore processors include the main unified processor and the PCP on the same chip. It is possible for the PCP to decidated as a co-processor for managing power. This is due to the organization of the TriCore modules around the Flexible Peripheral Interconnect (FPI) bus which makes it possible for both the processors to access all the peripherals and memory.

2. Board Support: The TriCore processor is available for easy prototyping and evaluation using a set of evaluation and development boards. This allows for quick development for verification and testing of the ideas.

3. Software support: The TriCore architecture is well supported by RTOS software. The μC/OS-II RTOS is available in source form, for the

TriCore TC1797. Real-time Linux and T-Kernel have also been ported to the TriCore architecture.

#### *2.4.1.2 T-Engine*

T-Engine offers an efficient development environment for the development of portable information devices, home electronic appliances and other network devices in a short period of time. To support efficient development, T-Engine standardizes hardware (T-Engine board) and real-time kernel (T-Kernel), and especially encourages distribution of middleware.

T-Engine aims to reduce the development time, efforts and cost by creating an environment that eases the cooperation among semiconductor manufacturers, hardware designers, software developers and system manufacturers, which leads to a synergic product development environment. As stated in [Tefs03], the combination of advanced semiconductors, implementation and software technologies in T-Engine, makes it suitable for the development of advanced application products.

The standard T-Engine hardware is built around a 75mm-by-120mm CPU board that can be combined with an LCD board, power supply board, expansion boards and the like to configure the target system hardware. The CPU board of μT-Engine is even smaller at 60mm by 85mm. Standard specifications are adopted for the mechanical dimensions of the CPU boards and their external connectors. Detailed specifications are shown in the table. Various chips can be accommodated; the hardware is not limited to any particular CPU architecture. A feature of the T-Engine hardware is the compact form factor resembling that of a target system.

*Table 4: T-Engine Specifications*

| | Standard T-Engine | μT-Engine |
|---|---|---|
| CPU | 32 bit | |
| MMU | required | optional |
| RAM | 16MB or higher | 4MB or higher |
| Flash memory | 4MB or higher | |
| Serial I/O | 38,400bps or higher | |
| Calendar clock | Yes | |
| Sound CODEC | Yes(IN:1ch, OUT:2ch) | No |
| eTRON Chip I/F | Yes | |
| LCD Panel I/F | Yes | No |
| Touch panel I/F | Yes | No |
| Extension bus I/F | Yes | |
| Other I/Fs | PCMCIA Slot Type-II x 1<br>USB Host Type A connector x 1 | CF Card Slot x 1<br>MMC Card Slot x 1 |
| Board size | 75mm x 120mm | 60mm x 85mm |

Broadly speaking, the term "T-Engine" comprises specified hardware, operating system, drivers, subsystems, and the applications running on them. But people usually take the narrow sense of "T-Engine". It means the family of hardware based on standard specifications. It is separated into 4 types of Standard T-Engine, μT-Engine, nT-Engine, and pT-Engine depending on the use. The Standard T-Engine and μT-Engine are positioned as development platforms while nT-Engine and pT-Engine are positioned as target platforms.

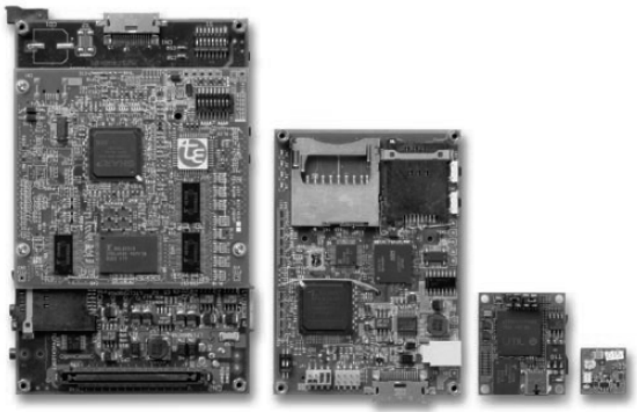


*Figure 13: 4 types of T-Engine Platform*

The Renesas's SH7727 T-Engine development board which is compatible with the standard T-Engine specification would chosen as the choice platform as the board has a plethora of devices such as LCD, USB port, multiple buttons and a built-in touch panel.

### 2.4.2 Embedded Operating System

A few real-time operating systems have been identified for this project. In the first instance, μC/OS-II [Labr99a] [Labr00a] has been identified as the primary RTOS for the activity. In addition, the Linux operating systems has been identified, since it's a popular open source kernel and have similar structure to computer platform that can be exploited in the framework. Lastly, we would look in depth into T-Kernel which is another open source kernel that is packaged with the T-Engine platform.

#### *2.4.2.1 Linux*

Embedded Linux is the use of a Linux operating system in embedded computer systems such as mobile phones, personal digital assistants, media players, set-top boxes, and other consumer electronics devices. Unlike desktop and server versions of Linux, embedded versions of Linux are designed for devices with relatively limited resources. This is mainly due to concerns such as cost and size; embedded devices usually have less memory than desktop computers counterparts. Since embedded devices serve specific rather than general purposes, developers optimize their embedded Linux distributions to target specific hardware configurations and usage situations. These optimizations can include reducing the number of device drivers and software applications, and modifying the Linux kernel to be a real-time operating system. Instead of a full suite of desktop software applications, embedded Linux systems often use a small set of free software utilities and replace the glibc C standard library with a more compact alternative such as dietlibc, uClibc, or Newlib.

Embedded Linux has been ported to a variety of processors not suited for use as the processor of desktop or server computers, such as various CPUs including ARM, avr32, blackfin, cris, frv, h8300,IP7000 m32r, m68k, mips, mn10300, powerpc, sh, or xtensa processors, as an alternative to using a proprietary operating system and toolchain.

The advantages of embedded Linux over other embedded operating systems include no royalties or licensing fees, a stable kernel, a support base that is not restricted to the employees of a single software company, and the ability to

modify and redistribute the source code. The disadvantages include a comparatively larger memory footprint (kernel and root file system), complexities of user mode and kernel mode memory access and complex device driver framework [Wiki09].

Another option is to consider companies who offer "commercial" Embedded Linux distributions generally possess a high level of expertise and have well trained staff ready and waiting to assist you with your project, for a price. Paying money to one of the "commercial" Embedded Linux suppliers can have many advantages, including development tools, useful utilities -- and, of course, support. Most commercial suppliers of Embedded Linux distributions are busy investing in the development of tools and services that will differentiate their Linux offerings from the pack, in order to advance their standing as a prospective partner for companies building Linux-based embedded applications. In many cases, these commercial Embedded Linux distribution suppliers are also making significant contributions to the overall pool of open source software [Lehr09].

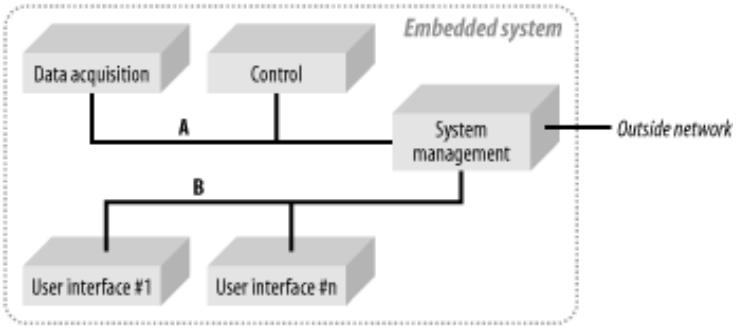



*Figure 14: Example of Embedded Linux System Architecture* [Yagh03]

#### *2.4.2.2 uCos/II*

The µC/OS-II RTOS is a highly portable, ROM-able, scalable, pre-emptive multitasking, real-time operating system (RTOS) kernel for microprocessors and micro-controllers. It is a thin RTOS that offers the basic features that are needed in an embedded operating system. The following are the main reasons for choosing the µC/OS-II for this work:

1. It is a thin, scalable RTOS that is easy to use and understand.

2. It has been used for many embedded applications [Labr00a].
3. It is available in source form (mostly in C).
4. It is highly portable, and has been ported to many target processors.
5. A port for the Infineon TriCore TC1797 is already available.
6. Various aspects of the RTOS have been characterized, and have been presented in research literature. This gives a basis for comparison against the work that others are doing in academia.
7. It is well documented, and well supported.

#### *2.4.2.3 T-Kernel*

T-Kernel is a real-time operating system for new generation embedded systems. It is succeeding µITRON technology, which has been used in the world of embedded systems for many years and has many achievements, and it has been newly designed so that it ought to cope with embedded systems that are becoming larger in scale and higher in complexity. It is also the de-facto RTOS for the T-Engine platform.

Currently, there are various embedded systems from high performance systems using 32-bit RISC processors to small systems using 8-bit/16-bit single chip microcomputers. The adoption of multiprocessors and multi core processors in high performance systems such as home information appliances and equipment in vehicles is being examined to further improve performance. For example, in cars, dozens of processors that work together are being installed. T-Kernel supports all these embedded systems through a common architecture. It is designed for 32-bit processors and supports advanced hardware functions suitable for building large-scale system such as MMU.

T-Kernel have only implemented the basic functions of a real-time operating system in the main body of the kernel, and to provide extend functions, such as a file system, in the form of T-Kernel extensions. This makes it possible to use T-Kernel as a microkernel, and, by combining it with extensions, we can also grasp it as something for constructing highly functional systems.

Being derived from the standard T-Kernel, a different version and extension of T-Kernel have been developed: MP T-Kernel to support Multiprocessor/multi

core processors, µT-Kernel for small systems, and the T-Kernel/Standard Extension which provides more advanced OS functions.

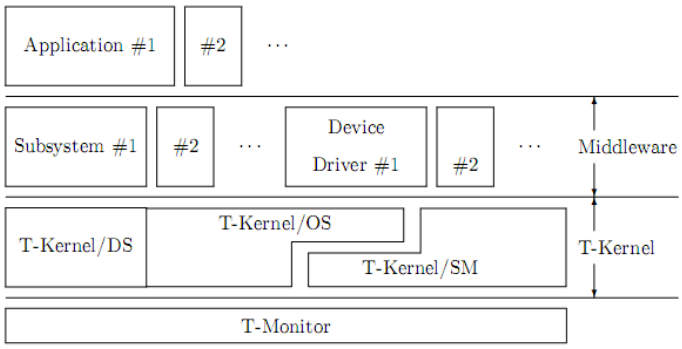



*Figure 15: T-Engine Software Stack*

T-Kernel generally is divided into 4 sections:

1. T-Kernel Operating System (T-Kernel/OS) provides functions from task control, task communication, memory management, exception/interrupt control, time management and subsystem management.
2. T-Kernel System Manager (T-Kernel/SM) provides the following kinds of functions; system memory management, address space management, device management, interrupt management, I/O port access support, basic power management and system configuration information management.
3. T-Kernel Debugger Support (T-Kernel/DS) is utilized exclusively for debugging from kernel internal state reference to tracing an application.
4. T-Monitor is the software used for starting the OS and for debugging. It sits below the T-Kernel thus is not part of the RTOS. However, it sets up the platform akin to the BIOS for the computer platform. Its specifications are defined, and it interfaces with the development environment.

T-Kernel emphasizes the distribution of middleware, and in order to improve the portability and reusability of the software, it is a “single source code” OS with a single source code repository. The source code is distributed to the public free of charge under T-License.

***Subsystem and Middleware***

In T-Kernel, there are functions, called subsystems, which support modules for extending its own operating system functionality. Employing different extension would allow various highly functional systems on top of T-Kernel. The importance of subsystems can be understood by the fact that they help build many middleware which adds a group of customizing functions to the T-Kernel.

Middleware are the supporting layers providing standard interfaces for well-known system infrastructures such as, a network protocol stack, file system, language processing, eTRON-related security software, graphical user interface (GUI), audio processing, and Java. The availability of a wide range of middleware makes it possible to develop stable application products in a short time. This system provides powerful support for distributing software for use on T-Engine. Also, it is very easy and convenient for a developer to incorporate subsystems into the T-Kernel and call the functions provided by the system. T-Kernel itself uses subsystems to implement several modules like process management and file management (in T-Kernel Extension).

***Driver Architecture***

Device drivers are special subsystem that forms a logical middle layer between the platform devices and the applications interacting with them. A driver typically communicates with the device through the bus or communications subsystem to which the hardware is connected. When a calling program invokes a routine in the driver, the driver will issue commands to the device.

Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface. Here is a diagram that shows the driver's position in T-Engine system.

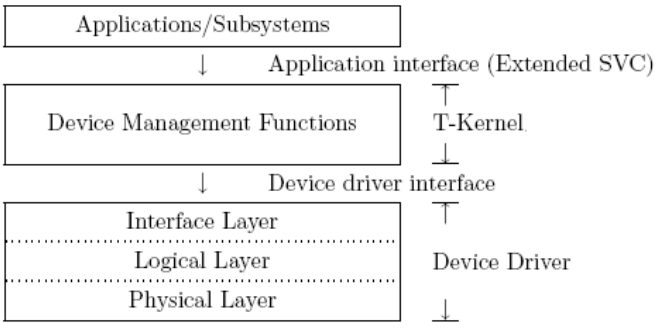



*Figure 16: Device driver in T-Kernel*

The device drivers consist of an interface layer, logical layer, and physical layer. This structure makes it easy to maintain the drivers and to port them between different hardware platforms.

The interface layer handles the interface with the T-Kernel device management function. The logical layer is common to each device and does not depend on the hardware controller. The physical layer provides actual control of the hardware controllers. Of these, the interface layer is able to achieve a large degree of commonality across most device drivers. The role of the device driver interface layer is to reduce the burden on device driver developers and to prevent object code from becoming bloated, by bringing together the aspects that can be standardized.

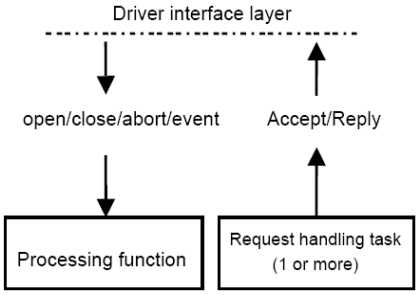



*Figure 17: Relationship between driver & task*

A driver provides open/close/abort/event functions for operating system to invoke. The requests from applications or subsystems are transferred to driver by OS device manager. These requests are not processed immediately, but stored in a queue. The driver will fetch the requests in sequence, and process them asynchronously.

## 2.5 Motivation for RTOS Power Management

Most system-wide power management is achieved by using the RTOS. RTOS have a holistic view of the systems and are the key to managing power on embedded system. In addition, RTOS provide a wealth of services that simplifies the process of application development. Most of the services applications require would be readily available in a typical RTOS. However, recent efforts on OS directed power management lack application-specificity and overlook real-time considerations of embedded systems. Current RTOS that have power management features included are limited in scope and are coupled very tightly to the respective RTOS.

Power management framework in RTOS kernel maintains system power management policy and implements core functions for static power management and dynamic power management technologies. Power management framework plays the role of coordinating energy resource among multiple tasks and adapting system operating state according to task specific requirements. Eventually each application would have its own ideal policy.

There is also a shift in trend that is looking into the hybrid of both techniques (static & dynamic) in making a system more robust and adaptable yet being able to manage power efficiently. However, most of the work tends to only provide benefits to specific context and mostly are still in prototype or design stages due to lack of certain supporting tools.

An additional advantage would be the portability associated with having a common API for the same framework, irrespective of the RTOS and target hardware. Policy developer would only need to develop their policies on the framework and does not need to concern itself with the application.

An ACPI-like framework tailored for embedded system coupled with power management policies offer an untapped potential. However, many of the policies available for various platforms are tied to specific application. Many algorithms to identify power savings hotpots are available but are also predetermined to specific applications.

In this thesis, a framework is described that would allows various power policies algorithms to run on the RTOS. This implies that the application and power policy can be decoupled. For this to be possible and feasible, software tools are required that can analyze the application behaviour and select and configure a power policy to optimally meet it needs.

### 2.5.1 Other Issues in Power Management for Embedded System

There is a constant need to balance constraints, cost and time when incorporating power management in embedded devices. These platforms have more constraints in terms of processor speed, memory space and real-time aspects it might need to adhere to. However, there are a number of other issues with power management for embedded systems that would be explored.

***Reliability***

Power management techniques should not impair the proper working of the system. It should give higher priority to the normal running of the system rather than the ability to save some power. For example, no devices should be accidentally switched off when it is in running mode. There should be sufficient safeguard measures that ensure that no action would be taken for errant policies or wrong predictions. This scope of these measures would depend on the platform and its application.

***Real-time***

The essential requirement of a real-time system is that the system be deterministic with respect to time. The importance of the timeliness in a real time system is reflected in classifying the system to be either, hard, soft, firm, safety-critical or mission-critical real-time systems. Any techniques designed to aid a real-time embedded system should also adhere to the principles of a real-time system and should not contains methods or techniques that would directly or indirectly violate the real-time aspect of the system.

***Flexible & Adaptable***

A good technique would allow the system to adapt to the application and its platform. It should also consider future hardware expandability. The environment of the platform could also change the behaviour of the application and affect the system. Another aspect would be the ability for the technique to work on a range of platforms with minimal recoding (code portability). Embedded systems encompass a wide variety of products ranging from industrial control, consumer electronics, automotive, medical and telecommunications. Hence, a power management framework should cater to all types of embedded systems, making adaptability an important design consideration.

***Resources Overhead (Memory & CPU)***

Power management typically put more pressure on processor and memory, due to the need to execute more algorithms and maintain a set of data to facilitate in making more intelligent decisions. In order to minimise resource overhead, it is vital to reduce the footprint of the framework. A power management technique that raises the memory footprint of the product significantly might not be a desirable design.

***Developmental Efforts***

Developmental efforts are classified under R&D costs, and they contribute to the NRE costs. Especially for low volume productions, it is very important to keep the NRE costs under check. To achieve this, it is necessary that that a major part of the power management system is designed as a standard framework, minimizing the developer's efforts. It also aids in pushing products out to market with a simple power policy and later doing a policy update to improve power savings. The developmental effort when using the power management framework should be less than the effort of tailoring a power management scheme for each application.

### 2.5.2 Adapting ACPI for Embedded Systems

ACPI is a matured architecture for providing power management for personal computers as it has been around for 15 years. Extracting ideas and design

method from this can assist in developing an RTOS based power management framework. However, this requires careful analysis and selection of only relevant ideas. The key aims are to preserve the real time and embedded systems requirements, yet strive to achieve a sophisticated power management system.

Having a good understanding of ACPICA and embedded system helps identify areas that can be optimised, merged or removed. However, as embedded system was not considered, much of the design reference is not applicable and cannot be used for the embedded system domain. Nonetheless, certain portions of their design can be extracted and adapted for embedded system. To further reduce engineering overhead and improve flexibility, AML interpreter could be removed, as each microprocessor requires their own interpreter. The next chapter will propose a power management framework for a RTOS that would review the different modules that make up ACPI and adapt them for embedded system domain.

## 2.6 Summary

In this chapter, a review of the existing techniques in the field of power management for real-time operating system has been presented. An overview of power management techniques that encompass design methodology, hardware control (power coprocessor), hardware optimisation, software control (e.g. power policy algorithm and OS-based power management) and software optimisation have been explained. A critical analysis of these techniques was undertaken which identifies the limitations of the existing techniques. Thereafter, the challenges on power management for embedded system is explained and the three major challenges of ensuring real-time constraints of embedded system, RTOS adaptability and hardware scalability are described. This was followed by identifying the candidate platform and RTOS. ACPICA (ACPI reference implementation) system architecture and the interaction between internal modules were explained. Key ideas from ACPI specification will be adapted to the constraints of embedded systems and used in this thesis.

# Chapter 3
# RTOS Power Management Framework

*This chapter proposes for the creation of a power management infrastructure that is optimized towards specific performance parameters while considering the constraints of current Embedded System.*

## 3.1 Proposed Framework

The proposed framework aims to develop an RTOS based power management framework for embedded systems that would satisfy stringent power requirements. One of the main aims is to propose a novel methodology that would support different power management strategies to work on a standardised architecture.

Having a better understanding of an application will help facilitate the RTOS to utilize prior knowledge of the behaviours and characteristics of the application for supporting dynamic power management. This could help match the appropriate power management strategies to application to obtain the most power savings with nominal effort.

Application-specific characteristics can be identified through offline profiling techniques in order to formulate tailored application-aware power management policies.

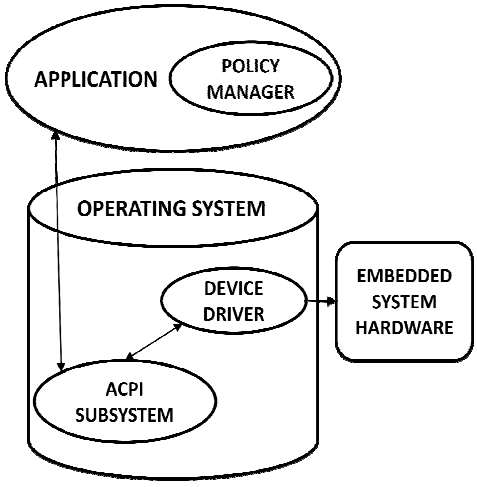



*Figure 18: RTOS-ACPI Architecture*

### 3.1.1 T-Kernel RTOS

T-Kernel is a microkernel that has only implemented basic functions of a real-time operating system in the main body of the kernel, but allows extend functions, such as a file system, in the form of T-Kernel extensions. T-Kernel emphasizes the distribution of middleware, in order to improve the portability and reusability of the software. They provide functionalities to make adding middleware more developer friendly. In additional, the middleware would be able to work across various hardware platforms. A subsystem is a special kind of middleware, running above the kernel which defines a specialized, high-level API for use by application programs. Once a subsystem is installed, its service calls are analogous to RTOS system calls, which are available always to any applications.

Currently, T-Kernel has a limited power management mechanism, which allows the developer to define their own functions to handle power management. This makes T-Kernel a good candidate to implement the proposed power management framework as it would enhance the RTOS with a broader power management scheme.

T-Kernel also boasts a comprehensive device management subsystem, which allows the development of device drivers. Device drivers are special subsystem that forms a logical middle layer between the platform devices and the applications interacting with them. This structure makes it easy to maintain the drivers and to port them between different hardware platforms.

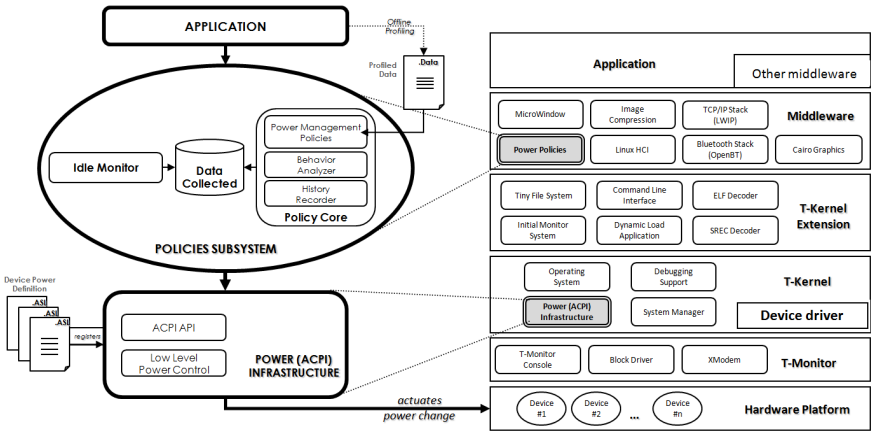



*Figure 19: RTOS-ACPI Framework for T-Engine*

### 3.1.2 Responsibilities of Framework

The essence of the framework is to design a subsystem (kernel module) that will extend the functionality of the RTOS. This would enable the RTOS to manage power at a holistic level and reduce the dependence of application to manage power. The subsystem would not be utilised if an application accesses the hardware for normal usage that is non-power related. The subsystem has sole control of all power related functions in the system.

The framework has the following responsibilities:

1. Offers a comprehensive infrastructure which the *policy manager* could utilize to implement intelligent power management in the system.
2. Define and provide standard interfaces for interactions between the system components like platform hardware, RTOS and the *policy manager*.
3. Independent of the RTOS and/or the hardware platform used.

## 3.2 Preliminary Design of RTOS-ACPI

The initial design was derived from a general idea of ACPI rather than the reference implementation in ACPICA. A data structure, *device power state*, is constructed during the subsystem initialisation and will populate itself with data obtained from the *device manager* and *ACPI device driver* via the *ACPI manager*. Each *ACPI device driver* would contain the necessary power information for the particular device, such as device name, the various power modes, methods to change the power state of the device, etc.

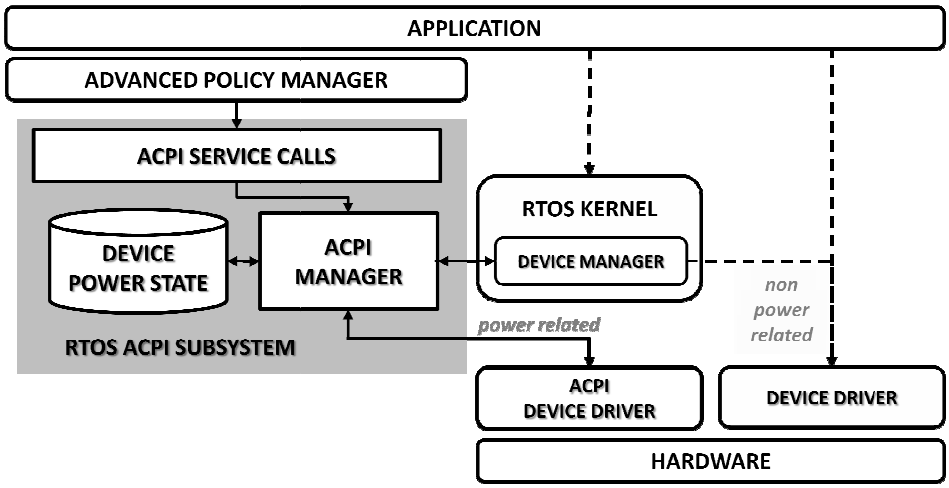



*Figure 20: Preliminary ACPI Framework*

### 3.2.1 Core RTOS-ACPI Subsystem (*preliminary*)

The subsystem is composed of three modules; *device power state*, *ACPI service calls* and *ACPI manager*. The *ACPI manager* interacts with the *ACPI device driver* and T-Kernel's *device manager* to obtain power information of devices. The figure below illustrates the various modules and the interaction between them. The nucleus of the subsystem is the *ACPI manager* that connects all the modules together.

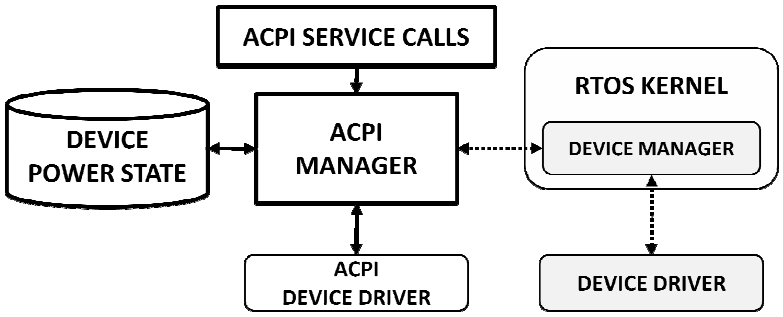



*Figure 21: Interaction among modules of the RTOS-ACPI Core Subsystem (beta)*

### 3.2.2 ACPI Service Calls

This module exposes the system calls needed by the application and power policy to utilise the ACPI subsystem. It takes in the request with the essential inputs and passes the request along to the ACPI manager to obtain and link the requested device.

Application can have direct access to the *ACPI device drivers*, hence bypassing the ACPI subsystem. Though, this would circumvent the subsystem and would jeopardise the system as the subsystem would only have a vague power status of the platform. Therefore, it is vital for the application developer to ensure any requests to manage power would have to go through the ACPI subsystem.

The provided functions help the power policy to obtain power information of the devices (e.g. *GetCurrentState*, *GetDeviceNames*, *etc*) in the platform. The power policy would determine the next state using this information and access the methods (e.g. *ChangeDeviceState*) provided to change the device to the desired power state.

Some functions, such as *EnterSystemState*, *ChangeBusState* and *EnterG0State* assist the power policy to change the power state for the entire platform with a single system call. The following table list the system calls that are offered by the ACPI for power policy.

*Table 5: ACPI system calls for application and power policy*

| System Call | Description | |
|---|---|---|
| ReadDeviceState | *description* | read's the device's state from the stored value, not current |
| | *input arguments* | device name |
| | *return value* | device power state if successful |
| GetCurrentState | *description* | gets the current state of the device by reading from the device's registers |
| | *input arguments* | device name |
| | *return value* | device state if successful |
| GetIdleStart | *description* | obtain idle start time |
| | *input arguments* | device name |
| | *return value* | idle start time if successful |
| GetDeviceNames | *description* | count the number of attached devices |
| | *input arguments* | none |
| | *return value* | number of attached devices if successful |
| DetStateTransTime | *description* | the time to transit a device from one power state to another |
| | *input arguments* | device name, from state and to state |
| | *return value* | time in ms if successful |

| | | |
|---|---|---|
| CheckDeviceBusy | *description* | find out if the device is being used |
| | *input arguments* | device name |
| | *return value* | positive integer if successful (busy), 0 if not busy |
| ChangeDeviceState | *description* | changes device's state to the specified power state |
| | *input arguments* | device name, state to transit to |
| | *return value* | E_OK - device transition successful |
| EnterSystemState | *description* | transition the entire system to predefined custom state |
| | *input arguments* | state from S0 to S4, typically |
| | *return value* | E_OK if successful |
| EnterG0State | *description* | transition the entire system to the wake state, all devices D0 |
| | *input arguments* | none |
| | *return value* | E_OK if successful |
| ChangeBusState | *description* | transition all devices in a bus to a required state |
| | *input arguments* | bus name, state to transit to |
| | *return value* | E_OK if successful |

### 3.2.3 ACPI Manager

The nucleus of ACPI subsystem, *ACPI Manager* interacts with the other modules in the ACPI subsystem.

***Subsystem initialisation***

When the subsystem initialises, the *ACPI Manager* obtains a list of devices from T-Kernel's *device manager*. It would then attempt to call each device (via the *ACPI device driver*) to acquire more information on power features for each device stored in *device power state* repository. After populating the *device power state* repository, the *ACPI Manager* would initialise all the devices in the platform to the predefined power states. The *device manager* will interaction with the ACPI subsystem after it provides the list of devices in the platform. It is the role of the *ACPI manager* to validate if the devices in the list are active.

***Installing new devices***

Adding new devices require the accompanying device driver (firmware obtained from device supplier) to be installed. This device driver will register the device to the *device manager* (this operation is performed by T-Kernel and

not the RTOS-ACPI framework). The device driver would also inform the *ACPI manager* of the new device and registers all the necessary information (registers, control method, etc) required by the subsystem.

***Subsystem operation***

The *ACPI Manager* would take in requests from power policy via the *ACPI Service Calls* to find requested device and its corresponding methods that was recorded into the *device power state*. If the request is to obtain device power information, the *ACPI Manager* would retrieve them from the *device power state*, but if the request is to change the power state of a particular device, it would access the *ACPI device driver* using information from the *device power state*.

### 3.2.4 Device Power State *(power information repository)*

The *device power state* is a structure that contains information with the purpose of describing the platform's power features. A list of devices is obtained from the *device manager* while the remaining power information is obtained from the *ACPI device drivers* in the system.

In addition to the name of device, its current power state and its status (open, close, etc), the structure also contains data (policy block) that would be common and used by various to power policies, such as *idle time* and *prior power state*. The structure would also encapsulate the various states that particular device can transit to. Some devices can support only two power states (on and off) while more advanced devices can support various granularities of power state.

The *ACPI Manager* only need to know which state the device need to transit to and the *ACPI device driver* would execute the corresponding control method. The control method does not reside in the *device power state* and is kept in the *ACPI device driver*. These functions configure the device via register read/write. However, configuration for certain devices might require complicated commands (e.g. H8 co-processor in T-Engine).

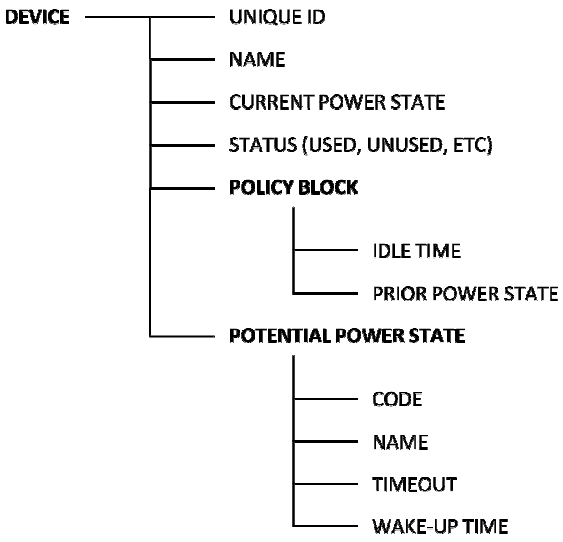



*Figure 22: Outline of device power state structure*

### 3.2.5 ACPI Device Driver

The *ACPI device driver* contains control methods and data on the power features of the device. When the *ACPI device driver* is installed, it would register itself (e.g. *pmDevReg*) to the *ACPI manager* and initialise the device to the predefine power state. It would then update the *ACPI manager* on the power features of the device. All data, except policy block (shown in figure above) would be sent to the *device power state* via the *ACPI manager.*

*Table 6: ACPI Device Driver System Calls*

| System Call | Description | |
|---|---|---|
| pmDevReg | *description* | register itself into the RTOS-ACPI manager, when a new device driver is installed |
| | *input arguments* | device name, number of power states, details of power state |
| pmDevUnreg | *description* | unregister itself from ACPI manager, when a device driver is removed (uninstalled) |
| | *input arguments* | device name (e.g. LCD) |
| getState | *description* | returns the power state of the device |
| | *input arguments* | device name |
| deviceState | *description* | change the device to the requested state |
| | *input arguments* | device name, power state (e.g. D0, D1, etc) |

### 3.2.6 Design Considerations and Enhancement

To test the subsystem, a few policies were developed (see next chapter). This led to alternative design and further enhancement of the framework.

***Encapsulated ACPI device driver***

A device comprising of two drivers (*ACPI device driver* and original *device drivers*) eases development, as the developer does not need to modify their existing drivers and just create an additional driver specifically for the power features of the device. This method is useful when the *device driver* source code is not available.

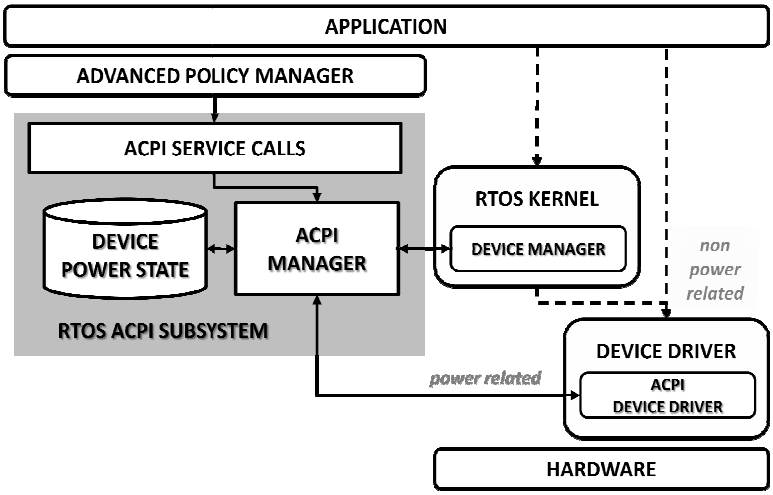



*Figure 23: Alternative ACPI Framework*

Access to the source code of these *device drivers* could lead to an alternative subsystem design. Encapsulating the *ACPI device driver* into the original *device drivers* would allow the *device driver* to manage the device for both normal operations and power control methods. This embedding of *ACPI device driver* into the original *device driver* requires the *device driver* to be recompiled.

***Updating System Power Information***

The power management subsystem would update itself when a new driver is installed in the RTOS. This allows the system to expand and support future hardware upgrades. However, there are embedded systems that are predominantly application-specific and rarely require hardware redesign. These platforms do not require the on-the-fly update and a single update on reboot

could reduce the communication overhead brought about by the interaction between *device manager*, *ACPI manager* and *device drivers*. This alternative mechanism could allow the framework to update itself at start-up.

***Data redundancy versus Memory constraint***

Another design feature is the power data kept in the device power state is the same as the data found in the device driver. The ACPI manager would need to ensure that the two are kept synchronised. A subset of device power state structure is also kept within the policy, though this is maintained by the policy itself and not by the ACPI manager. On the other hand, if the platform has limited memory, an alternative design would be to consolidate all the system power information in the ACPI repository. Power policy can quickly tap on this central repository to obtain power information of a particular device without the need to maintain its own structure.

***Review ACPICA***

Further review of ACPICA help procure beneficial features that assist in tackling these design considerations. Key areas include ASL and namespace that could describe the platform's power features externally and internally respectively. Another useful mechanism is a novel technique to actuate power transition that could simplify power policy development.

## 3.3 Improved Design of RTOS-ACPI

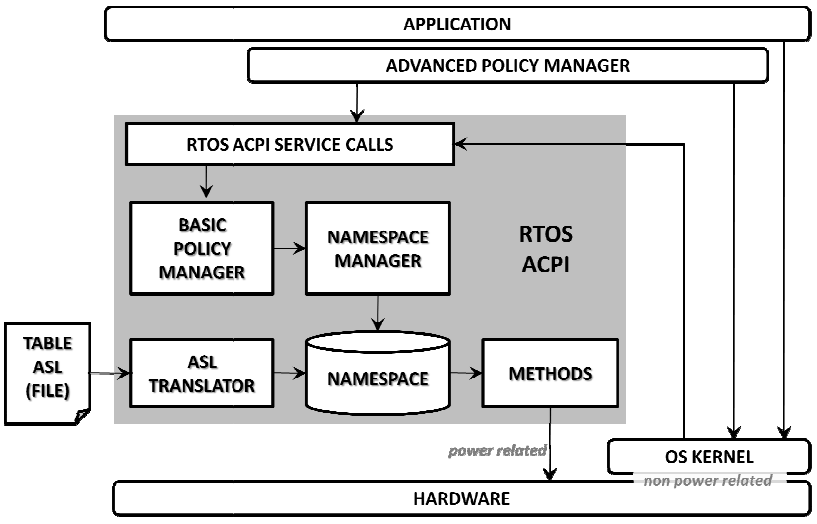



*Figure 24: Proposed Improved ACPI Framework*

Salient features of ACPICA were identified and incorporated into the design after reviewing the preliminary design. The improvement (shown in figure above), significantly expands the functionality of the RTOS-ACPI subsystem. One notable improvement is centralising the power features into the subsystem, making it the only module that can manage power in the system.

The improved subsystem takes in an *ASL table (file)* which describes the platform's power features and power control methods. This is then translated by the *ASL translator* to a *Namespace* structure in the RTOS native format (see Appendix F). The *Namespace* comprises of information like register addresses and device names, but does not contain any functions to control power; these are located in the *Methods.* The *Namespace manager* would be responsible to manage any request to access the *Namespace.*

*Advanced policy manager* is another subsystem that implements the OSPM. It makes intelligent decisions based on the entire system activity and then uses the RTOS-ACPI subsystem to implement its decisions; actuate the devices. However, these decisions are subjected to the *basic policy manager* that contains rules to guarantee they do not compromise the integrity of the RTOS.

### 3.3.1 Core RTOS-ACPI Subsystem (*enhanced*)

The improved subsystem is composed of several modules that provide the essential services. The figure below illustrates the various modules and the interaction between them.

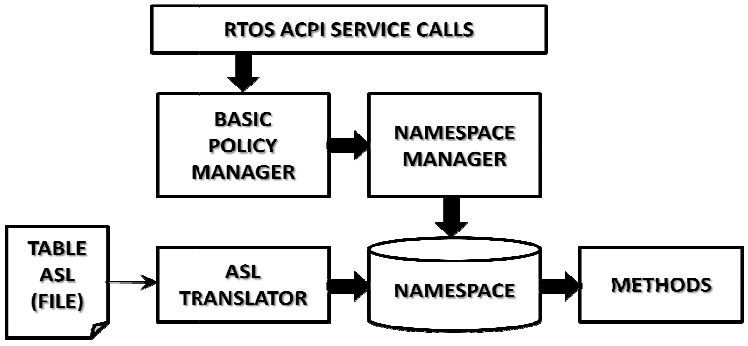



*Figure 25: Internal Modules of the RTOS-ACPI Core Subsystem*

***Fixed Events***

RTOS-ACPI interacts with the hardware directly, since it is equipped with the hardware-specific information in its *Namespace* and its corresponding control methods found in *Methods*. Accessing appropriate part of the *Namespace* provides a gateway to the hardware devices or components.

Most embedded system has on-chip devices which are easily accessible via registers addresses. In ACPICA, events can be treated as fixed or generic. In embedded systems plug-and-play devices are limited and would still require the device driver to be installed onto the RTOS prior to the device being plugged in. This allows be treated as a fixed event. Adding new devices requires reboot, as the information can only be obtained from the updated *ASL table (file)*. Therefore, all requests for RTOS-ACPI service has been simplified and treated as fixed events.

### 3.3.2 ACPI Source Language (ASL) Tables

Typically, platform hardware developers provide the tables (DSDT, RSDT, etc) that contain data and control methods pertaining to each device on the platform. In ACPI, the table are stored in the ACPI-BIOS and is read by the operating system during start-up. Since, ACPI-BIOS is not a standard part of an embedded system, the design of RTOS-ACPI has been modified such that the tables can be provided to the RTOS via a plaintext file. The tables are loaded into the subsystem and used to build a large hierarchical data structure called *Namespace*.

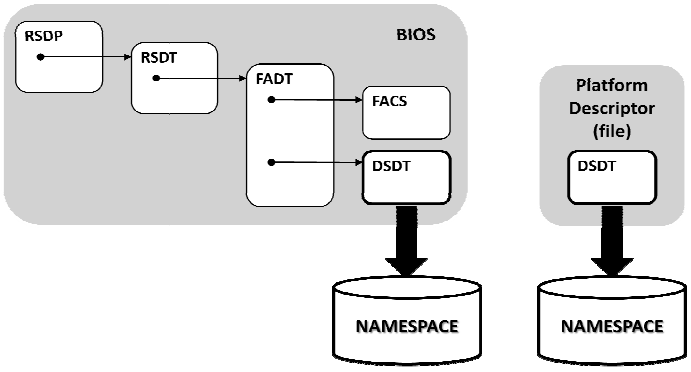



*Figure 26: Difference in generating Namespace between ACPI and RTOS-ACPI*

Like ACPI, the tables are coded in ASL to make them OS independent. However, in this implementation, ASL code is not compiled to AML code, but is translated to OS specific data structures which are utilized at run time. The ASL compiler and the AML interpreter have been discarded to simplify the framework and reduce overheads.

The ASL is described in plaintext file using ASL syntax (see Appendix C and D), and not in assembly language or OS native format. Reviewing the ACPI tables revealed that only the DSDT table contained vital power information and was incorporated into the plaintext file while omitting the other ACPI tables. In addition, the features and format of the DSDT is retained to preserve the matured standard to describe the platform sufficiently. This makes it easier for people who have experience in ACPI to use this framework. Any changes in hardware would only require the input ASL file to be updated and the subsystem restarted.

The control methods to configure devices in most embedded system are simply register read and write. This resulted in incorporating only a small subset of ASL commands to realize the RTOS-ACPI subsystem implementation. The current implementation of ASL also supports both fixed and hot-pluggable devices, multiple processors, on-chip peripherals and communications bus, such as USB and system bus. A complete list of ASL operators is in Appendix D.

```
Scope (_SB) {
        Device (KBPD) {
                Name ( _ADR, 0x2000)
                Name ( _S0D, 0x0)
                Register (SystemIO, 8, 0, 0x0020, 8, TPLC)
                Register (SystemIO, 8, 0, 0x001A, 8, TPLR)
                External (H8_WRITE, MethodObj, IntObj, 3, IntObj, IntObj, IntObj)
                Method (_PS0 ) {
                        H8_RESET ()
                        H8_WRITE( TPLC, 1, 0x0F)
                        H8_WRITE( TPLR, 1, 0x04)
                }
        }
}
```

*Figure 27: Sample of ASL script*

***Updating ASL Table to Include a New Device***

A device in a system can come under any of the following scopes; system bus, on-chip devices, processor and USB bus. These scopes are the top level hierarchical objects in a *Namespace*. If the device does not fall under any of the current scopes, a new scope object can be created with the new device object underneath it. All power information pertaining to that device needs to be described in the *ASL table (file).*

The *ASL table (file)* component makes it easy to add a new device to the platform to make it accessible the subsystem. As mentioned earlier, adding new hardware to the system, requires updating the *ASL table (file)* and a subsystem restart. However, if the new device cannot use existing control methods (found in *Methods*) to manage the device, then these control methods need to be incorporated into the *Methods* module. The change would require the subsystem to be recompiled.

The following are the ASL operators used in this implementation of RTOS-ACPI:

| ASL Operator | Description |
|---|---|
| Scope | Open named scope |
| Device | Declare a bus/device object |
| Processor | Declares a processor |
| Name | Declares a named object |
| Register | Declares a register |
| External | Declares an external object |
| Method | Declares a Control method |
| AND | Integer Bitwise And |
| OR | Integer Bitwise Or |

### 3.3.3 ASL Translator

The *ASL translator* is a module in RTOS-ACPI that gets the ASL file as input and converts it into a data structure in the native OS language that can be accessed by power policy to manage power on the platform.

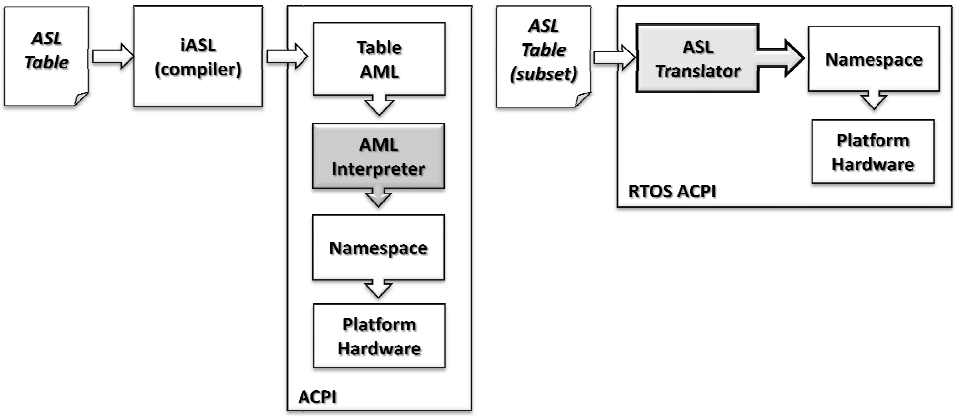



*Figure 28: Difference between how ACPICA & RTOS-ACPI handles ASL table.*

The previous section described the ASL code as a XML-script plaintext and not a compiled code, like AML. This removes the need for an interpreter and introduces a simple translator to map the platform's power information onto *Namespace.* This improves portability as AML interpreters does not need to be developed for each platform. This module is similar to the *ACPI Table Management* (see figure 16) where it manages the tables in ACPI-BIOS for ACPI. Simplifying the *ACPI tables* to a single DSDT table enabled us to replace an interpreter to a simple and flexible translator.

***Lex and Yacc Tools***

ASL translation is done using two tools; lexical analyzer called *Lex* and a parser called *Yacc*. *Lex* and *Yacc* are free development tools that help in writing software in C which interprets or transforms structured input. This tool (see Appendix B) was initially used within compilers but is now incorporated into the RTOS-ACPI which takes in the input *ASL file* (ASL file for T-Engine at Appendix E) and translates it into C-structure called *Namespace*. Future extension is supported as more 'grammar' (ASL operators) can be added when needed. When a subsystem requires additional device information, it can make the changes in the ASL file which would update the *Namespace* structure without any modifications to the RTOS-ACPI subsystem.

A distinct difference in this implementation is that the *Namespace* is created by the *ASL translator* and not the *Namespace manager* (as in ACPICA).

### 3.3.4 Namespace

*Namespace* is a hierarchical structure that represents the platform hardware in the subsystem. The *Namespace* is built by the ASL translator using the ASL file that contains the DSDT table during subsystem initialization. After the *Namespace* is built, the subsystem will update the state of the device in *Namespace*. When a request is made to access the device, the *Namespace* is accessed by the *Namespace manager* find the control methods used to access the device.

Basically, *Namespace* organises the named objects in the DSDT (data and control methods) in the form of a tree structure. It contains only the names, not the actual functions, for example, a control method's name and register addresses would be present in the *Namespace* but the methods that utilises these registers addresses would reside in *Methods* component. The relevant data (registers addresses) would then be fed into the control method and executed.

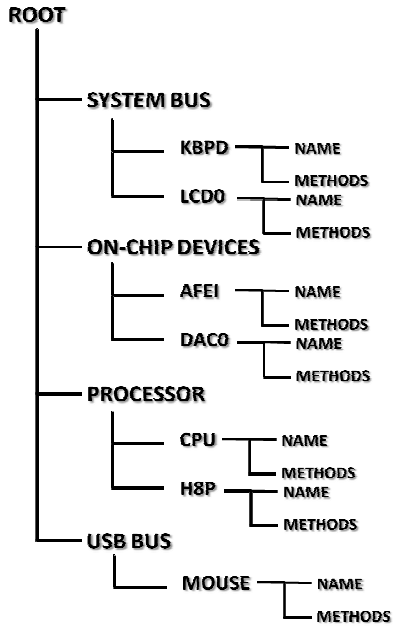



*Figure 29: Outline of RTOS-ACPI Namespace*

The *Namespace* is analogous to a central repository which contains information that describes the platform's power features. This makes it possible to tap onto a single repository to develop power management policies for the system. It can

also be extended to support policies by keeping track of various generic data for a particular device; e.g. idle time of device. These additions can easily be made by appending the *ASL table (file)*. Note that these data are derived and calculated by the policy and not RTOS-ACPI, but are stored in the *Namespace*.

### 3.3.5 Namespace Manager

*Namespace manager* provides functionalities to access and maintain the *Namespace* structure. Access is provided to the structure components via system calls that parse and evaluate named objects on the *Namespace*. There is no other means to use *Namespace* except *Namespace manager* that is granted exclusive access. The responsibilities are device enumeration, obtain device information, update device information and finding device control methods. In this implementation, the creation of *Namespace* is done by *ASL translator* and not *Namespace Manager*. This imply that adding new devices would require the *ASL table (file)* to be updated and a subsystem restart.

*Table 7: Namespace manager system calls to access data from Namespace*

| System Call | Description | |
|---|---|---|
| getData | *description* | obtain data stored under the object tag |
| | *input arguments* | device name, object tag (*e.g. _DSS, _ADR*) |
| | *return value* | integer data stored in object tag |
| getMethod | *description* | obtain detail of method to change a device to a specific power state |
| | *input arguments* | device name, power state |
| | *return value* | returns a structure that contains the details of the requested method |
| getTransTime | *description* | retrieve the transition time for state 1 → state2 |
| | *input arguments* | device name, state1, state2 |
| | *return value* | returns the transition time (integer, ms) |
| totalDevice | *description* | obtains total number of devices under a specific scope |
| | *input arguments* | scope tag (*e.g. _SB, _USB*) [*if NULL, then all devices in system*] |
| | *return value* | total number of devices under a scope |

Each request to *Namespace manager* requires prior approval from the *basic policy manager* before proceeding to access the *Namespace* for the relevant

control method. If the request goes against any of the rules establish in the *basic policy manager*, the operation is rejected and is not executed.

A major distinction from the previous design, the *Namespace manager* does not monitor the *ACPI device drivers* which have been removed. It's sole responsibility is to manage *Namespace* and gets its filtered request from *basic policy manager*.

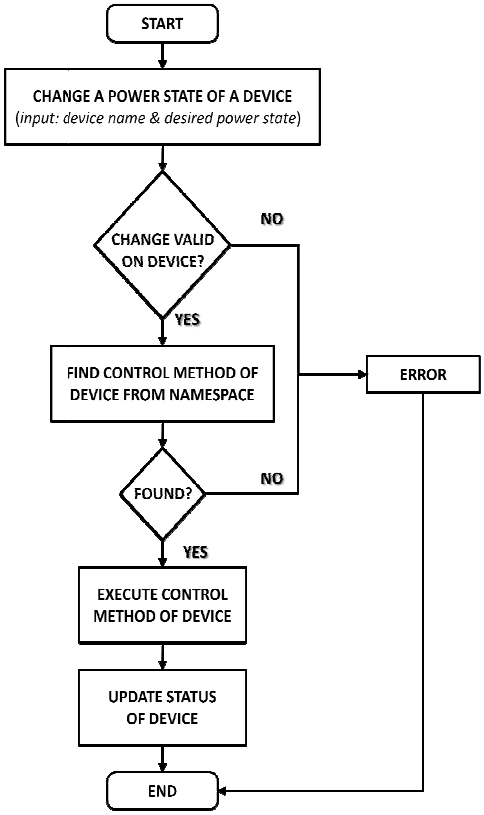



*Figure 30: Power state transition flowchart*

### 3.3.6 Methods

Most embedded platforms allow the device to be configured with register read and write mechanisms. To solve the problems encountered (combining power controls with *device driver* or splitting them into their own *ACPI device driver*), a decision was made to put them within the control of the RTOS-ACPI subsystem. All the consolidated control methods for power, now reside in the subsystem, thus removed from all device drivers. This would significantly simplify the framework as device drivers does not need to be recompiled. This

would also reduce disrupting the device drivers which is accessed by applications.

This is consistent in making the subsystem the only entity that can manipulate power features of the platform. Moreover, this would also remove any direct access to power features by applications and power policies as compared to the prior implementations that could have allowed developers to exploit the loophole and totally bypass ACPI subsystem, consequently compromising the framework.

The *Methods* default method of controlling power is through hardware registers in performing power state transitions. The subsystem can tweak the device via their respective registers to configure the device to the required power state while totally bypassing the *device driver*, even though the device is being accessed by the *device driver*. The onus would be on the *Namespace manager* and *basic policy manager* to ensure that the device is always in a valid state when the system needs to access it.

In some circumstances, certain embedded platform have special methods to access power features, like the SH7727 T-Engine, where H8 co-processor is used as a power controller for a variety of on-board devices. Special methods are required in order to control the devices that are connected this controller. In this case, methods that send and receive commands over a serial interface (connected to the H8 co-processor) to control the devices are required.

The original AML code would have contained all the methods to control these devices, but the design decision to remove the interpreter to reduce the overhead, necessitate this module. Besides, the reduced ASL operator set was not capable of handling the ability to have different control methods for each device. Hence, this module was required to interface device specific code to RTOS-ACPI subsystem. Thus *Methods* module was created to contain all the different methods or function that are used by the devices in the platform. This brings in hardware dependency into the RTOS-ACPI subsystem, but is necessary for devices that cannot be control via simple register read and writes.

This module generally imitates the *ACPI Hardware Management* and are responsible for the following:

- initialize devices
- configurations method
- various power control methods
- device status manipulation
- execute control method

***Adding new methods***

New methods can be added to the source files of the RTOS-ACPI subsystem. The file genmeth.c contains the definitions of all methods and is the place where new methods should be included. Additional support for new methods need to be added in the RTOS-ACPI code to handle the new symbols corresponding to the new methods.

### 3.3.7 Basic Policy Manager (BPM)

The *basic policy manager* is an important module that help maintains the integrity of the RTOS by validating all requests against its set of rules. This is to ensure none of the requests might take system power transitions that might violate basic logic rules.

This module was not described in the ACPI specification, but was incorporated into this design to prevent any disruptive decisions by the power policy and help preserve the real-time aspect of the system. It was also designed to help alleviate the additional burden from the *advanced power policy* which would allow the *advanced power policy* to concentrate on determining the next state of the devices in the platform.

More rules can and should be added as and when new rules are identified that would help the RTOS. However, it should be kept simple as not to cause significant overhead.

Currently, this module supports two basic policies:

1. Transition of a device to low power (D3) shall not be permitted if the device has been opened by any application task for operation.
2. An error code is thrown when the advanced policy manager attempts to transit an unconnected device to any power state.

### 3.3.8 RTOS-ACPI Service Calls

In this implementation in which the *device manager* of the T-Kernel does not allow any changes to be made, a workaround is made to embed instructions into the device driver so that the device driver would notify the subsystem of an event and synchronise information of the controlled device.

However, if the *device manager* could be modified, it would only need to inform the *ACPI manager* when the device driver is being accessed, thus reducing the burden on the device driver.

The subsystem needs to be aware of the device usage which is available in the device manager. Whenever a device is opened, closed, dynamically inserted or removed (in the case of hot pluggable devices), the respective device driver informs the RTOS-ACPI subsystem for it to stay updated. For this purpose, the subsystem provides a set of system calls exclusively to be used by the device drivers.

*Table 8: Device Driver System Calls*

| System Call | Description | |
|---|---|---|
| DevOpenNotify | *description* | notify RTOS-ACPI, when the device driver is *opened* |
| | *input arguments* | device name |
| | *return value* | E_OK if successful |
| DevCloseNotify | *description* | notify RTOS-ACPI, when the device driver is *closed* |
| | *input arguments* | device name |
| | *return value* | E_OK if successful |
| DevAttachNotify | *description* | notify the RTOS-ACPI, when a plug-and-play device is *attached* |
| | *input arguments* | device name |
| | *return value* | E_OK if successful |

| | | |
|---|---|---|
| DevDetachNotify | *description* | notify the RTOS-ACPI, when a plug-and-play device is *detached* |
| | *input arguments* | device name |
| | *return value* | E_OK if successful |

***Interaction with Advanced Power Management Policies***

The responsibility of a policy is to calculate and decide when best to change the state of a device. Whether the decision is valid or if can even be done should be the responsibility of the subsystem (*basic policy manager*). This would significantly reduce the amount of device specific data that it requires to keep track. Most of device specific data can then be placed into the *Namespace* so that it can be easily accessed by any policy. Though the data can be accessed from *Namespace*, it is the policy's responsibility to measure and update this data.

The interaction between the *advanced policy manager* is representative of the method in which the RTOS directs power management decisions. This is not a part of the subsystem but within the proposed framework.

The primary client of the subsystem is the power policy, which is the intelligent part of the framework which utilizes RTOS-ACPI services to actuate the corresponding device. Its interactions with the application and the RTOS, the *advanced policy manager* calculates and predicts power management decisions, such as when to transit a device to a low power state and to what extent. The subsystem provides the *advanced policy manager* with the necessary information and services it might require in order to take an informed decision to manage the system power.

A case to illustrate how RTOS-ACPI could be useful to the *advanced policy manager*. When the *advanced policy manager* predicts that a device is not going to be used for a period of time, it might decide to transition the device to a low power state. However, it would use the device transition time and compares it with the estimated device idle time to decide whether this decision is profitable. RTOS-ACPI provides service (e.g. *DetStateTransTime*) which gives the transition time of the device. These features assist the *advanced policy manager*

in determining the merits of a policy decision. RTOS-ACPI also provides convenient services like putting all devices in a bus to a particular state, waking the entire system to the working power state, et al. A well designed policies framework can stand to gain from the subsystem leading to a synergic system.

***RTOS-ACPI Error Codes***

Each calls in the subsystem returns the code E_OK code if the control method is successful, else a defined error code is returned which the developer can use to take corrective actions.

*Table 9: RTOS-ACPI Error Codes*

| Error Macro | Numerical Value | Definition |
|---|---|---|
| E_OK | 0 | Success |
| E_PM_DISAPPROVE | -1 | RTOS-ACPI power manager disapproved |
| E_DEV_NOEXS | -2 | Device not present in the platform |
| E_STATE_NOEXS | -3 | State not supported |
| E_DEV_NOATT | -4 | Device not attached |
| E_TT_UNKNOWN | -5 | Transition time unknown |
| E_ERR | -6 | General error |
| E_BUS_NOEXS | -7 | Bus not on system |

## 3.4 Comparing proposed framework with existing frameworks

The proposed power management framework utilizes key concepts of ACPI specification. The ACPI specification has not been implemented for this project because of its unsuitability for hard real-time systems.

Android which supports its own power management on top of the standard Linux power management is limited to using "wake locks" to request for CPU resources. Our framework handles all devices on the platform as power consuming device, including the CPU. In addition, our framework can incorporate advanced power policy algorithm tailored to the application to manage the power of the platform.

QNX Neutrino which is a hard real-time system have incorporated the power management framework into the RTOS and would require the application to be

migrated to the RTOS to take full advantages of the power features. Similar to our framework, QNX's framework can manage all the devices in the platform, but does this through device driver. This requires each device drivers needs to be embedded with their respective power methods and information. Our framework reduces the engineering effort needed to update the RTOS of power methods and data of the platform by incorporating ASL script to describe the entire platform power features (see section 3.3.2).

VxWorks also utilizes key concepts of ACPI specification for their power management framework. Similar to Android, VxWorks power manager focus solely on CPU to reduce power usage. Akin to QNX Neutrino, VxWorks framework is embedded into the RTOS. Our framework have been shown to adapt to various RTOS by offering alternative design to complement the RTOS.

## 3.5 ACPI versus RTOS-ACPI

ACPI have been around for a long while and has evolved to handle many situations related to managing power for non-embedded system domain. This was the motivation to use the extensive and well establish framework and extract key design to be adapted for embedded systems platform. Major difference between the two platforms requires many portions of ACPICA to be reviewed and adapted to embedded systems. The table and figure below summarises the key difference between the ACPICA and this implementation of RTOS-ACPI.

*Table 10: Overall difference between ACPICA and RTOS-ACPI*

| ACPICA | RTOS-ACPI |
|---|---|
| OS utilizes ACPI capabilities to control power. | Application controls power through RTOS-ACPI framework. |
| BIOS is ACPI compatible. | No BIOS present in embedded systems. |
| System’s power information is present in the tables supplied by the BIOS to ACPI. | Platform's power information is described in the ASL file. |

| | |
|---|---|
| All devices are present under the system bus tree of the namespace. | Namespace should include additional tree for on-chip peripherals, apart from the external devices tree. |
| Power capabilities are encoded inside the tables. | Power functionalities split between the ASL file and the methods in ACPI subsystem. |
| All hardware devices attached are ACPI-aware. | No specialized hardware for embedded devices as of now. |
| iASL compiler is used. | ASL translator is designed to extract information from ASL file into Namespace. |
| ACPI categories event as *Fixed* or *Generic*. | All events are treated as *Fixed*. |
| ACPI Thermal Model is an important aspect of ACPI specifications. | Thermal management is not considered. |

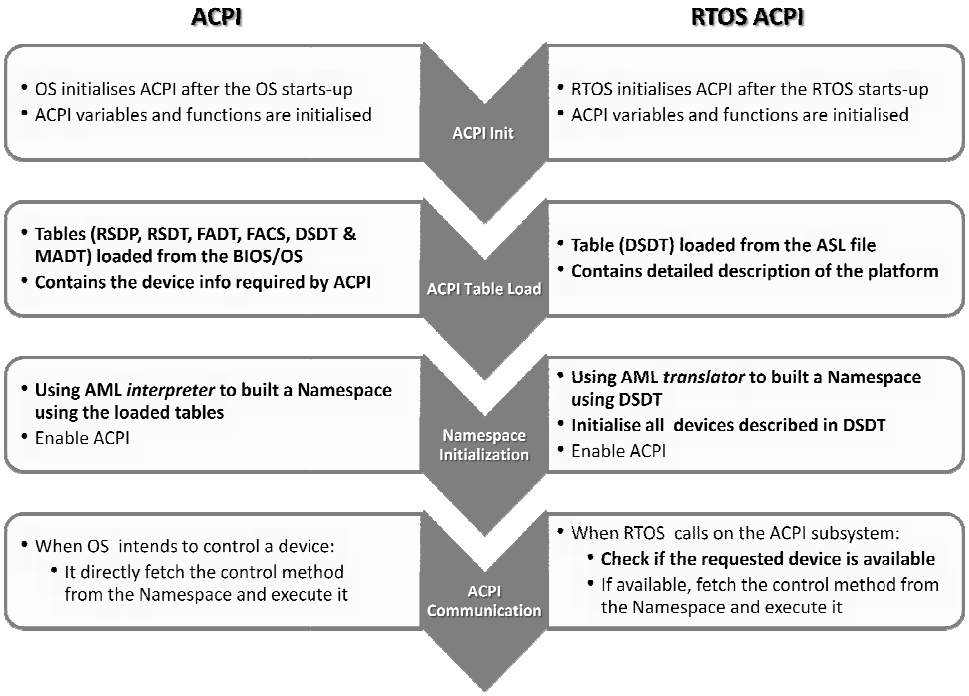



*Figure 31: Difference in control flow between ACPI and RTOS-ACPI*

## 3.6 Key features of this approach

The proposed approach for power management framework would have the following advantages:

1. *Hardware Abstraction*

   The hardware is described independently from the RTOS native language; in the form of ASL tables. Any change in the hardware only requires updating the *ASL table (file)*. Hence the framework can work without any dependency on specific hardware configuration of the system.

2. *Policy Abstraction*

   Power policy refers to the rules according to which the system hardware devices transition among various different power states. The subsystem only provides the infrastructure for the policy manager to implement its decisions.

3. *Standardized Middleware for Power Management*

   The framework is designed to be platform independent. Hence it can be deployed onto most platforms with minimal change in framework. This feature is also seen in ACPICA which provides a similar capability for the PC systems.

4. *Support for Dynamic Power Management*

   The framework is designed to facilitate dynamic power management for embedded system. It does not assume any static set of policies or timed power transition models. The framework is designed to take requests from the policy in run time and act accordingly.

5. *Scalability*

   Changes to the hardware only require details, comprising of the data and methods necessary to control platform's power features in the script-like language. Another facet of scalability is its capability to easily change the power policy to fit the application.

## 3.7 Summary

Exploring ACPICA and proposing a counterpart framework for resource constraint embedded system was one of the main aspects of this project.

In this implementation, T-Kernel RTOS was selected as it has been migrated to various embedded computing platforms. Their standard T-Engine hardware platform also offers a plethora of peripherals which fits in nicely with the power management framework being proposed. Moreover, the T-Kernel RTOS does not have a comprehensive mechanism to handle power management which makes it an ideal candidate as the ACPI-like infrastructure would greatly augment its power management capabilities.

The RTOS-ACPI subsystem designed and implementation was inspired from both ACPI specification and ACPICA reference implementation. Various design considerations were analysed and discussed with reference to embedded system. The subsystem is far from ideal, though it gives a basic infrastructure to support dynamic power management but is adaptable to different embedded systems platform and embedded operating system.

However, infrastructure itself is not sufficient to make a power management framework comprehensive. For the framework to be complete, another important aspect is the power policies. The performance of framework depends on policy's ability and compatibility with the application. To illustrate how the framework can facilitate in power management, a few power policy algorithms would be reviewed and implemented utilising the RTOS-ACPI subsystem. The next chapter to shall discuss and highlight the usage of the power policies in the framework for saving power.

# Chapter 4
# Power Management Policies

*This chapter describes high level policies in power management system. Two power policies we selected to be discussed and implemented to run on the framework; "Predictive Shutdown and Wakeup" & "Adaptive Learning Tree". An alternative approach of application-aware strategy is also discussed.*

## 4.1 Overview of Power Management Policies

Making power management effective requires more than simply turning peripherals on or off. It is necessary for the framework to be flexible. Optimization can be achieved through techniques that can coordinate wake-up sequences instead of turning peripherals on all at once. In a power-managed system, the state of operation of various devices is dynamically adapted to the required performance level, in an effort to minimize the power wasted by idle or underutilized devices.

Unfortunately, there is no policy that performs well in all circumstances. However, a policy can get good performance with specific applications behaviour, but underperforms for others. Thus we can implement a set of policies, and activate the appropriate policy for specific application behaviour. A monitor would be able to keep track of an application's behaviour, and note which policy is most suitable.

## 4.2 Incorporating Power Management Policies

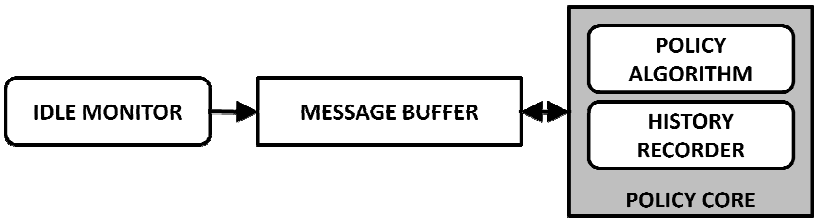



*Figure 32: A simple structure of a power management policy*

The figure above shows current structure of policy. The "*Idle Monitor*" checks the state of the devices and sends a request to "*Message Buffer*" once it detects a change in state. For policies that have pre-wakeup mechanism, when the

wakeup alarm is activated after a time out, a request will be send to the "*Message Buffer*". The "*Policy Core*" would obtain the data from the "*Message Buffer*" for its "*Policy Algorithm*". "*History Recorder*" is used as a temporary storage for the current working policy algorithm. The requests will be processed in, according to the request's type code.

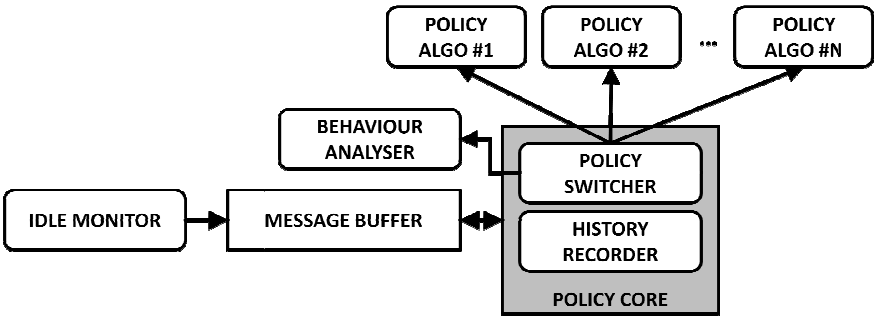



*Figure 33: Structure of a multi-algorithm power management policy*

The figure above shows a possible structure of policy adding a "*Behaviour Analyzer*" that can analyze and identify the behaviour of the current application and switch to the appropriate policy algorithm using the "*Policies Switcher*". The "*Behaviour Analyzer*" will maintain monitoring the application's behaviour which would provide real-time information for the "*Policies Switcher*". When the current policy is no longer the most suitable, it can switch to another more suitable policy. The next section shall look at a few power policy algorithms and attempts to match these policies to a set of applications behaviours.

## 4.3 Power Saving and Real-time Ability

One aspect that needs to be considered when developing policies for embedded system is balance between the amount of power saved and maintaining real-time. Other consideration would be when the power manager decides to switch a device's state from busy to idle; it has to consider the extra power consumption during state transition. The figure below illustrates an example of power consumption in each state and during state transition.

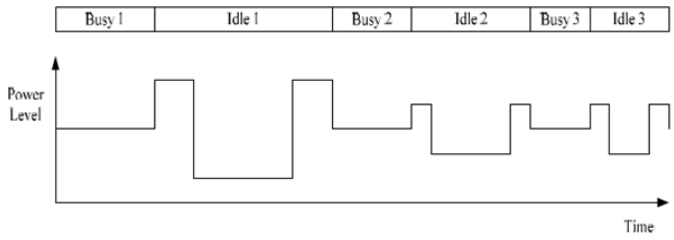



*Figure 34: An example of power consumption graph*

Take the period of *Busy 1* to *Idle 1* as an example. As *Idle 1* reached its predetermined threshold time, the algorithm might determine to switch the device to a deeper sleeping state. The algorithm must carefully consider the overall power consumption when doing a state transition as switching to a deeper sleeping state might incur a higher energy cost as compared to a shallower sleeping state. A typical device is unable to immediately switch to another power state as there is a state transition delay. This state transition would incur more energy. To compensate this extra energy dissipation, the time which the device must stays long enough in the next power state. The following illustrates the situation of state transitions.

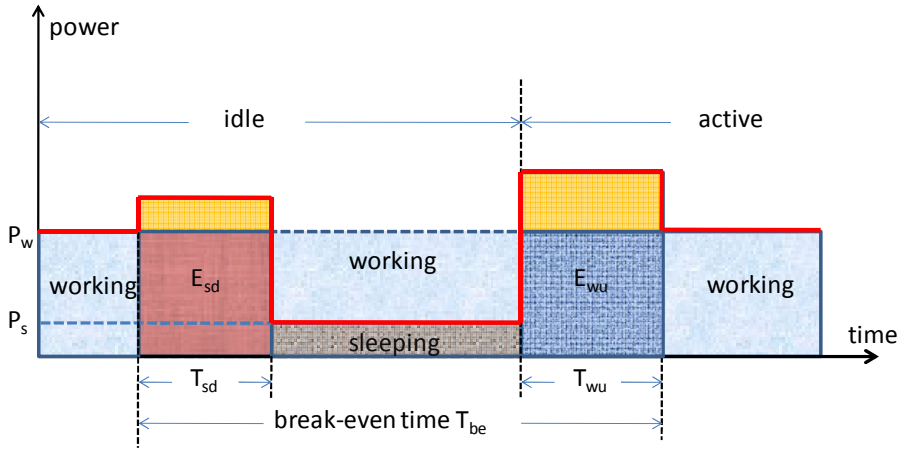



*Figure 35: Power consumption graph illustrating break-even time*

In this picture, $P_w$ and $P_s$ denote the power consumption in working state and sleep state. $E_{sd}$ and $E_{wu}$ represent the energy consumption of shutdown and wakeup. $T_{sd}$ and $T_{wu}$ means the shutdown delay and wakeup delay.

To save power, the idle time must be long enough to compensate the extra energy: $E_{sd}$ and $E_{wu}$. Thus we can derive the following inequality:

$$P_w \cdot t \geq E_{sd} + E_{wu} + P_s \cdot (t - T_{sd} - T_{wu})$$
$$t \geq \frac{E_{sd} + E_{wu} - P_s \cdot (T_{sd} + T_{wu})}{P_w - P_s} \qquad (4.0)$$
$$T_{be} = \frac{E_{sd} + E_{wu} - P_s \cdot (T_{sd} + T_{wu})}{P_w - P_s}$$

We call the minimum length of idle time to achieve power saving as "*break-even time*" $T_{be}$. If we don't consider the drawback of wakeup delay and only see the energy point of view, the system should shutdown when idle period $t$ is longer than $T_{be}$. For the same idle time, a longer break-even time means less power saving due to overhead for recovering state.

If a device gets a request from application just as it is set to low power, the device will have to wake up immediately which will lead to delay penalty. This delay penalty can cause serious performance failure in a real-time embedded system. The challenge is to find the right moment to wake up the device to avoid the delay penalty.

## 4.4 Power Policy Algorithms

Two existing power policy algorithms have been identified and implemented to work on the proposed RTOS based power management framework. The algorithms were tested for its accuracy in predicting idle period in various conditions. The first algorithm, *predictive shutdown and wakeup* was adapted for the proposed framework. The algorithm can effectively predict idle periods in most cases, except the occurrence of impulse-like idle periods (a sudden, very long idle period occurring after continuous, nearly uniform idle periods) during the prediction process [Hwan97]. To alleviate this problem, the algorithm was revised and two variations (*weighted-average of recent history* and *average of recent history*) of the algorithm were tested to observe how they fare in different conditions. The second algorithm, *adaptive learning* [Chun99] was also adapted and implemented onto the proposed framework. These algorithms were then tested with five applications behaviour (see figure 41 &

42). This section demonstrates the flexibility of the proposed framework in quickly adapting and incorporating existing power management algorithms.

***Policy 1 - Predictive Shutdown and Wakeup (Exponential-average approach)***
The minimum required idle period for achieving a power consumption decrease is called break-even time. This policy algorithm attempt to accurately predict the next idle period's length based on previous actual and predicted idle periods. An accurate prediction of idle period can significantly improve a power policy algorithm.

The exponential average approach is used for the prediction of the idle period in CPU scheduling problem. The OS needs to predict the length of the next CPU burst in order to make appropriate process scheduling. The next predicted CPU burst is an accumulative average of the measured lengths of previous CPU bursts. Borrowing this idea, we can predict the next idle period by the accumulative average of the previous idle periods. The recursive prediction formula is shown below:

$$I_{n+1} = a \cdot i_n + (1-a) \cdot I_n \qquad (4.1)$$

Here $I_{n+1}$ is the new predicted value, $I_n$ is the last predicted value, $i_n$ is the latest idle period, and $a$ is a constant attenuation factor in the range between 0 to 1. In this formula, $I_n$ is the inertia and $i_n$ is the force to push the predicted idle period toward the actual idle period. We can use this equation to predict the upcoming idle period, which is a function of the latest idle period and the previous predicted value. The parameter to controls the relative weight of recent and past history in the prediction. If $a = 0$, then $I_{n+1} = I_n$. On the other hand, if a = 1, then $I_{n+1} = i_n$. Equation (4.1) can be expanded to be:

$$I_{n+1} = a \cdot i_n + a(1-a) i_{n-1} + \cdots + a(1-a)^n i_0 + (1-a)^{n+1} I_0 \qquad (4.2)$$

Equation (4.2) indicates that the predicted idle period is the weighted average of previous idle periods. Early idle periods have less weight as specified by the exponential attenuation factors.

*1. Proposed weighted-average of recent history*

Introducing temporal locality into the algorithm, recent idle periods have referential value for next prediction. Predicting next idle period's length using a predetermined number of recent history records. If *m* records were set, the algorithm for the next prediction will be:

$$I_{n+1} = \underbrace{a \cdot i_n + a(1-a) i_{n-1} + \cdots + a(1-a)^{m-1} i_{n-m+1}}_{m \ \ records} \qquad (4.3)$$

As compared to the exponential-average approach, this algorithm would be more robust and able to handle sudden changes in idle period.

*2. Proposed average of recent history*

Building on top of weighted average algorithm, this approach assumes each history record has same weight. Using *m* records, the algorithm for the next prediction will be:

$$I_{n+1} = \underbrace{\frac{1}{m} \cdot i_n + \frac{1}{m} \cdot i_{n-1} + \cdots + \frac{1}{m} \cdot i_{n-m+1}}_{m \ \ records} \qquad (4.4)$$

This is another variation of the weighted-average algorithm and would share the same characteristics, thus capable of handling sudden changes in idle period. However, this algorithm gives equal weight to all idle periods unlike the weighted-average algorithm that gives increasing importance to the most recent idle period.

***Adaptive Learning (Policy 2)***

This policy is based on tree structure (data structure) and is used to predict the most appropriate sleep state at the start of an idle period. It can work with systems or devices with arbitrary number of sleep states.

The adaptive learning tree uses the concept of "*Idle Group*". If *n* is the number of sleep states, then the total number of power states in the device is *n+1*. There would be *n* break-even time, one for each sleep state. The time axis can

be partitioned in *n+1* disjoint intervals, bounded by the break-even time. An idle period *t* can be associated with the index of the power state "*Idle Group*" giving the optimal idle period:

$$IdleGroup(t) = \begin{cases} 0 \quad if \ t < I_0 \\ i+1 \ \ if \ \mathrm{T_{be}}(i) < t < \mathrm{T}(i+1) \\ \mathrm{n} \ \ if \ \mathrm{T_{be}}(n) < t \end{cases} \qquad (4.6)$$

A sequence of idle periods can be transformed into a sequence of integers, which represent the optimal power state for each idle period. Predicting the next *IdleGroup*(*t*) would let the system choose the optimal sleep state. A sequence, *s* has finite length *l* and is denoted as $s_l$. Also, $s_l$ denotes the $i_{th}$ value of the sequence and $s_0$ is the most recent event among all $s_i$'s. The optimal power state for an idle period represented by $s_i$ is *psi*.

A tree consists of decision nodes (circles), history branches (solid lines), prediction branches (dashed lines), and leaf nodes (rectangles). The tree is layered: the top decision node corresponds to $s_0$, nodes in the second level correspond to $s_l$ and so on. All leaf nodes are predictions for next idle period regardless of their ancestor levels. Each leaf stores the Prediction Confidence Level (PCL). The higher the PCL is, the higher the confidence is for a prediction.

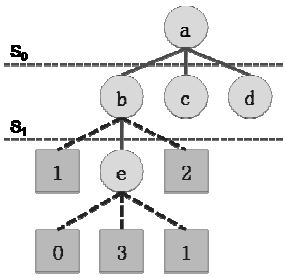



*Figure 36: Adaptive learning tree (with 2 states)*

Each decision node can have both history branches and prediction branches, but the total number of branches is always *n*, and a prediction branch can only be used when the ancestor is decision node is associated with the index of a power state *IdleGroup*(*t*) = {*0,1,…,n*}. From left to right, they are denoted as $b_i$, i=0, 1… n regardless of their types.

The prediction of next idle state is calculated based on a *path matching procedure*. A path for a given sequence, $s_l$ is defined as a series of decision nodes such that from the top node, a history branch $bs_i$, is recursively selected and move to the lower level decision node connected to $bs_i$. The recursion is terminated when the $bs_i$ is a prediction branch or the level of the decision node corresponds to $s_l$ *-1*. Path length (pl) is defined as the number of decision nodes included in the path. While matching the path, the leaf nodes connected to the decision node included in the path are checked and the leaf node which has the highest PCL is selected. When there are multiple leaf nodes which have the same highest PCL, the leftmost leaf node is selected. After path matching, the index of the selected leaf node becomes the prediction for the next event.

It is shown in the figure above that if the sequence $s_2$ = "*01*", the path "*a---b---e*" will be matched. After path matching, the middle leaf node *e,* is selected, thus the tree predicts *IdleGroup*(*t*) = 1 for the next idle period and issues a command to shut down the system to power state 1. Also, when the sequence is $s_2$ = "*00*" or $s_2$ = "*02*", the path "*a---b*" will be matched. Now that node *e* is no longer included in the path any more, the rightmost leaf node *b* is selected. Path length, the number of old events used in decision is varied according to the given sequence. In addition, two different sequences ("*00*" and "*02*") are classified in the same category and can share the resources of the tree to reduce memory usage.

*Learning and Updating*

Other than determining the next prediction, a learning process is needed to maintain the tree to preserve the accuracy of the prediction. Whenever an idle period ends, the tree is updated to reflect the quality of prediction made when the previous idle period occurred. When the prediction is correct, the learning tree should be updated to increase the possibility to choose the same leaf node for the given sequence. This task is achieved by updating the PCL of the leaf nodes. PCL update is controlled by a finite-state machine as shown in figure below.

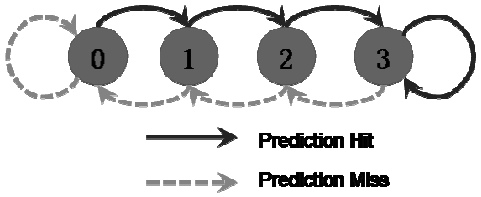



*Figure 37: PCL operation*

When the prediction is correct, the PCL state is changed to the higher state, in the reverse situation, the PCL state is changed to the lower state. And when it reaches either end state, it keeps the current state. Thus, the PCL is an adaptive feature of the learning tree for non-stationary sequences. The adaptive learning tree has insufficient information to distinguish the given sequence from other sequences and the PCL of the leaf node which should have been selected.

A workaround requires two additional procedures to be performed. A desired value $d_v$ = *IdleGroup*(*t*). To improve the algorithms ability to distinguish, increase the path length of the current path by replacing the leaf node on the prediction branch connected to the last decision node in the path with a new decision node. Next, increase the PCL of the desired leaf node, finding all leaf nodes which are connected through the prediction branch $bd_v$ on the path and increase their PCL.

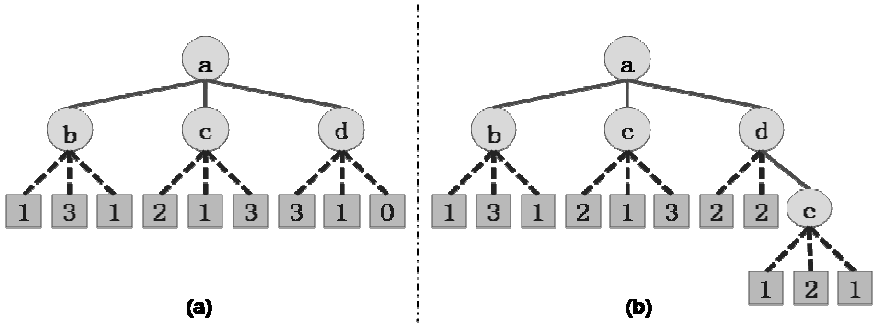



*Figure 38: An example of learning for prediction miss*

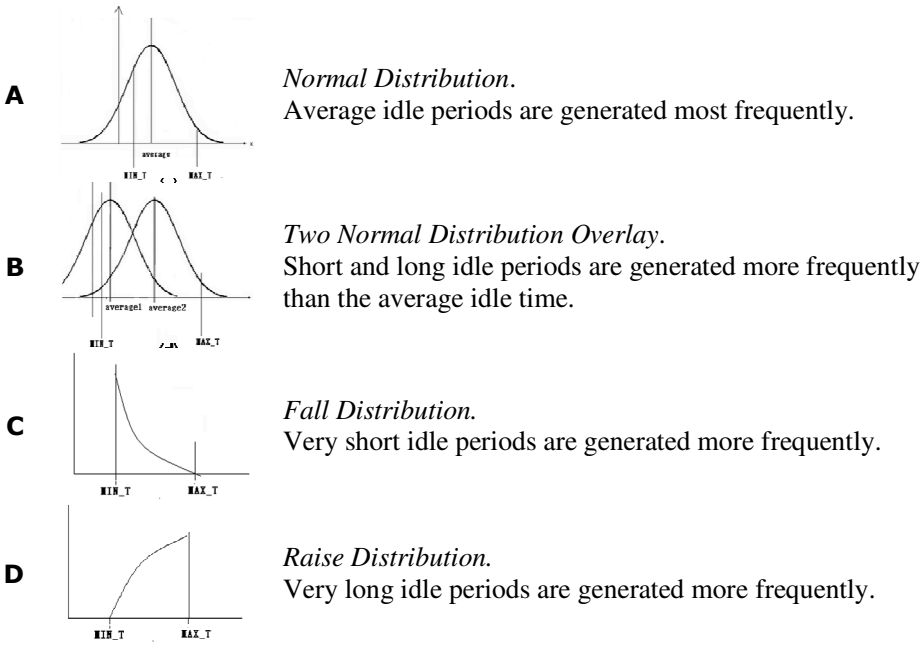
An example of an updating procedure is shown in figure shown above. Suppose there are 3 different sequences such that *A* = "*20*", *B* ="*21*", *C* = "*22*" and the next idle state after A and B should be 0, but the next idle state after *C* is *1*. The path "*a---d*" will be matched for all those sequences in figure (41-a) shown above and the learning tree will predict *0* for every sequence. This

prediction is correct when the given sequence is *A* or *B*; it is wrong when the given sequence is *C*. It is necessary to distinguish sequence *C* from *A* and *B*.

When a miss-prediction occurs, the PCL of the leftmost leaf node of node d is decreased. Then, the rightmost leaf node of node d is replaced with a new decision node because $s_1 = 2$. The leaf nodes of the new decision node have the initial PCL value (in this case, it is *1*). Then, the second additional procedure is applied and the final PCL of leaf nodes are as shown in figure (b) above. These additional procedures causes the adaptive learning tree to grow in an unbalanced manner. This characteristic keeps the tree small and naturally determines the correlation depth between the future idle state and old history depending on the sequence characteristics.

## 4.5 Application Behaviours

Different kinds of testing applications are used to measure the system's performance. The testing applications have idle periods with different lengths which are randomly generated. Various distributions of idle periods are designed to test policies' performance under different circumstance. Common applications behaviours have been modelled as various distributions as shown below:




*Normal Distribution*.
Average idle periods are generated most frequently.

*Two Normal Distribution Overlay*.
Short and long idle periods are generated more frequently than the average idle time.

*Fall Distribution.*
Very short idle periods are generated more frequently.

*Raise Distribution.*
Very long idle periods are generated more frequently.

*Figure 39: Various Distribution to simulate application behaviour*

As shown below, an additional behaviour in which idle periods are not generated randomly, but in burst-like behaviour is designed to simulate network application. The application generates 5 short idle periods and 5 long idle periods which is repeated.

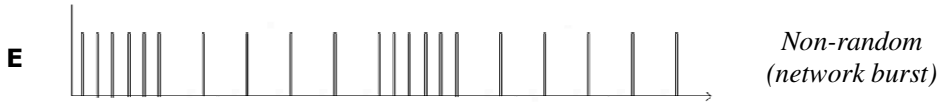



*Figure 40: Burst behaviour for application*

## 4.6 Experiments, Results and Analysis (*predictive algorithms*)

***Experiment Setup***

The use of random distribution is to test the algorithm's worst case performance. A simple application is used to opens and use a device. After a predetermined time, it would closes the device and wait for a random amount of time (follow the various random distributions in figure 41). This is procedure is repeated 1000 times.

***Results & Analysis for Predictive Shutdown and Wakeup (Policy 1)***

*Table 11: Probability of saving power for predictive shutdown & wakeup algorithms*

| ***Predictive Shutdown and Wakeup Algorithms*** | **Normal Distribution** | **2 Normal Distribution Overlay** | **Fall Distribution** | **Raise Distribution** | **Network Burst** |
|---|---|---|---|---|---|
| ***Exponential-average (a = 0.5)*** $I_{n+1} = a \cdot i_n + (1-a) \cdot I_n$ | 9.5 | 19.5 | 8.9 | 56.7 | 77.7 |
| ***Exponential-average ($a_1$ = 0.5, $a_2$ = 0.25)*** $I_{n+1} = a_1 . i_n + (1 - a_2) . I_n$ | 9.5 | 20.0 | 9.1 | 57.5 | 77.5 |
| ***Weighted-average (a = 0.5, m = 5)*** $I_{n+1} = \underbrace{a \cdot i_n + a(1-a) i_{n-1} + \cdots + a(1-a)^{m-1} i_{n-m+1}}_{m \ records}$ | 22.1 | 53.2 | 27.1 | 58.9 | 57.9 |
| ***Average (a = 0.5, m = 5)*** $I_{n+1} = \underbrace{\frac{1}{m} \cdot i_n + \frac{1}{m} \cdot i_{n-1} + \cdots + \frac{1}{m} \cdot i_{n-m+1}}_{m \ records}$ | 20.4 | 53.7 | 27.0 | 58.6 | 58.9 |

***Observation***

The table shown above summarises the results (detailed results in Appendix G) of the experiment. It shows the probability of switching to low power during potential power savings. Potential power savings occur when the idle time is greater than the break-even time (see figure 26).

The following can be observed:

- Algorithm performs poorly in random behaviours.
- Exponential-average yield best probability for network burst behaviour.
- Exponential-average yields the worst probability for fall distribution application behaviour (frequently have idle time > break-even time).
- Weighted-average and Average algorithms tend to perform better then exponential-average algorithms in most random situations.

***Analysis of results***

Analysing the data obtained, it can be inferred:

- Overall increase in power consumption if high frequency of idle time that is shorter then break-even time.
- Significant decrease in power consumption if high frequency of idle time that is longer than break-even time.
- Mild increase in power consumption if high frequency of idle time that is the same as the break-even time.
- Network burst showed good results as its behaviours are non-random, thus having a deterministic pattern that the policy is able to identify and take advantage of.
- Generally, it can be inferred that the performance in random condition is not optimal. This is expected as there is no deterministic pattern among any randomly generated idle periods which works against the heuristic characteristic of this policy algorithm. Conversely, this is the policy's lower bound performance as random condition is the worst case situation.

***Results for Adaptive Learning (Policy 2)***

*Table 12: Probability of saving power for adaptive learning algorithm*

| ***Adaptive Learning Algorithms*** | **Normal Distribution** | **2 Normal Distribution Overlay** | **Fall Distribution** | **Raise Distribution** | **Network Burst** |
|---|---|---|---|---|---|
| $IdleGroup(t) = \begin{cases} 0 \quad if \ t < I_0 \\ i+1 \ \ if \ \ \mathrm{T}_{be}(i) < t < \mathrm{T}(i+1) \\ \mathrm{n} \ \ if \ \ \mathrm{T}_{be}(n) < t \end{cases}$ | 48.9 | 51.0 | 48.0 | 52.0 | 99.0 |

***Observation***

The table above summarises the results (detailed results in Appendix G) of the experiment for adaptive learning algorithm. It shows the probability of switching to low power during potential power savings. Potential power savings occur when the idle time is greater than the break-even.

The following can be observed:

- The policy performs poorly in random behaviours (prediction hit-ratio are around 50%).
- Excellent performance in network burst (non-random) behaviours.

***Analysis***

- Poor prediction mechanism of the policy due to it looking for regular patterns which are nonexistent in random situations.
- In non-random behaviours, most of the predictions are correct. This is because the network burst behaviour have clear pattern. This allows the strength of the policy to find the patterns and make more accurate predictions.

## 4.7 Application-aware Strategy

Power policies that are used to match application that *fit* their behaviour would still be limited to the unpredictable behaviour of an application. Application can react to outside stimuli that would sometimes change its general behaviour. This might affect policy predictions that use historical events which would lead to a decrease in performance.

Another proposed approach would be to identify key characteristics of an application. This can be done by profiling the application so that the system would know exactly where and when to toggle the devices; *hotspots*.

### 4.7.1 Overview of Existing Profile-based Application-aware Analysis

One method of identifying these *hotspots* is to use profile-based analysis. This methodology has been detailed in the following master's thesis [Mage09]. This section would take certain portions of the thesis to give the reader a basic understanding of the methodology.

*Control Flow Graph (CFG)*

The CFG is a fundamental data structure needed by almost all static and dynamic analyses techniques. The CFG is represented through basic blocks which their connecting edges. This helps see the flow between the blocks thereby providing an idea about the possible flows of execution for a program.

*Control Flow Analysis for I/O Power Management*

This approach attempts to utilize the control flow profile of an embedded code to take power management decisions of the peripherals rather than the CPU. Focus would be on static analysis techniques assuming uninterrupted program execution on a single processor. A CFG would represent an embedded application where certain nodes of the graph represent CPU-related operations and others represent I/O device related operations.

From such an application profile, given the estimates of least possible execution times of the nodes of the CFG, one can estimate the possible idle-time durations

between successive I/O requests. Such control flow based timing information gives a predictable knowledge about the operations of the devices. Given the peripheral power consumption parameters, suitable decisions can be taken for both performance-centric functioning of the peripherals as well as to put them in low power mode for profitable energy savings.

*Identifying and selecting hotspots*

The following is the control-flow based approach:

1. The CFG representing an embedded application needs to be analyzed offline using suitable algorithms that can estimate the potential idle time durations. An I/O device when it is required of its service needs to be turned ON at appropriate moment without neglecting performance.

2. Suitable locations in the application have to be chosen for switching it ON. Similarly, when potential idle time durations are detected, the device can be turned OFF for profitable energy savings. Such locations in the application are appropriate for switching it OFF.

3. The algorithm has to determine the placement of Switching Points at suitable locations. The basic inputs that are needed are the estimates of least possible execution times of basic blocks in a CFG (at any abstraction level) and peripheral parameters such as Turn-ON time and Break-even time.

4. The algorithm would iteration through the application's CFG many times till the ideal *hotspots* that would profit both performance and energy savings are identified.

5. The extend of power saved would be limited to how well the *hotspots* are identified/selected from the CFG. Ideal *hotspots* would be identified if accurate device specifications and well-written application is supplied to the algorithm.

### 4.7.2 Incorporating *hotspots* into application

Typically, the application would require access to the software drivers to change the power state of the devices. This method increases the complexity of the operation as developers are required to write their additional functions to actuate the devices at the designated *hotspots.* In addition, they would have to consider issues like device contention and real-time constraints.

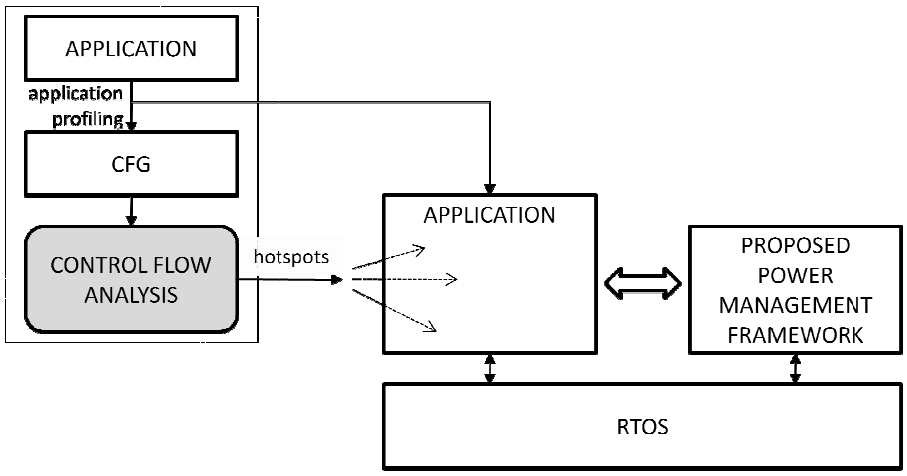



*Figure 41: Incorporating hotspots into application using the proposed framework*

To solve this problem, the identified *hotspots* are inserted into the application by accessing the proposed power management framework. Each *hotspot* requires a system call (include the device name and power state) to the framework to change the power state of the device. This differs from the previous power policy algorithms (see section 4.4) which is encapsulated in a subsystem to access the proposed power management framework.

## 4.8 Overall Analysis of Policy Algorithms

The testing applications help to identify the policies performance under different idle periods. It can be observed that random distribution which has too many short idle periods are not easy to predict. On the other hand, if most of the idle periods are long periods, the prediction would be more accurate.

As mentioned in the previous sections that utilizing random distributions aids in highlighting the policy's performance lower bound (worst case situation). *Predictive shutdown and wakeup* policy tend to perform well if the idle periods are locally correlated, but will perform poorly if vice-versa. *Adaptive learning* policy tend to show excellent performance if there are many regular patterns in an application's behaviour, but will crumble under a random condition.

It is also observed that application with the same general behaviour but with slight variations can influence the way the policy's performance. Thus, further work could be done to investigate the characteristics of these policies and learn what variables in the algorithm can affect its result and outcome. Moreover, using these data obtained, it is possible to create a module that would switch to the appropriate power policy depending on the system's behaviour.

Another aspect would be to do a thorough application profiling which would identify the *hotspots* for switching the state of the devices. The current algorithm would only perform well if the application fall under the few behavioural patterns that works for them. Power saving would therefore be limited or even nonexistent if the application does not fall under the behavioural pattern discussed earlier.

This method would generally yield better and more accurate predictions as the offline algorithm can see the entire application from start to end and make the most appropriate decisions to reduce power. However, it is noted that the profiling currently works for a single application and is currently not capable to handle multiple applications.

Alternately, a detailed offline profile of the application would aid in selecting an appropriate policy. Additional information can be extracted from profiling the application to provide the algorithm with parameters specific to the application which could improve its performance.

## 4.9 Summary

This chapter shows two heuristic power policy algorithms implemented on the power management subsystem. Predictive shutdown algorithm was adapted from a CPU scheduling to predicting idle time. Adaptive learning algorithm derived from tree data structure has also been presented. A critical analysis and testing of these policies was undertaken which identifies the limitations of these algorithms. Thereafter, the power policy was profiled to identify its optimal application behaviour. This was followed by matching the power policy algorithm with common application behaviours. A novel offline profiling approach was also shown to illustrate the flexibility of the framework to handle static and dynamic power management policies.

# Chapter 5
# Conclusion and Recommendations

## 5.1 Conclusion

Power management on embedded systems is an important and widely researched topic that can benefit from a standardised framework on embedded OS. In this thesis, novel techniques based on Advanced Configuration and Power Interface (ACPI) have been proposed for incorporating power management feature into RTOS. Noting that the current power management features in embedded OS support only primitive power management features that are mostly tightly coupled with power-policy algorithms, emphasis was given to the development of a comprehensive power management framework that could be adapted to various RTOS.

The proposed power management framework was implemented on Renesas SH7727 T-Engine platform running T-Kernel RTOS to evaluate its suitability to an embedded system environment with real-time constraints. The framework comprises of two parts; RTOS-ACPI subsystem and power policy algorithms.

The framework allows the developers with the flexibility to incorporate application-specific power management strategies into a variety of RTOS. This was made possible due to the systematic incorporation of ACPI subsystems into the RTOS framework such that it can be adapted to support any power management policy with ease. In particular, the blueprint for RTOS-ACPI subsystem was adapted from ACPICA but its implementation was tailored for embedded OS. The ACPI specifications were examined in details to facilitate the establishment of the RTOS-ACPI subsystem for managing the power features in the system.

One of the main features involves the incorporation of various granularity of power down modes, which describe the configuration of each power down modes in a plaintext form. In addition, transition timing from one power state to another can also be appended in order to assist the development of a more

advanced power policy. Such an advanced power policy would also be able to engender more reliable decisions, such as: switching to a lesser degree of low power state that has a shorter latency during shorter idle period to help maximise power savings.

Two different power policy algorithms, namely Predictive Shutdown and Wakeup and Adaptive Learning were implemented and it was demonstrated that potential savings could be realized when tested with certain application behaviours. This confirmed that a hybrid policy could be relied upon to tackle various application behaviours in real-time. It was also shown that tweaking the variables in the power policy algorithms can influence the performance of the policy.

The two heuristic based power policies have been shown to sporadically make wrong predictions which can affect the real-time system. This problem was circumvented with a few simple rules, for example: checking if the device is currently being used before putting it onto low power and checking before changing a USB bus into a low power state when a USB device is still running. Moreover, to facilitate future expansion of rules, a Basic Policy Manager has been included into the RTOS-ACPI subsystem to house all these essential rules. We have also shown that the characteristics of a power policy can be refined by examining the application behaviour in order to further improve the performance of the power policy algorithm.

Further work has also carried out to show that offline analysis of the application software can help to encapsulate the application characteristics into the RTOS-ACPI subsystem. Preliminary investigations show that it can significantly reduce the overhead on the system as all the calculations and predictions were performed offline. However, this method requires the application to be thoroughly profiled to extract runtime characteristics.

We have shown that a well developed power policy can manage the system's power without any changes to the application. The proposed framework can be incorporated into RTOS without any changes to its core kernel, albeit minor changes to the device drivers were necessary to inform the RTOS-ACPI

subsystem when a device is being accessed by an application. Such modifications can be merged with the device manager module to entirely remove the need to make future amendments to the core kernel.

In conclusion, a systematic framework for incorporating power management feature into a real-time operating system has been proposed. The proposed framework is based on ACPI concepts to introduce several attributes for enhancing the power management on real-time operating systems. This has led to the standardization of incorporating power management with platform abstraction. Secondly, it abstracts the power policies from the application to facilitate system-wide power management. Finally, the proposed framework provides for a comprehensive and flexible power management support in RTOS based embedded systems.

## 5.2 Recommendations for future work

While we were able to propose solutions for many outstanding issues in RTOS-based power management, there is still scope for future work in this area. We discuss these avenues for future research work in this section.

- **Optimizing the power policy algorithms:** In this research work, we have shown that the characteristics of a power policy can be refined by examining the application behaviour in order to further improve the performance of the power policy algorithm. This is shown by tweaking the variables in the power policy algorithms that influences the performance of the policy. (see table 11). Further research into how the application characteristics would influence the power policy would be useful as it will provide for additional information to tune the policy algorithm in a reliable manner.
- **Switching power policy algorithm:** In this work, we found that power policy algorithm performs differently for different application behaviours. It would be interesting to extend this technique to select a suitable power policy algorithm for each application behaviour. A application monitoring module could be developed to switch between these power policy algorithm in real-time to adapt to the changes in the application's behaviours.

- **Subsystem for application-aware strategy**: The application-aware strategy required the identified *hotspots* to be embedded into the applications via direct calls to the proposed framework. We assumed that the developer would have a good understanding on using the framework and inserting the calls into the application. Alternatively, these identified *hotspots* could be incorporated during compilation. This would require work on making changes to the compiler. This would provide a seamless means in embedding the *hotspots* without making any changes to the application.
- **Merging RTOS's device manager into the framework**: It has been shown that the proposed framework can be incorporated into RTOS with minor changes to each device drivers to inform the framework when a device is being accessed by an application. Such modifications can be merged with the RTOS's device manager to entirely remove the need to make future amendments to the core kernel and device drivers.

# References

[Alte00b] Altera Corporation. "Nios Soft Core Embedded Processor Data Sheet", June 2000.

[Alte02a] Altera Corporation. "Nios Embedded Processor Development Board - Data Sheet", April 2002.

[Burd96] T. Burd and R. Brodersen. “Processor design for portable systems,” *Journal of VLSI Signal Processing*, 13(2-3):203-222, 1996.

[Beni98a] L. Benini, G.D. Micheli, E. Macii, D. Sciuto and C. Silvano. "*Address bus encoding techniques for system-level power optimization*," IEEE, 1998.

[Beni98b] L. Benini and G.D. Micheli, “*Dynamic Power Management: Design Techniques and CAD Tools*,” Kluwer, 1998.

[Beni99] L. Benini, A. Bogliolo, G.A. Paleologo, and G.D. Micheli, “Policy Optimization for Dynamic Power Management,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.18 ,no.6, pp.813-833*, June 1999.

[Beni00] L. Benini, et al., “*A Survey of Design Techniques for System-Level Dynamic Power Management*,” IEEE Transactions on Very Large Scale Integration Systems (VLSI), Vol. 8, No. 3, June 2000.

[Birc08] W.L. Bircher and L.K. John, "Analysis of dynamic power management on multi-core processors," *In International Conference on Supercomputing*, pg. 327-338, 2008.

[Cant05] Canturk Isci, Alper Buyuktosunoglu and M. Martonosi. “Long-term workload phases: Duration predictions and applications to DVFS,” *IEEE Micro, 25(5):39–51,* September 2005.

[Cant06] Canturk Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture (HPCA’06)*, February 2006.

[Chan96] A. Chandrakasan, V. Gutnik and T. Xanthopoulos, “Data Driven Signal Processing: An Approach for Energy Efficient Computing,” International Symposium on Low Power Electronics and Design pp. 347-352, Aug. 1996.

[Chan97] J. Chang and M. Pedram. “Energy minimization using multiple supply voltages,” *In International Symposium on Low Power Electronics and Design (ISLPED-96)*, pages 157-162, August 1996.

[Chri06] Chris Gniady, Ali R. Butt, Y. Charlie Hu, and Yung-Hsiang Lu. “Program Counter Based Prediction Techniques for Dynamic Power Management,” *IEEE Transactions on Computers*, 55(6):641–658, June 2006.

[Chun99] E.-Y. Chung, L. Benini, and G.D. Micheli, “Dynamic Power Management Using Adaptive Learning Tree*,” Proc. Int’l Conf. Computer-Aided Design,* Nov. 1999.

[Chun02] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G.D. Micheli, “Dynamic Power Management for Nonstationary Service Requests,” *IEEE Transactions on Computers, vol. 51, no. 11, pp. 1345-1361,* Nov. 2002.

[Dong03] Dongkun Shin and Jihong Kim,"Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," *International Symposium on Low Power Electronics and Design*, pp. 408-413, Korea, 2003.

[Doug95] F. Douglis, P. Krishnan, B. Bershad, “Adaptive Disk Spin-down Policies for Mobile Computers,” *USENIX Symposium on Mobile and Location-Independent Computing*, pp. 121-137, Apr. 1995.

[Gius03] Giuseppe Ascia, Vincenzo Catani, Maurizio Palesi, Antonio Parlato, "An evolutionary approach for reducing the energy in address buses," *In ACM International Conference Proceeding Series*, vol. 49, pg. 76-81, 2003.

[Gold96] R Golding, P. Bosh and J. Wilkes, “Idleness is not Sloth,” *HP Laboratories Technical Report HPL*, pg. 96-140, 1996.

[Govi95] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed- setting of a low-power CPU. *In the 1st ACM International Conference on Mobile Computing and Networking (MOBICOM-95)*, pages 13-25, November 1995.

[Heal04] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini, “Code Transformations for Energy-Efficient Device Management”, *IEEE Transactions on Computers,*Aug. 2004.

[Helm96] D. Helmbold, D. Long, E. Sherrod, “Dynamic Disk Spin-down Technique for Mobile Computing,” *Conference on Mobile Computing*, pp. 130-142, Nov. 1996.

[Henk07] J. Henkel, S. Parameswaran, and N. Cheung, "Application-Specific Embedded Processors," *Designing Embedded Processors – A Low Power Perspective*, pg. 3-23, 2007.

[Himp09] HP, Intel, Microsoft, Phoenix and Toshiba, “*Advanced Configuration and Power Interface Specification 4.0*,” http://acpi.info/spec.htm (Last accessed on Jun 22, 2009).

[Hong99] I. Hong, M. Potkonjak And R. Karri, "Power Optimization Using Divide-and-Conquer Techniques for Minimization of the Number of Operations," *ACM Transactions on Design Automation of Electronic Systems*, vol. 4, no. 4, pp. 405 - 429, October 1999.

[Hung06] Hung Cheng Shih and Kuochen Wang, "An Adaptive Hybrid Dynamic Power Management Method for Handheld Devices," *Proceedings of*

*the IEEE Int'l Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing,* 2006.

[Hwan97] C. Hwang, A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event Driven Computation," *Int'l Conference on Computer Aided Design*, pp. 28-32, Nov. 1997.

[Inte03] Intel, "*ACPI Component Architecture Programmer Reference*," http://acpica.org/documentation/ (Last accessed on Sep 22, 2008).

[Ishi98] T. Ishihara and H. Yasuura. "Voltage scheduling problem for dynamically variable voltage processors," *In Int'l Symposium on Low Power Electronics and Design (ISLPED-98),* pages 197-202, August 1998.

[Jinf01] Jinfeng Liu, Pai H. Chou, Nader Bagherzadeh and Fadi Kurdahi, "A constraint-based application model and scheduling techniques for power-aware systems," *International Conference on Hardware Software Codesig*n, pp 153-158, Denmark, 2001.

[Jong07] Jong-Eun Lee, Kiyoung Choi, and Nikil D. Dutt, "Synthesis of Instruction Sets for High-Performance and Energy-Efficient ASIP," Designing Embedded Processors – A Low Power Perspective, pg. 51-64, 2007.

[Kada05] I. Kadayif, M. Kandemir, G. Chen, N. Vijaykrishnan, M. J. Irwin and A. Sivasubramaniam, "Compiler-directed high-level energy estimation and optimization," ACM Transactions on Embedded Computing Systems (TECS), vol. 4, no. 4, pp. 819-850, November 2005.

[Kris95] P. Krishnan, P. Long, J. Vitter, "Adaptive Disk Spindown Via Optimal Rent-to-buy in Probabilistic Environments," *Int'l Conference on Machine Learning*, pp. 322-330, July 1995.

[Labr99a] Labrosse J J, "MicroC/OS-II: the real-time kernel", Lawrence, Kansas R& D Publication, 1999.

[Labr00a] Labrosse J J, "*Official website of MicroC/OS-II*", see: http://www.ucos-ii.com/

[Lehr09] Rick Lehrbaum, "*Using Linux in Embedded Systems and Smart Devices*", see: http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Using-Linux-in-Embedded-Systems-and-Smart-Devices/ (Last accessed on Oct 23, 2009).

[Lorc98] J. Lorch and A. Smith, "Software Strategies for Portable Computer Energy Management," *IEEE Personal Communications*, vol. 5, no. 3, pp. 60-73, June 1998.

[Luyh00a] Y.-H. Lu, E.Y. Chung, T. Simunic, L. Benini and G. De Micheli, "Quantitative Comparison of Power Management Algorithms," DATE, *Proceedings of Design Automation and Test in Europe*, March 2000.

[Luyh00b] Y.-H. Lu, L. Benini, and G.D. Micheli, “Operating-system directed power reduction,” *Proc. Int. Symposium Low Power Electronic Design, Rapallo,Italy*, July 2000.

[Luyh00c] Y.-H. Lu, G.D. Micheli, and L. Benini, “Requester-Aware Power Reduction,” *Proc. Int’l Symp. System Synthesis,* Sept. 2000.

[Mage09] Kannan, Magesh, "Control Flow Analysis for Dynamic Power Management", 2010.

[Much97] S. Muchnick. “*Advanced Compiler Design and Implementation*,” Morgan Kaufmann Publishers, Inc., 1997.

[Nich07] Nicholas H. Zamora, Jung-Chun Kao and Radu Marculescu, "Distributed Power-Management Techniques for Wireless Network Video Systems," *Design, Automation, and Test in Europe*, pp. 564-569, France, 2007.

[Ohpe98] J. Oh and M. Pedram, “Gated clock routing minimizing the switched capacitance," *Design Automation and Test in Europe Conference*, pp. 692-697, Feb. 1998.

[Oztu06] O. Ozturk, G. Chen, M. Kandemir and M. Karakoy, "Cache miss clustering for banked memory systems," *International Conference on Computer Aided Design*, pp. 244-250, 2006.

[Para07] S. Parameswaran1, J. Henkel, A. Janapsatya, T. Bonny and A. Ignjatovic, "Design and Run Time Code Compression for Embedded Systems," *Designing Embedded Processors – A Low Power Perspective*, pg. 97-130, 2007.

[Paul07] JoAnn M. Paul and Brett H. Meyer, "Power-Performance Modelling and Design for Heterogeneous Multiprocessors," *Designing Embedded Processors: A Low Power Perspective*, pg. 423-448, Springer, 2007.

[Peri98] T. Pering and R. Brodersen. “Energy efficient voltage scheduling for real time operating systems,” *In 4th IEEE Real-Time Technology and Applications Symposium (RTAS-98),* 1998.

[Pute94] M. Puterman, *Finite Markov Decision Processes*, John Wiley and Sons, 1994.

[Qiup99] Q. Qiu and M. Pedram, “Dynamic Power Management Based on Continuous-Time Markov Decision Processes,” *Proc. Design Automation Conference,* June 1999.

[Rafa96] Rafael Peset Llopis and Manoj Sachdev. “Low Power, Testable Dual Edge Triggered Flip-Flops,” *In International Symposium on Low Power Electronics and Design (ISLPED-96)*, pages 341-345, August 1996.

[Ross97] S. Ross, *Introduction to Probability models*, 6th ed. Academic Press, 1997.

[Sama08] Samarjit Chakraborty and Ye Wang, "Multimedia power management on a platter: from audio to video & games," *International Multimedia Conference*, pp. 1165-1166, 2008.

[Sher09] Sheridan Ethier, "*Implementing Power Management on the Biscayne SH7760 Reference Platform Using the QNX® Neutrino® RTOS*," http://www.qnx.com/developers/articles/article_296_2.html (Last accessed on Oct 11, 2009).

[Sriv96] M. Srivastava, A. Chandrakasan and R. Brodersen, "Predictive System Shutdown and other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 42-55, March 1996.

[Stro00] A.G.M. Strollo, E. Napoli, and D. De Caro, "*New clock-gating techniques for low-power flip-flops*," *In International Symposium on Low Power Electronics and Design (ISLPED-00)*, pages 114-119, 2000.

[Tefs03] T-Engine Forum Specification, *T-Format (3): Global Symbol Naming Rule in C Language,* Revision 01.01.00.

[Tiwa96] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. "Instruction level power analysis and optimization of software," *Journal of VLSI Signal Processing,* 13(2/3): pg 1-18, 1996.

[Uhri04] Sascha Uhrig and Theo Ungerer, "Fine-grained power management for multithreaded processor cores," *In Symposium on Applied Computing*, pg. 907-908, 2004.

[Weis94] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. 1st USENIX Symposium on Operating Systems Design and Implementation*, pp. 12-23, Nov. 1994.

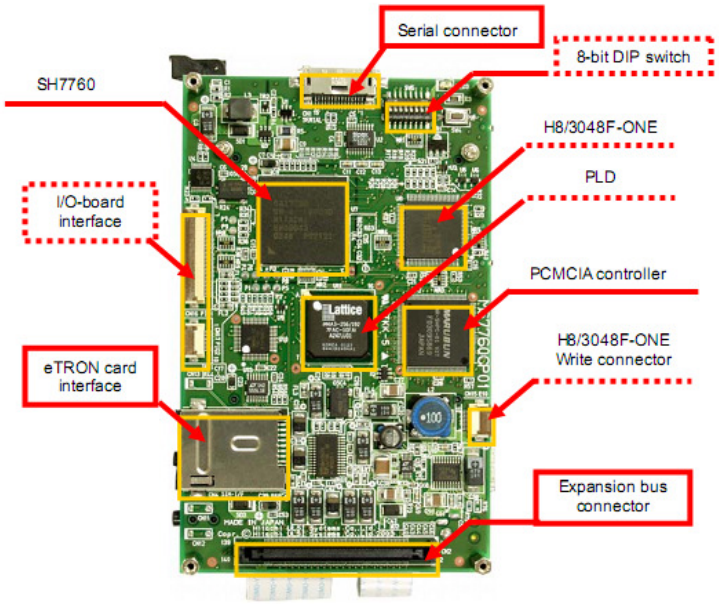
[Wiki09] Wikipedia, "*Embedded Linux*", http://en.wikipedia.org/wiki/Embedded_Linux (Last accessed on Oct 23, 2009).

[Yagh03] Karim Yaghmour, "*Building Embedded Linux Systems*", O'Reilly & Associates Inc., USA, Apil 2003.

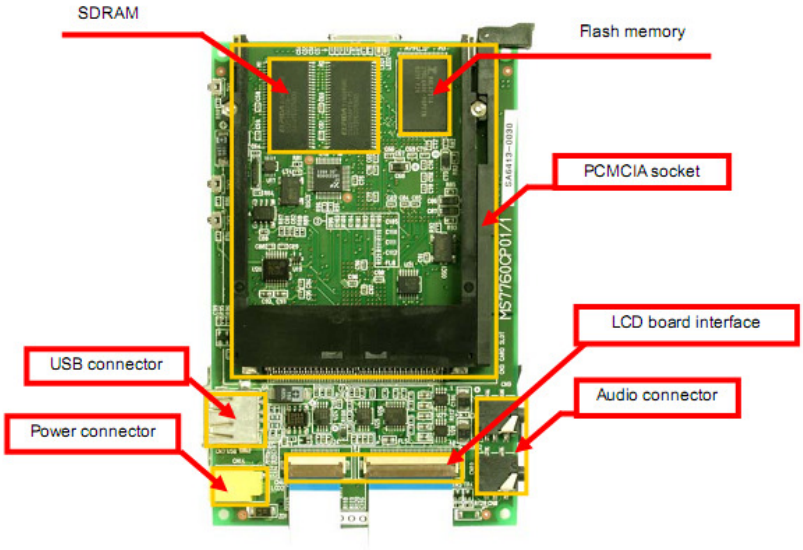
[Yung00] Yung-Hsiang Lu, Luca Benini and Giovanni De Micheli,"Low-power task scheduling for multiple devices," *International Conference on Hardware Software Codesign*, USA, 2000.

# Appendix A: Renesas SH7727 Specifications

| Item | Specifications | Target device |
|---|---|---|
| CPU | SH7727<br>Model name: HD6417727BP160CV<br>(Renesas Technology)<br>Input clock: 12MHz<br>Operating clock (Internal): 96MHz (x 8)<br>(External): 48MHz (x 4) | |
| Flash memory | Capacity: 8MB<br>S29JL064H70TF1000 (Spansion) x 1 | |
| SDRAM | Capacity: 64MB<br>MT48LC16M16A2P-75 (Micron) x 2 | |
| PCMCIA Card I/F | One slot<br>Controller:MR-SHPC-01 V2T-F (Marubun) | |
| Serial I/F | 2ch<br>Controller: XR16L2550IM-F (EXAR) | ChA: H8/3048F-ONE I/F<br>ChB: Monitor for debugging |
| Sound | Model name: AK4550VTP<E2>P<br>(Asahi Kasei)<br>Earphone/microphone: 1ch<br>Headphone output: 1ch<br>- Microphone input<br>Impedance: 2.2Ω<br>Sensitivity: 51dB/Pa<br>- Headphone output<br>Impedance: 32Ω | |
| USB Host | 1ch<br>Controller: SH7727 on-chip | |
| TFT color LCD module | LS037V7DW01(SHARP)<br>Display color: 262,144 colors<br>Display area: 240(H) x 320(V)<br>Controller:SH7727 on-chip LCDC | |
| Power supply controller | H8/3048F-ONE<br>Model name: HD64F3048BVTF25V<br>(Renesas Technology)<br>Operating frequency: 7.3728MHz | The control SH7727 working for power supply control, RTC, or tablet interface infrared remote control must be interfaced via the serial chA. |
| RTC | Model name: RV5C348B<E2>-F (RICOH) | Via the H8/3048F-ONE |
| Touch panel I/F | Model name: ADS7843E (TI) | Via the H8/3048F-ONE<br>(To be mounted on the LCD board) |
| Serial EEPROM | Capacity: 512 bytes<br>Model name: BR93L66FJ-W (ROHM) | Via the H8/3048F-ONE |
| Infrared remote control | Transmission<br>Model name: GL390 (SHARP)<br><br>Reception<br>Model name: GP1US301XP (SHARP)<br>Transmission carrier: 38KHz | Via theH8/3048F-ONE |




*T-Engine SH7727 board CPU Side*




*T-Engine SH7727 Board Underside*

# Appendix B: Lex and Yacc Tools

*Lex* and *Yacc* can generate program fragments that discover a structure from a source program and later generate target program. The task of discovering the source structure again is decomposed into subtasks:

1. Split the source file into tokens (*Lex*).
2. Find the hierarchical structure of the program (*Yacc*).

**LEX (Lexical Analyzer Generator)**

Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream. It is well suited for editor-script type transformations and for segmenting input in preparation for a parsing routine.

Lex source is a table of regular expressions and corresponding program fragments. The table is translated to a program which reads an input stream, copying it to an output stream and partitioning the input into strings which match the given expressions. As each such string is recognized the corresponding program fragment is executed. The recognition of the expressions is performed by a deterministic finite automaton generated by Lex. The program fragments written by the user are executed in the order in which the corresponding regular expressions occur in the input stream.

**YACC (Yet Another Compiler-Compiler)**

Computer program input generally has some structure; in fact, every computer program that does input can be thought of as defining an ``input language'' which it accepts. An input language may be as complex as a programming language, or as simple as a sequence of numbers. Unfortunately, usual input facilities are limited, difficult to use, and often are lax about checking their inputs for validity.

Yacc provides a general tool for describing the input to a computer program. The Yacc user specifies the structures of his input, together with code to be invoked as each such structure is recognized. Yacc turns such a specification into a subroutine that handles the input process; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine.

# Appendix C: ASL Complete Reference

| SNo | Operator Name | Description |
|---|---|---|
| 1 | Acquire | Acquire a mutex |
| 2 | Add | Integer Add |
| 3 | Alias | Define a name alias |
| 4 | And | Integer Bitwise And |
| 5 | ArgX | Method argument data objects |
| 6 | BankField | Declare fields in a banked configuration object |
| 7 | Break | Continue following the innermost enclosing While |
| 8 | BreakPoint | Used for debugging, stops execution in the debugger |
| 9 | Buffer | Declare Buffer object |
| 10 | Case | Expression for conditional execution |
| 11 | Concatenate | Concatenate two strings, integers or buffers |
| 12 | ConcatenateResTemplate | Concatenate two resource templates |
| 13 | CondRefOf | Conditional reference to an object |
| 14 | Continue | Continue innermost enclosing While loop |
| 15 | CopyObject | Copy and existing object |
| 16 | CreateBitField | Declare a bit field object of a buffer object |
| 17 | CreateByteField | Declare a byte field object of a buffer object |
| 18 | CreateDWordField | Declare a DWord field object of a buffer object |
| 19 | CreateField | Declare an arbitrary length bit field of a buffer object |
| 20 | CreateQWordField | Declare a QWord field object of a buffer object |
| 21 | CreateWordField | Declare a Word field object of a buffer object |
| 22 | DataTableRegion | Declare a Data Table Region |
| 23 | Debug | Debugger output |
| 24 | Decrement | Decrement an Integer |
| 25 | Default | Default execution path in Switch() |
| 26 | DefinitionBlock | Declare a Definition Block |
| 27 | DerefOf | Dereference an object reference |
| 28 | Device | Declare a bus/device object |
| 29 | Divide | Integer Divide |
| 30 | DMA | DMA Resource Descriptor macro |
| 31 | DWordIO | DWord IO Resource Descriptor macro |
| 32 | DWordMemory | DWord Memory Resource Descriptor macro |
| 33 | DWordSpace | DWord Space Resource Descriptor macro |
| 34 | EisaId | EISA ID String to Integer conversion macro |
| 35 | Else | Alternate conditional execution |
| 36 | ElseIf | Conditional execution |
| 37 | EndDependentFn | End Dependent Function Resource Descriptor macro |
| 38 | Event | Declare an event synchronization object |
| 39 | ExtendedIO | Extended IO Resource Descriptor macro |
| 40 | ExtendedMemory | Extended Memory Resource Descriptor macro |

| | | |
|---|---|---|
| 41 | ExtendedSpace | Extended Space Resource Descriptor macro |
| 42 | External | Declare external objects |
| 43 | Fatal | Fatal error check |
| 44 | Field | Declare fields of an operation region object |
| 45 | FindSetLeftBit | Index of first least significant bit set |
| 46 | FindSetRightBit | Index of first most significant bit set |
| 47 | FixedIO | Fixed I/O Resource Descriptor macro |
| 48 | FromBCD | Convert from BCD to numeric |
| 49 | Function | Declare control method |
| 50 | If | Conditional execution |
| 51 | Include | Include another ASL file |
| 52 | Increment | Increment a Integer |
| 53 | Index | Indexed Reference to member object |
| 54 | IndexField | Declare Index/Data Fields |
| 55 | Interrupt | Interrupt Resource Descriptor macro |
| 56 | IO | IO Resource Descriptor macro |
| 57 | IRQ | Interrupt Resource Descriptor macro |
| 58 | IRQNoFlags | Short Interrupt Resource Descriptor macro |
| 59 | LAnd | Logical And |
| 60 | LEqual | Logical Equal |
| 61 | LGreater | Logical Greater |
| 62 | LGreaterEqual | Logical Not less |
| 63 | LLess | Logical Less |
| 64 | LLessEqual | Logical Not greater |
| 65 | LNot | Logical Not |
| 66 | LNotEqual | Logical Not equal |
| 67 | Load | Load differentiating definition block |
| 68 | LoadTable | Load Table from RSDT/XSDT |
| 69 | LocalX | Method local data objects |
| 70 | LOr | Logical Or |
| 71 | Match | Search for match in package array |
| 72 | Memory24 | Memory Resource Descriptor macro |
| 73 | Memory32 | Memory Resource Descriptor macro |
| 74 | Memory32Fixed | Memory Resource Descriptor macro |
| 75 | Method | Declare a control method |
| 76 | Mid | Return a portion of buffer or string |
| 77 | Mod | Integer Modulo |
| 78 | Multiply | Integer Multiply |
| 79 | Mutex | Declare a mutex synchronization object |
| 80 | Name | Declare a Named object |
| 81 | NAnd | Integer Bitwise Nand |
| 82 | NoOp | No operation |
| 83 | NOr | Integer Bitwise Nor |
| 84 | Not | Integer Bitwise Not |

| | | |
|---|---|---|
| 85 | Notify | Notify Object of event |
| 86 | ObjectType | Type of object |
| 87 | One | Constant One Object -1 |
| 88 | Ones | Constant Ones Object (-1) |
| 89 | OperationRegion | Declare an operational region |
| 90 | Or | Integer Bitwise Or |
| 91 | Package | Declare a package object |
| 92 | PowerResource | Declare a power resource object |
| 93 | Processor | Declare a processor package |
| 94 | QWordIO | QWord IO Resource Descriptor macro |
| 95 | QWordMemory | QWord Memory Resource Descriptor macro |
| 96 | QWordSpace | Qword Space Resource Descriptor macro |
| 97 | RefOf | Create Reference to an object |
| 98 | Register | Generic register Resource Descriptor macro |
| 99 | Release | Release a synchronization object |
| 100 | Reset | Reset a synchronization object |
| 101 | ResourceTemplate | Resource to buffer conversion macro |
| 102 | Return | Return from method execution |
| 103 | Revision | Constant revision object |
| 104 | Scope | Open named scope |
| 105 | ShiftLeft | Integer shift value left |
| 106 | ShiftRight | Integer shift value right |
| 107 | Signal | Signal a synchronization object |
| 108 | SizeOf | Get the size of a buffer, string, or package |
| 109 | Sleep | Sleep n milliseconds (yields the processor) |
| 110 | Stall | Delay n microseconds (does not yield the processor) |
| 111 | StartDependentFn | Start Dependent Function Resource Descriptor macro |
| 112 | StartDependentFnNoPri | Start Dependent Function Resource Descriptor macro |
| 113 | Store | Store object |
| 114 | Subtract | Integer Subtract |
| 115 | Switch | Select code to execute based on expression value |
| 116 | ThermalZone | Declare a thermal zone package. |
| 117 | Timer | Get 64-bit timer value |
| 118 | ToBCD | Convert Integer to BCD |
| 119 | ToBuffer | Convert data type to buffer |
| 120 | ToDecimalString | Convert data type to decimal string |
| 121 | ToHexString | Convert data type to hexadecimal string |
| 122 | ToInteger | Convert data type to integer |
| 123 | ToString | Copy ASCII string from buffer |
| 124 | ToUUID | Convert Ascii string to UUID |
| 125 | Unicode | String to Unicode conversion macro |
| 126 | Unload | Unload definition block |
| 127 | VendorLong | Vendor Resource Descriptor |
| 128 | VendorShort | Vendor Resource Descriptor |

| | | |
|---|---|---|
| 129 | Wait | Wait on an Event |
| 130 | While | Conditional loop |
| 131 | WordBusNumber | Word Bus number Resource Descriptor macro |
| 132 | WordIO | Word IO Resource Descriptor macro |
| 133 | WordSpace | Word Space Resource Descriptor macro |
| 134 | Xor | Integer Bitwise Xor |
| 135 | Zero | Constant Zero object 0 |

# Appendix D: ASL Operators and Keywords

*ASL Operators*

| **REGISTER (Generic Register Resource Descriptor Macro)** | |
|---|---|
| Syntax | Register(AddressSpaceKeyword, RegisterBitWidth, RegisterBitOffset, RegisterAddress, AccessSize, DescriptorName) |
| Description | AddressSpaceKeyword specifies the address space where the register exists. The register can exist in I/O space (SystemIO), memory (SystemMemory), PCI configuration space (PCI_Config), embedded controller space (EmbeddedControl), SMBus (SMBus) or fixed-feature hardware (FFixedHW).<br>RegisterBitWidth evaluates to an 8-bit integer that specifies the number of bits in the register.<br>RegisterBitOffset evaluates to an 8-bit integer that specifies the offset in bits from the start of the register indicated by RegisterAddress.<br>RegisterAddress evaluates to a 64-bit integer that specifies the register address. The 64-bit field<br>DescriptorName. _ADR is automatically created in order to refer to this portion of the resource descriptor.<br>AccessSize evaluates to an 8-bit integer that specifies the size of data values used when accessing the address space as follows:<br>0-Undefined (legacy)<br>1-Byte access<br>2-Word access<br>3-Dword access<br>4-Qword access<br><br>*DescriptorName is an optional argument that specifies a name for an integer constant that will be created in the current scope.* |

| **NAME (Declare Named Object)** | |
|---|---|
| Syntax | Name(ObjectName, Object) |
| Arguments | Creates a new object named ObjectName. Attaches Object to ObjectName in the Global ACPI namespace. |
| Description | Creates ObjectName in the namespace, which references the Object. |
| Example | *The following example creates the name PTTX in the root of the namespace that references a package.*<br>Name (\PTTX, // Port to Port Translate Table<br>Package () {Package () {0x43, 0x59}, Package) {0x90, 0xFF}}<br>)<br>*The following example creates the name CNT in the root of the namespace that references an integer data object with the value 5.*<br>Name (\CNT, 5) |

| **SCOPE (Open Named Scope)** | |
|---|---|
| Syntax | Scope (Location) {ObjectList} |

| | |
|---|---|
| Arguments | Opens and assigns a base namespace scope to a collection of objects. All object names defined within the scope act relative to Location. Notice that Location does not have to be below the surrounding scope, but can refer to any location within the namespace. The term Scope itself does not create objects, but only locates objects in the namespace; the located objects are created by other ASL terms. |
| Description | The object referred to by Location must already exist in the namespace and be one of the following object types that have a namespace scope associated with it:<br>• Predefined scope such as: \ (root), _SB, \GPE, _PR, _TZ, etc.<br>• Device<br>• Processor<br>• Thermal Zone<br>• Power Resource |

| **EXTERNAL (Declare External Objects)** | |
|---|---|
| Syntax | External (ObjectName, ObjectType, ReturnType, ParameterTypes) |
| Arguments | ObjectName is a NameString.<br>ObjectType is an optional ObjectTypeKeyword. If not specified, "UnknownObj" type is assumed.<br>ReturnType is optional. If the specified object type is MethodObj, then this specifies the type or types of object returned by the method. If the method does not return an object, then nothing is specified or UnknownObj is specified. To specify a single return type, simply use the ObjectTypeKeyword (e.g. IntObj, PkgObj, etc.). To specify multiple possible return types, enclose the comma-separated ObjectTypeKeywords with braces. For example: {IntObj, BuffObj}.<br>ParameterTypes is optional. If the specified object type is MethodObj, this specifies both the number and type of the method parameters. It is a comma-separated, variable-length list of the expected object type or types for each of the method parameters, enclosed in braces. For each parameter, the parameter type consists of either an ObjectTypeKeyword or a comma-separated sub-list of ObjectTypeKeywords enclosed in braces. There can be no more than seven parameters in total. |
| Description | The External compiler directive is to let the assembler know that the object is declared external to this table so that the assembler will not complain about the undeclared object. During compiling, the assembler will create the external object at the specified place in the namespace (if a full path of the object is specified), or the object will be created at the current scope of the External term. |

| **AND (Integer Bitwise And)** | |
|---|---|
| Syntax | And (Source1, Source2, Result) => Integer |
| Arguments | Source1 and Source2 are evaluated as Integers. |
| Description | A bitwise AND is performed and the result is optionally stored into Result. |

| **OR (Integer Bitwise Or)** | |
|---|---|
| Syntax | Or (Source1, Source2, Result) => Integer |
| Arguments | Source1 and Source2 are evaluated as Integers. |
| Description | A bitwise OR is performed and the result is optionally stored in Result. |

| **DEVICE (Declare Bus/Device Package)** | |
|---|---|
| Syntax | Device (DeviceName) {ObjectList} |
| Arguments | Creates a Device object of name Device Name, which represents either a bus or a device or any other similar hardware. Device opens a name scope. |
| Description | A Bus/Device Package is one of the basic ways the Differentiated Definition Block describes the hardware devices in the system to the operating software. Each Bus/Device Package is defined somewhere in the hierarchical namespace corresponding to that device's location in the system. Within the namespace of the device are other names that provide information and control of the device, along with any sub-devices that in turn describe sub-devices, and so on. |

| **METHOD (Declare Control Method)** | |
|---|---|
| Syntax | Method (MethodName) |
| Arguments | Creates a new control method of name MethodName.<br>MethodName is evaluated as a Namestring data type. |
| Description | Declares a named package containing a series of object references that collectively represent a control method, which is a procedure that can be invoked to perform computation. Method opens a name scope. System software executes a control method by referencing the objects in the package in order.<br>The current namespace location used during name creation is adjusted to be the current location on the namespace tree. Any names created within this scope are "below" the name of this package. The current namespace location is assigned to the method package, and all namespace references that occur during control method execution for this package are relative to that location. |
| Example | *The following block of ASL sample code shows a use of Method for defining a control method that turns on a power resource.*<br>Method (_ON) {<br>Store (One, GIO.IDEP) // assert power<br>Sleep (10) // wait 10ms<br>Store (One, GIO.IDER) // de-assert reset#<br>Stall (10) // wait 10us<br>Store (Zero, GIO.IDEI) // de-assert isolation<br>} |

*RTOS-ACPI ASL Keywords*

| No. | Keywords | Description |
|---|---|---|
| 1 | _SB | System Bus |
| 2 | _DSS | An object under Device scope. Must be the first object in the device. Holds Device current State, Initialized to 0x63 if it is a hot-pluggable device, yet to be attached, 0x3otherwise. This object is updated when the RTOS-ACPI subsystem is initialized and continually updated to hold the current power state of the device. |
| 3 | _USE | An object under Device scope. Must be the second object in the device. Holds the data whether the device is currently being used by any application task. |
| 3 | _S0D | An object under Device scope. Holds the state the device is to transit to when the system state is S0 |
| 4 | _S1D | An object under Device scope. Holds the state the device is to transit to when the system state is S1 |
| 5 | _S2D | An object under Device scope. Holds the state the device is to transit to when the system state is S2 |
| 6 | _S3D | An object under Device scope. Holds the state the device is to transit to when the system state is S3 |
| 7 | _PS0 | An object under Device scope. Name of the control method that transits the device to power state D0 |
| 8 | _PS1 | An object under Device scope. Name of the control method that transits the device to power state D1 |
| 9 | _PS2 | An object under Device scope. Name of the control method that transits the device to power state D2 |
| 10 | _PS3 | An object under Device scope. Name of the control method that transits the device to power state D3 |
| 11 | _PSC | An object under Device scope. Name of the control method that evaluates the current power state of the system by reading the device registers |
| 12 | _CS0 | An object under Device scope. Name of the control method that evaluates whether the device is in power state D0. Used internally by the RTOS-ACPI subsystem |
| 13 | _CS1 | An object under Device scope. Name of the control method that evaluates whether the device is in power state D1. Used internally by the RTOS-ACPI subsystem |
| 14 | _CS2 | An object under Device scope. Name of the control method that evaluates whether the device is in power state D2. Used internally by the RTOS-ACPI subsystem |
| 15 | _CS3 | An object under Device scope. Name of the control method that evaluates whether the device is in power state D3. Used internally by the RTOS-ACPI subsystem |
| 16 | _PTT | An object under Device scope. Name of the control method that contains objects that define the power state transition times. |
| 17 | PTT_0 | An object under Device scope. Name of the object that holds the transition times from state D0 to other |

| | | |
|---|---|---|
| | | states as an array of 4 numbers representing time in milliseconds. |
| 18 | PTT_1 | An object under Device scope.<br>Name of the object that holds the transition times from state D1 to other states as an array of 4 numbers representing time in milliseconds. |
| 19 | PTT_2 | An object under Device scope.<br>Name of the object that holds the transition times from state D2 to other states as an array of 4 numbers representing time in milliseconds. |
| 20 | PTT_3 | An object under Device scope.<br>Name of the object that holds the transition times from state D3 to other states as an array of 4 numbers representing time in milliseconds. |

# Appendix E: Sample DSDT Table

```
Scope (_SB) {
    Device (KBPD) {
        Name (_DSS, 0x3)
        Name (_USE, 0)
        Name ( _ADR, 0x2000)
        Name ( _S0D, 0x0)
        Name ( _S1D, 0x0)
        Name ( _S2D, 0x1)
        Name ( _S3D, 0x3)
        Register (SystemIO, 8, 0, 0x0020, 8, TPLC)
        Register (SystemIO, 8, 0, 0x0021, 8, TPLS)
        Register (SystemIO, 8, 0, 0x0022, 8, TPLR)
        External (H8_WRITE, MethodObj, IntObj, 3, IntObj, IntObj, IntObj)
        External (H8_READ, MethodObj, IntObj, 2, IntObj, IntObj)
        External (H8_RESET, MethodObj, IntObj, 0, NULL)
        Method (_PS0 ) {
            H8_RESET ()
            H8_WRITE( TPLC, 1, 0x0F)
            H8_WRITE( TPLR, 1, 0x04)
        }
        Method (_PS1) {
            H8_RESET ()
            H8_WRITE( TPLC, 1, 0x0F)
            H8_WRITE( TPLR, 1, 0x10)
        }
        Method (_PS2) {
            H8_RESET ()
            H8_WRITE( TPLC, 1, 0x0F)
            H8_WRITE( TPLR, 1, 0x80)
        }
        Method (_PS3) {
            H8_RESET ()
            H8_WRITE( TPLC, 1, 0x00)
        }
        Method (_PSC) {
            H8_RESET ()
            H8_READ (TPLC, 1)
            H8_READ (TPLR, 1)
        }
        Method (_CS0) {
            AND (TPLC, 0x0000000F, 0x0F)
            AND (TPLR, 0x000000FF, 0x04)
        }
        Method (_CS1){
            AND (TPLC, 0x0000000F, 0x0F)
            AND (TPLR, 0x000000FF, 0x10)
        }
        Method (_CS2){
            AND (TPLC, 0x0000000F, 0x0F)
            AND (TPLR, 0x000000FF, 0x80)
        }
        Method (_CS3) {
            AND (TPLC, 0x0000000F, 0x0)
```

```
                }
                Method (_PTT) {
                        PTT_0(0, 5, 10, 15)
                        PTT_1(5, 0, 10, 15)
                        PTT_2(10, 5, 0, 15)
                        PTT_3(5, 15, 10, 0)
                }
        }
        Device (LCD0) {
                Name (_DSS, 0x3)
                Name (_USE, 0)
                Name ( _S0D, 0x0)
                Name ( _S1D, 0x1)
                Name ( _S2D, 0x3)
                Name ( _S3D, 0x3)
                Name (_STS, 2)
                Register (SystemIO, 8, 0, 0x00A1, 8, LCDR)
                Method (_PS0) {
                        H8_RESET ()
                        H8_WRITE( LCDR, 1, 0x01)
                }
                Method (_PS3) {
                        H8_RESET ()
                        H8_WRITE( LCDR, 1, 0x00)
                }
                Method (_PSC){
                        H8_RESET ()
                        H8_READ(LCDR, 1)
                }
                Method (_CS0) {
                        AND (LCDR, 0x01, 0x01)
                }
                Method (_CS3){
                        AND (LCDR, 0xFF, 0x0)
                }
        }
}
Scope (_OCP) {
        Device (AFEI) {
                Name (_DSS, 0x3)
                Name (_USE, 0)
                Register (SystemIO, 8, 0, 0xFFFFFF82, 8, STB1)
                Register (SystemIO, 8, 0, 0xFFFFFF88, 8, STB2)
                Register (SystemIO, 8, 0, 0xA4000230, 8, STB3)
                External (REG_WR, MethodObj, IntObj, 4, IntObj, IntObj, IntObj,
IntObj)
                External (REG_RD, MethodObj, IntObj, 3, IntObj, IntObj, IntObj)
                Name ( _S0D, 0x0)
                Name ( _S1D, 0x0)
                Name ( _S2D, 0x0)
                Name ( _S3D, 0x3)
                Method (_PS0){
                        REG_WR (STB3, 8, 5, 0)
                }
                Method (_PS3){
```

```
                REG_WR (STB3, 8, 5, 1)
            }
            Method (_PSC){
                REG_RD (STB3, 8)
            }
            Method (_CS0){
                AND (STB3, 0x20, 0x0)
            }
            Method (_CS3){
                OR (STB3, 0xDF, 0xFF)
            }
        }
        Device (DAC0) {
            Name (_DSS, 0x3)
            Name (_USE, 0)
            Register (SystemIO, 8, 0, 0xFFFFFF82, 8, STB1)
            Register (SystemIO, 8, 0, 0xFFFFFF88, 8, STB2)
            Name ( _S0D, 0x0)
            Name ( _S1D, 0x0)
            Name ( _S2D, 0x2)
            Name ( _S3D, 0x3)
            Method (_PS0){
                REG_WR (STB2, 8, 3, 0)
            }
            Method (_PS3){
                REG_WR (STB2, 8, 3, 1)
            }
            Method (_PSC){
                REG_RD (STB2, 8)
            }
            Method (_CS0){
                AND (STB2, 0x08, 0x0)
            }
            Method (_CS3){
                OR (STB2, 0xF7, 0xFF)
            }
        }
}
Scope (_PR0){
        Device (CPU0){
            Name (_DSS, 0x3)
            Name (_USE, 0)
            Name ( _S0D, 0x0)
            Name ( _S1D, 0x0)
            Name ( _S2D, 0x3)
            Name ( _S3D, 0x3)
            Register (SystemIO, 16, 0, 0xFFFFFF80, 16, FRQC)
            Method (_PS0){
                REG_WR (FRQC, 16, 0x0112)
            }
            Method (_PS3){
                REG_WR (FRQC, 16, 0x0111)
            }
            Method (_PSC){
                REG_RD (FRQC, 16)
```

```
                }
                Method (_CS0){
                        AND (FRQC, 0x0112, 0x0112)
                }
                Method (_CS3){
                        AND (FRQC, 0x0111, 0x0111)
                }
        }
        Device (H8P){
                Name (_DSS, 0x3)
                Name (_USE, 0)
        }
}
Scope (_USB)
{
        Device (USB0) {
                Name (_DSS, 0x3)
                Name (_USE, 0)
                Name ( _S0D, 0x0)
                Name ( _S1D, 0x0)
                Name ( _S2D, 0x3)
                Name ( _S3D, 0x3)
                Register (SystemIO, 8, 0, 0xA4000230, 8, STB3)
                Method (_PS0){
                        REG_WR (STB3, 8, 4, 0)
                        REG_WR (STB3, 8, 3, 0)
                }
                Method (_PS3){
                        REG_WR (STB3, 8, 4, 1)
                        REG_WR (STB3, 8, 3, 1)
                }
                Method (_PSC){
                        REG_RD (STB3, 8)
                }
                Method (_CS0){
                        AND (STB3, 0x18, 0x0)
                }
                Method (_CS3){
                        OR (STB3, 0xE7, 0xFF)
                }
        }
        Device (mouse){
                Name (_DSS, 0x63)
                Name (_USE, 0)
                Method (_PS0){
                        AND (STB3, 0x18, 0x0)
                }
                Method (_PS3){
                        OR (STB3, 0xE7, 0xFF)
                }
        }
}
                Name ( _S1D, 0x0)
                Name ( _S2D, 0x3)
                Name ( _S3D, 0x3)
```

```
                Register (SystemIO, 8, 0, 0xA4000230, 8, STB3)
                Method (_PS0){
                        REG_WR (STB3, 8, 4, 0)
                        REG_WR (STB3, 8, 3, 0)
                }
                Method (_PS3){
                        REG_WR (STB3, 8, 4, 1)
                        REG_WR (STB3, 8, 3, 1)
                }
                Method (_PSC){
                        REG_RD (STB3, 8)
                }
                Method (_CS0){
                        AND (STB3, 0x18, 0x0)
                }
                Method (_CS3){
                        OR (STB3, 0xE7, 0xFF)
                }
        }
        Device (mouse){
                Name (_DSS, 0x63)
                Name (_USE, 0)
                Method (_PS0){
                        AND (STB3, 0x18, 0x0)
                }
                Method (_PS3){
                        OR (STB3, 0xE7, 0xFF)
                }
        }
}
```

# Appendix F: Namespace Structure

```
struct GlobalStructure
{
        int iNumberOfScopes;
        struct Scope *sScopes;
};

struct Scope
{
        char *Name;
        int NumDev;
        struct Device *devDevices;
};

struct Name
{
        char *Name;
        unsigned long iValue;
};

struct Method
{
        char *cMethodName;
        int NumSttmnts;
        struct Function *fFunctions;
};

struct Function
{
        char *sFunctionName;
        int iNumOfArguments;
        unsigned long *Args;
};

struct Device
{
        char *sName;
        int iNumOfNames;
        struct Name *nDeviceName;
        int NumMeth;
        struct Method *maMethods;
};
```

# Appendix G: Detail Results of Policy Algorithms

### *Result Format*

| Ideal State | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|
| Sleep | Sleep | **Total Power Saved** | Early | **Real-time Maintained** |
| | | | Late | **Real-time Compromised** |
| | Normal | **No Power Saved** | - | - |
| Normal | Sleep | **Power Lost and Real-time Compromised** | - | - |
| | Normal | **No Power Saved** | - | - |

*Format #1 of the statistical table*

| Ideal State | Actual State | Percentage |
|---|---|---|
| Sleep | Sleep | **Save Power & Real-time Maintained** |
| | Normal | **No Power Saved** |
| Normal | Sleep | **Lose Power & Real-time Compromised** |
| | Normal | **No Power Saved** |

*Format #2 of the statistical table*

** The Adaptive Learning policy does not have a wakeup policy, thus the table column is removed. 'Normal' relates to the device being in a turned-on state.*

### *Results for Predictive Shutdown and Wakeup (Policy 1)*

*Predictive shutdown & wakeup policy under **normal distribution (A)***

***a = 0.5***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (29.4%) | Sleep | **4.1%** | Early | **2.8%** |
| | | | | Late | **1.3%** |
| | | Normal | 25.3% | - | - |
| Normal | (70.6%) | Sleep | **7.7%** | - | - |
| | | Normal | 62.9% | - | - |

***a1 = 0.5, a2 = 0.25***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (29.4%) | Sleep | **3.8%** | Early | **2.8%** |
| | | | | Late | **1.0%** |
| | | Normal | 25.6% | - | - |
| Normal | (70.6%) | Sleep | **7.6%** | - | - |
| | | Normal | 63.0% | - | - |

***weighted average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (29.4%) | Sleep | **8.2%** | Early | **6.5%** |
| | | | | Late | **1.7%** |
| | | Normal | 21.2% | - | - |
| Normal | (70.6%) | Sleep | **14.8%** | - | - |
| | | Normal | 55.8% | - | - |

***average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (29.4%) | Sleep | **7.7%** | Early | **6.0%** |
| | | | | Late | **1.7%** |
| | | Normal | 21.7% | - | - |
| Normal | (70.6%) | Sleep | **15.0%** | - | - |
| | | Normal | 55.6% | - | - |

*Predictive shutdown & wakeup policy under **2-normal distribution overlay (B)***

***a = 0.5***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (58.8%) | Sleep | **14.9%** | Early | **11.5%** |
| | | | | Late | **3.4%** |
| | | Normal | 43.9% | - | - |
| Normal | (41.2%) | Sleep | **10.1%** | - | - |
| | | Normal | 31.1% | - | - |

***a1 = 0.5, a2 = 0.25***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (58.8%) | Sleep | **14.7%** | Early | **11.8%** |
| | | | | Late | **2.9%** |
| | | Normal | 44.1% | - | - |
| Normal | (41.2%) | Sleep | **10.7%** | - | - |
| | | Normal | 30.5% | - | - |

***weighted average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (58.4%) | Sleep | **41.6%** | Early | **31.1%** |
| | | | | Late | **10.5%** |
| | | Normal | 16.8% | - | - |
| Normal | (41.6%) | Sleep | **28.9%** | - | - |
| | | Normal | 12.7% | - | - |

***average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (58.4%) | Sleep | **42.3%** | Early | **31.4%** |
| | | | | Late | **10.9%** |
| | | Normal | 16.1% | - | - |
| Normal | (41.6%) | Sleep | **29.6%** | - | - |
| | | Normal | 12.0% | - | - |

*Predictive shutdown & wakeup policy under **fall distribution (C)***

***a = 0.5***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (35.7%) | Sleep | **4.2%** | Early | **3.2%** |
| | | | | Late | **1.0%** |
| | | Normal | 31.5% | - | - |
| Normal | (64.3%) | Sleep | **6.8%** | - | - |
| | | Normal | 57.5% | - | - |

***a1 = 0.5, a2 = 0.25***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (36.0%) | Sleep | **4.1%** | Early | **3.3%** |
| | | | | Late | **0.8%** |
| | | Normal | 31.9% | - | - |
| Normal | (64.0%) | Sleep | **6.4%** | - | - |
| | | Normal | 57.6% | - | - |

***weighted average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (35.3%) | Sleep | **13.1%** | Early | **9.6%** |
| | | | | Late | **3.5%** |
| | | Normal | 22.2% | - | - |
| Normal | (64.7%) | Sleep | **20.8%** | - | - |
| | | Normal | 43.9% | - | - |

***average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (35.5%) | Sleep | **12.9%** | Early | **9.6%** |
| | | | | Late | **3.3%** |
| | | Normal | 22.6% | - | - |
| Normal | (64.5%) | Sleep | **20.7%** | - | - |
| | | Normal | 43.8% | - | - |

*Predictive shutdown & wakeup policy under* ***raise distribution (D)***

***a = 0.5***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (89.0%) | Sleep | **75.5%** | Early | **50.5%** |
| | | | | Late | **25.0%** |
| | | Normal | 13.5% | - | - |
| Normal | (11.0%) | Sleep | **9.4%** | - | - |
| | | Normal | 1.6% | - | - |

***a1 = 0.5, a2 = 0.25***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (88.9%) | Sleep | **71.5%** | Early | **51.2%** |
| | | | | Late | **20.3%** |
| | | Normal | 17.4% | - | - |
| Normal | (11.1%) | Sleep | **9.3%** | - | - |
| | | Normal | 1.8% | - | - |

***weighted average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (88.7%) | Sleep | **88.1%** | Early | **52.3%** |
| | | | | Late | **35.8%** |
| | | Normal | 0.6% | - | - |
| Normal | (11.3%) | Sleep | **11.1%** | - | - |
| | | Normal | 0.2% | - | - |

***average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (89.1%) | Sleep | **88.1%** | Early | **52.3%** |
| | | | | Late | **35.8%** |
| | | Normal | 1.0% | - | - |
| Normal | (10.9%) | Sleep | **10.6%** | - | - |
| | | Normal | 0.3% | - | - |

*Predictive shutdown & wakeup policy under* ***non-random network burst***

***a = 0.5***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (49.9%) | Sleep | **39.0%** | Early | **38.8%** |
| | | | | Late | **0.2%** |
| | | Normal | 10.9% | - | - |
| Normal | (50.1%) | Sleep | **28.9%** | - | - |
| | | Normal | 21.2% | - | - |

***a1 = 0.5, a2 = 0.25***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (49.9%) | Sleep | **38.7%** | Early | **38.7%** |
| | | | | Late | **0%** |
| | | Normal | 11.2% | - | - |
| Normal | (50.1%) | Sleep | **29.2%** | - | - |
| | | Normal | 20.9% | - | - |

***weighted average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (49.9%) | Sleep | **28.9%** | Early | **28.9%** |
| | | | | Late | **0%** |
| | | Normal | 21.0% | - | - |
| Normal | (50.1%) | Sleep | **9.9%** | - | - |
| | | Normal | 40.2% | - | - |

***average***

| Ideal State | | Actual State | Percentage | Wakeup | Percentage |
|---|---|---|---|---|---|
| Sleep | (49.9%) | Sleep | **29.4%** | Early | **29.4%** |
| | | | | Late | **0%** |
| | | Normal | 20.5% | - | - |
| Normal | (50.1%) | Sleep | **17.7%** | - | - |
| | | Normal | 32.4% | - | - |

***Results for Adaptive Learning (Policy 2)***

*Adaptive Learning policy under **normal distribution (A)***

| Ideal State | | Actual State | Percentage |
|---|---|---|---|
| Sleep | (49.4%) | Sleep | **24.2%** |
| | | Normal | 25.2% |
| Normal | (50.6%) | Sleep | **25.5%** |
| | | Normal | 25.1% |

*Adaptive Learning policy under **2 normal distribution overlay (B)***

| Ideal State | | Actual State | Percentage |
|---|---|---|---|
| Sleep | (50.3%) | Sleep | **25.7%** |
| | | Normal | 24.6% |
| Normal | (49.7%) | Sleep | **24.5%** |
| | | Normal | 25.2% |

*Adaptive Learning policy under **fall distribution (C)***

| Ideal State | | Actual State | Percentage |
|---|---|---|---|
| Sleep | (50.0%) | Sleep | **24.0%** |
| | | Normal | 26.0% |
| Normal | (50.0%) | Sleep | **24.5%** |
| | | Normal | 25.5% |

*Adaptive Learning policy under **raise distribution (D)***

| Ideal State | | Actual State | Percentage |
|---|---|---|---|
| Sleep | (50.1%) | Sleep | **26.1 %** |
| | | Normal | 24.0% |
| Normal | (49.9%) | Sleep | **25.5%** |
| | | Normal | 24.4% |

*Adaptive Learning policy under **non-random (network burst)***

| Ideal State | | Actual State | Percentage |
|---|---|---|---|
| Sleep | (50.0%) | Sleep | **49.5%** |
| | | Normal | 0.5% |
| Normal | (50.0%) | Sleep | **0.5%** |
| | | Normal | 49.5% |