Identifying Bottlenecks & Optimizing Linux Performance in Embedded System Approach

By

Vipul Samar

08MCE015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

Identifying Bottlenecks & Optimizing Linux Performance in Embedded System Approach

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Vipul Samar

08MCE015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Computer Science & Engineering at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Vipul Samar

Certificate

This is to certify that the Major Project entitled **"Identifying Bottlenecks & Optimizing Linux Performance in Embedded System Approach"** submitted by **Vipul Samar** (08MCE015), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Shiraz Hashim
Project Manager,
ST Microelectronics,
Greater Noida ,
Delhi

Dr. S.N. Pradhan
P.G.Coordinator,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

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

Dr K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

Nowadays, embedded systems with Linux OS are commonly applied in automotive industry. Some of applications require strict time response, and others need to be lesser time to boot up and less power consumption such as mobile, PDA etc. But the major problem with the embedded system is to improve embedded linux performance without affecting the performance of the embedded system.

Dynamic power management (DPM) refers to the use of runtime strategies in order to achieve a tradeoff between the performance and power consumption of a system and its components.DPM has been a subject of intense research in the past decades driven by the need for low power in modern Embedded system. I present a formal method that have been explored in solving the system-level problem.

Bad Blocks are blocks that contain one or more invalid bits whose reliability is not guaranteed. Bad Blocks may be present when the device is shipped, or may develop during the lifetime of the device.A Bad Block does not affect the performance of valid blocks because it is isolated from the bit line and common source line by a select transistor. Bad Block Management, Block Replacement and the Error Correction Code software are necessary to manage the error bits in NAND Flash devices.

This dissertation addresses two key factors in embedded system design, namely minimization of power consumption and memory requirement. The first part of this dissertation considers the problem of optimizing power consumption. The second part deals with memory usage optimization mainly targeting the flash memory. Improving the uses of the flash memory with minimum time to access data.

Acknowledgements

My sincere thanks to the fine people around me who helped me in completing this project work. Their wisdom, clarity of thought and support motivated me to bring this project to its present state. First, I wish to thank **Mr. Amit Goel (Senior Section Manager)** for giving me an opportunity to work on this project. His continued support, guidance and vision have helped me in this project, and it has truly been a pleasure working with him.

My heartfelt thanks go to **Mr. Shiraz Hashim, Mr. Ajay Khandelwal** and my team members for their invaluable guidance, which not only enabled me to sort out the technical issues but also helped me in updating my knowledge, which undoubtedly will also be useful in the future.

I wish to place on record my gratitude to ST Microelectronics Pvt. Ltd, Greater Noida for providing me an opportunity to work with them on this project of such importance. My stay in the organization has been a great learning experience and a curtain raiser to an interesting and rewarding career. This exposure has enriched me with technical knowledge and has also introduced me to the attributes of a successful professional.

I wish to express my deep gratitude towards **Dr. S.N. Pradhan(PG Coordinator)** and all professors at Nirma Institute of technology who taught the fundamental essentials to undertake such a project. Without their valuable guidance it would have been extremely difficult to grasp and visualize the project theoretically.

Finally, I would like to thank my parents for their constant love and support and for providing me with the opportunity and the encouragement to pursue my goals.

- Vipul Samar

08MCE015

Contents

Declaration iii

Certificate iv

Abstract v

Acknowledgements vi

Contents vii

List of Tables x

List of Figures xi

Abbreviation xii

1 Introduction 1
- 1.1 Embedded linux . . . 1
- 1.2 Need of Embedded Linux . . . 1
- 1.3 Advantages of Embedded Linux . . . 2
- 1.4 Motivation . . . 4
- 1.5 Scope of work . . . 4
- 1.6 Thesis Organization . . . 5

2 Literature Survey 6
- 2.1 Embedded Systems . . . 6
- 2.2 Embedded systems design . . . 7
- 2.3 Challenges . . . 8
- 2.4 Plan Of Development . . . 11

3 Profiling Embedded Linux 12
- 3.1 Oprofile . . . 12
- 3.2 Linux Regression . . . 14
  - 3.2.1 Features of Linux Regression . . . 14
  - 3.2.2 List file format . . . 16

3.2.3 Platform Launching 18

4 Dynamic Power Management **21**
4.1 Power Management for embedded linux 21
4.2 Types of Power Management 23
4.2.1 Why **OSPM!**? 23
4.3 Dynamic Power Management 24
4.3.1 Application operating system interface 25
4.3.2 Operating System 25
4.3.3 Operating system-hardware interface. 25
4.3.4 States of Processor 26
4.3.5 C states: Almost all idle 26
4.3.6 P states: In operation 26
4.4 Suspend and Resume 27
4.4.1 Standby 27
4.4.2 Suspend to Ram 28
4.5 NAND Flash Memory 29

5 Implementation **31**
5.1 CPUfreq Scaling 31
5.1.1 Frequency power relation 32
5.1.2 Governors 33
5.2 Algorithm 35
5.2.1 Steps for modifying frequency 35
5.2.2 Prerequisites 37
5.3 Flashing Utility Design 39
5.3.1 Setting up memory partitions 40
5.3.2 Erasing Partitions 43
5.3.3 U-boot Through Script File 44
5.3.4 Selection of SOC 45
5.3.5 Programming Partitions 46
5.4 Bad Block Management 49
5.4.1 Bad Block Singed Table 50
5.5 Bad Block Manage Algorithm 51

6 Testing and Results **54**
6.1 Testing of Power Management 54
6.1.1 Test setup: 54
6.1.2 Using CPUfreq 55
6.2 Performance of NAND Flash Memory with Bad Bloack Management Algorithm 59

7 Conclusion and Future Scope **61**
7.1 Conclusion 61
7.2 Future Scope 62

A ST SPEAr^R Architecture **63**
A.1 SPEAr^R Technology 63
A.2 SPEAr^R architecture 64

Website References **67**

References **68**

Index **70**

List of Tables

5.1 Data Structure Of Bad Block Replacement Table 50
5.2 Structure Of Bad Block Signed Table 50

6.1 Power Management testing on SPEAr-300 56
6.2 Power Management testing on SPEAr-320 57
6.3 Power Management testing on SPEAr-600 58
6.4 Performance of NAND memory 60

List of Figures

2.1 Embedded System Architecture 8

3.1 Oprofile Block Diagram 14

4.1 a possible power state transition diagram for this PDA. 22
4.2 States of processor 27
4.3 Comparision of NAND and NOR memory 30

5.1 cpufreq Infrastructure 32
5.2 Setup memory partition 40
5.3 Erase memory partition 44
5.4 Program memory partition 47
5.5 Replacement Algorithm 52

6.1 Power Management Test Setup 55
6.2 Power Management chart for SPEAr-300 56
6.3 Power Management Chart for SPEAr-320 57
6.4 Power Management chart for SPEAr-600 58
6.5 NAND Performance chart 60

A.1 SPEAr Architecture 65

Abbreviation

SPEAr structured processor enhanced architecture

OS Operating system

ES Embedded system

DPM Dynamic Power Management

BBM Bad block management

SOC Systemon-chip

RTLinux Real Time Linux

RT Register Transfer

RTL Register Transfer Logic

ASICs Application Specific ICs

PM Power Management

LCD Liquid Crystal Display

Chapter 1

Introduction

1.1 Embedded linux

Embedded systems are dedicated computers that have fixed functions. Although they have been around for some time, they have explosive growth recently due to the ability of making smaller and faster semiconductor chips. Embedded systems can be divided into two categories. One has the OS, the other one is without the OS. When the functions of ES are getting more and more sophisticated, especially when networking is involved, the OS is needed to make the ES design easier. Although there are several embedded operating systems available, the embedded Linux has gained wide popularity.

1.2 Need of Embedded Linux

The power, reliability, flexibility, and scalability of Linux, combined with its support for a multitude of microprocessor architectures, hardware devices, graphics support, and communications protocols have established Linux as an increasingly popular software platform for a vast array of projects and products. Use of Linux spans the full spectrum of computing applications, tiny Linux wrist watch, to hand-held devices

(including PDAs and cell phones) and consumer entertainment systems, to Internet appliances, thin clients, firewalls, robotics, telephony infrastructure equipment, and even to cluster-based supercomputers[1].

Because Linux is openly and freely available in source form, many variations and configurations of Linux and its supporting software components have evolved to meet the diverse needs of the markets and applications to which Linux is being adapted. There are small-footprint versions and real-time enhanced versions. And despite the origins of Linux as a PC architecture operating system, there are now ports to numerous non-x86 CPUs, with and without memory management units, including PowerPC, ARM, MIPS, 68K, and even microcontrollers and there's more coming, all the time.

1.3 Advantages of Embedded Linux

a. **Vendor independence**

Multiple vendors offer embedded Linux product solutions with support, training and professional service options, using a variety of business models with a variety of methods of differentiation. Additionally, the fundamental source technology is available, making self development and self-support of a solution a vi-able (albeit expensive) alternative.

b. **Source code is available**

This gives the application developer a complete control over the functionality, down to the OS level, and allows the adaptation of modules already developed by others, instead of starting the development from scratch.

c. **Modularity and configurability**

At the kernel level, one can choose which modules to use depending on the hardware to support, and recompile and adapted kernel according to a well

documented and proved procedure. This allows to run Linux in systems with as little as 2 MB persistent storage and 4 MB RAM.

d. **Early availability of hardware support**

Linux is by far the preferred bring-up and initial support vehicle for hardware manufacturers. This hardware support starts at the microprocessor level, where Linux availability outstrips any other OS in the history of computing, and continues to the I/O level and beyond. Additionally, the vendor independence assures that if one Linux vendor is unable or unwilling to invest in the needed hardware support, there are alternative solutions.

e. **Early availability of new technologies in general**

Linux is also the preferred vehicle for computing system hardware and software innovation. Cutting-edge technology will almost always be available for Linux ahead of other system software platforms, and certainly relative to proprietary system software.

f. **Lower cost**

Linux starts by providing a royalty free deployment platform. Next, Linux deployment reduces costs to hire and train engineers, enhances the ability to customize, and offers a simpler and more convenient development environment. The result is a vastly reduced total cost of ownership[2][1].

1.4 Motivation

Embedded systems are normally embedded in a bigger system to perform a specific job. So they are expected to consume power as minimum as possible. Any electronics device as such is designed to consume less power as they are expected to work with battery power supply too. ES again has to run with those little power supply provided for electronic devices.Embedded systems should be optimized always for consuming less power. It will increase life of the system and will not hamper functioning of the parent device[3][4].

Boot time and memory utilization are two major problems in design of embedded system.Boot time is the time from initial power on to the time we can finally interact with the System.If system consume much time then it decreases the performance of the system.Memory utilization is in such a way that it can perform operation faster without waiting for resources.

The purpose of my dissertation is to optimize the power consumption,boot time and memory utilization in Embedded system.So that Embedded system can work properly without affecting the performance of high real time system.

1.5 Scope of work

As the title suggests the goal of this project, work carried out in this research is useful in improving performance and reliability of the Embedded system.

In this project, development of the DPM algorithm and Bad block management algorithm in embedded system. DPM is the mechanism that reduce power consumption and Bad Block Management is an algorithm that use to manage bad blocks in NAND flash memory in embedded system. With using these two algorithm we can reduce power consumption and optimize memory utilization in Embedded system.

1.6 Thesis Organization

The Thesis on Identifying Bottlenecks & Optimizing Linux Performance in Embedded System has been divided in chapters as follows:

Chapter 2,*Literature Survey*,presents the problem presents the literature review.It provides overview of Embedded system and ST SPEAr Architecture. It also explains challenges faced during the design of embedded system.

Chapter 3,*Profiling Embedded linux*,includes two methods Oprofile and Linux Regression to find out bottlenecks and limitation of Embedded linux. This explain the architecture and uses of both tools.

Chapter 4,*Dynamic Power Management*,includes power management mechanism in embedded system.The chapter includes mechanism use for reducing power consumption in embedded system.

Chapter 5,*Implementation*,it includes Algorithm and implementation of DPM and BBM Algorithm.

Chapter 6,*Simulation and Performance Evaluation*,covers simulation of DPM and Bad Block Management Algorithm. The chapter includes the output of the simulation and generates the graph.

Chapter 7,*Conclusion and Future work*,concludes this project with a summary, and provides possible directions for relevant future research.

Chapter 2

Literature Survey

Linux is making steady progress in the embedded arena. Because Linux is covered under the GPL, anyone interested in customizing Linux to his PDA, palmtop, or wearable device can download the kernel and applications freely from the Internet and begin porting or developing. Many Linux flavors cater to the embedded/realtime market. These include RTLinux, uclinux (Linux for MMUless devices), Montavista Linux (Linux distributions for ARM, MIPS, PPC), ARM-Linux (Linux on ARM), and others.Embedded Linux development broadly involves three tiers: the bootloader, the Linux kernel, and the graphical user interface.

2.1 Embedded Systems

Embedded systems nowadays are present in most of the electronic devices and instruments used in daily life; they can be found in consumer electronic devices (calculator, digital cameras, cell phones, etc.), office equipment (printers, copy machines, fax machines, etc.), home appliances (microwave ovens, washing machines, alarms, etc.), and automobiles (cruise control, transmission control, fuel injection, etc.).

Loosely speaking, an embedded system is any computing system other than a desktop computer [VG02]. An embedded system typically consists of four main com-

ponents: an embedded processor, synthesized circuit for dedicated hardware units, memory, and I/O interface. All of these are typically implemented in one chip constituting what is called a systemon-chip SOC.

Embedded systems exhibit certain characteristics that distinguish them from other computing systems. These characteristics are:

a. Single function: An embedded system usually executes a certain task (or program) repeatedly.

b. Real-time operation: Time constraint is very crucial in execution of tasks on embedded systems. Even a small execution delay might cause a serious malfunctioning or total failure.

c. Tight constraints:Because of the nature of embedded systems, their design metrics such as size, performance and power impose tight constraints.

At the system level, the design is described as a set of interacting subsystems (processes) to be mapped to either hardware or software components. These subsystems can be implemented using processors (software), ASICs, memories and dedicated hardware. At the behavioral level, each subsystem is specified in its algorithmic (or functional) form. At the RT level, the system is specified as a collection of communicating RTL units such as ALUs, registers, and multiplexers. The logic level specification is the hardware implementation of the logic functions given as a netlist of logic gates and flipflops. The physical level description is the physical implementation given as a netlist of transistors, capacitors, and resistors on a board.RT, logic, and physical levels belong to the hardware side. Module level and block level specifications are the typical levels of abstraction on the software side.

2.2 Embedded systems design

Embedded systems consist of hardware, software,and an environment. This they have in common with most computing systems. However, there is an essential difference

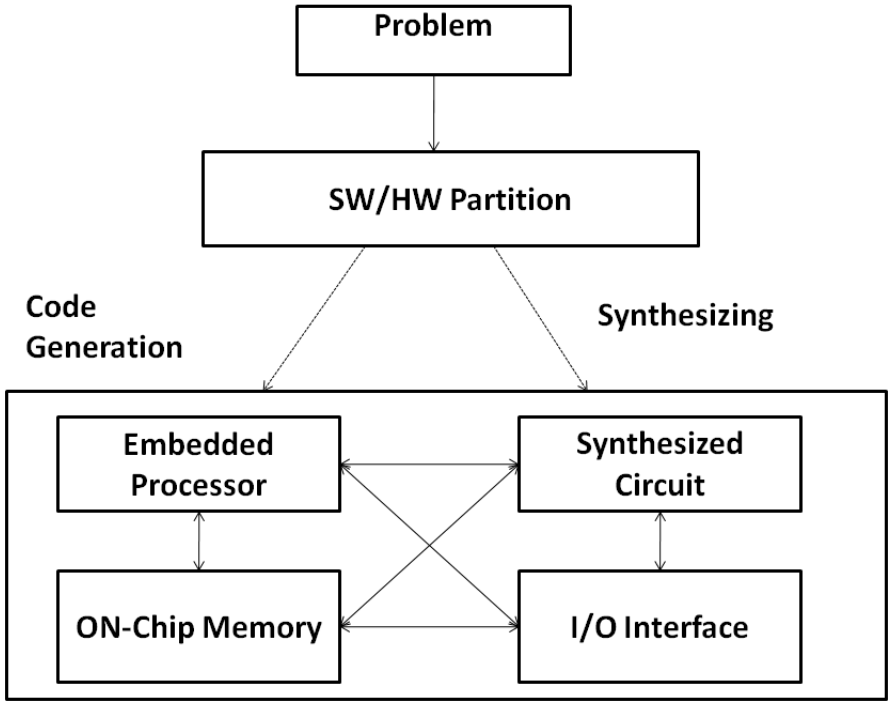

Figure 2.1: Embedded System Architecture

between embedded and other computing systems: since embedded systems involve computation that is subject to physical constraints, the powerful separation of computation (software) from physicalist (platform and environment), which has been one of the central ideas enabling the science of computing, does not work for embedded systems. Instead, the design of embedded systems requires a holistic approach that integrates essential paradigms from hardware design, software design, and control theory in a consistent manner[1].

2.3 Challenges

a. **Hard Real-Time** Embedded systems are mostly real-time systems, meaning that they have to respond to external events in a timely fashion. In many cases

(eg. multi-media systems) this real-time requirement is soft, meaning that such systems can tolerate missing a deadline occasionally.

Normal Linux is not suitable for hard real-time systems, and there are signs that the situation has recently worsened. Special real-time versions, such as RTLinux [FSM] and RTAI [RTA] address the problem by adding a real-time layer below the kernel proper, in order to have full control over interrupt handling. This leads to an architecture which is, in principle, capable of meeting real-time requirements, although at a cost of running the real-time components in the kernel

b. **Highly Robust** Embedded systems are often employed in life-critical or mission-critical scenarios. While the reliability of Linux on desktop and servers is very high, this typically applies to systems which are at least close to widely-deployed configurations. Massive changes to system configuration, as it is typically necessary for an embedded system, will inherently reduce stability and require a significant maturation process.

   A related issue is that of upgrading the system without downtime. While it is possible, in theory, to upgrade Linux kernel modules without rebooting the whole system, in practice this is very limited, as many modules are tied closely to a specific kernel version, making it impossible to load a newer version of the model into an old kernel. Other components of the kernel are impossible to upgrade without a reboot

c. **Power Consumption** Many technological improvements show an exponential increase: circuit density, storage capacity, processing power, network bandwidth. An exception is the energy density in batteries, which is improved, but much less dramatically. The power usage of processing per Watt has improved significantly, but unfortunately the processing needs have increased also rapidly. Overall the power consumption of for instance PCs is more or less constant[2].

There are multiple reasons to strive for less power consumption:

(1) Less heat dissipation, easier transport of waste heat.

(2) Increase stand by time.

(3) Increase operational time.

(4) Decrease acoustic noise of cooling.

(5) decrease power supply cost.

2.4 Plan Of Development

Ideas for optimizing the Linux kernel and kernel Boot Time in embedded System:

a. Increasing speed Configurations of Linux for desktop and server markets exhibit boot times in the range of 20 seconds to a few minutes, which is unacceptable for many embedded system. Increasing speed means reducing the kernel boot time. To produce real-time output within given hard deadline the kernel boot time play an important role. There are many techniques used to avoid delay in kernel boot process such as: Skip memory allocation,Avoiding Probing During Bootup etc.

b. Reducing size Disk footprint and RAM In order to boot Linux, require a boot loader, which is a small program that runs before the main kernel. The boot loader is expected to initialize various devices, and eventually call the Linux kernel, passing information to the kernel. Boot loaders first task is to initialize the RAM. Different methods that used to reduce RAM size are as follow: Remove kernel messages, Use simpler algorithms with fewer features, reduce the size of some kernel resources, Use a better allocator for small systems etc.

c. Reducing power consumption Power management is complex to implement in embedded Linux systems. In particular, device drivers must support it. Executing the kernel and applications directly from flash (no copy to RAM) allows to have smaller / fewer RAM chips, consuming less power.

Chapter 3

Profiling Embedded Linux

3.1 Oprofile

OProfile is very useful for identifying processor performance bottlenecks. OProfile can be configured to take samples periodically to get time-based samples to indicate which sections of code are executed on the computer system. On many architectures OProfile provides access to the performance monitoring counters. The performance monitoring counters allow samples to be collection based on other events such as cache misses, memory references, instructions retired, and processor clock cycles. These performance events allow developers to determine whether specific performance problems exist in the code and revise the code appropriately based on the performance problem observed[5].

OProfile is a low-overhead system-wider profiler for Linux. The Linux 2.6 kernel supports OProfile for a number of different processor architectures. The major OProfile components are shown in Figure. The kernel has a driver which controls the performance monitoring hardware and collects the samples. The driver interfaces to the user space with the oprofile pseudo file system. The daemon read data from the oprofile pseudo file system and converts the data into a sample database.

The opcontrol script manages OProfile's profiling sample system. The analysis programs such as opreport and opannotate read data from the sample database. OProfile can be divided into three sections: the kernel support (left part), the daemon (center), and the sample database with analysis programs (right part). To collect measurements the opcontrol script write the events to measure in the oprofile pseudo file system, once the kernel driver has been initialized, opcontrol spawns the oprofi led daemon which reads the sample data from the OProfile pseudo file system buffer. The daemon processes the data and places the converted data into /var/lib/oprofile/samples. Status information is placed into /var/lib/oprofile /oprofile.log by the daemon. The opcontrol script can also force data to be flushed to the sample database; the script determines when the daemon has finished the operation by referencing the file /var/lib/oprofile/complete_dump.

The sample database is stored in /var/lib/oprofile/samples. The opreport and opannotate programs extract the profile information from the sample database and display it in human-readable form.

Install and Configure OProfile

a. install OProfile

   (1) ./configure -with-linux=/linux/path

   (2) make install

b. Configure OProfile

   (1) To monitor the kernel execute following command
   Opcontrol -vmlinux=/vmlinux/path

   (2) Start oprofile
   Opcontrol start

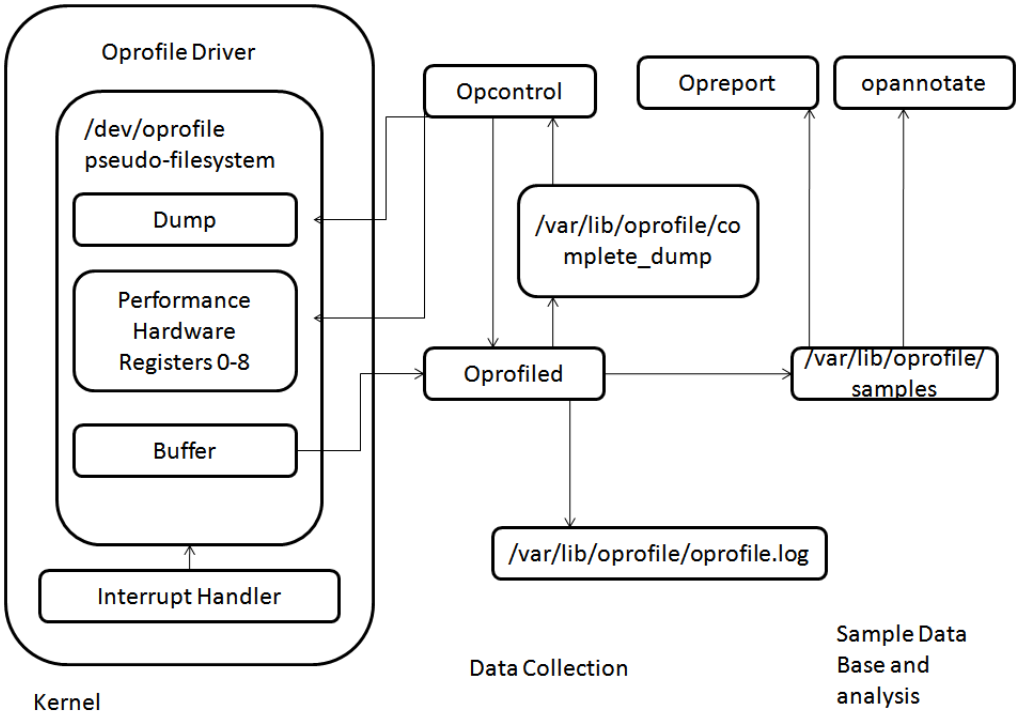

Figure 3.1: Oprofile Block Diagram

(3) Stop Oprofile

Opcontrol stop

(4) Generate the annotate report of the kernel

Opannotate -source=/path/of/linux

(5) It generate the report

Opreport

3.2 Linux Regression

3.2.1 Features of Linux Regression

Following are the Features supported by the Linux Regression Platform[6].

a. **Target Test Cases:** These test cases are launched on the target and the result of the test cases is reported to Host PC for result compilation. These test cases are marked by "B" in test case list files.

b. **Host PC Test Cases:** These test cases are launched on the host environment and are marked by "P" in test case list files.

c. **Interactive Test Cases:** These test cases require some human intervention for execution. They are marked by "I" in test case list fies. When these test cases are executed a shell is opened for the user to type the necessary commands. Shell is closed by typing "pass" or "fail". This reports the status of the interactive test case.

d. **Reboot command:** This feature enables the kernel image to be updated/changed at run time. After the kernel image is updated in the ash, the kernel is rebooted. This is particularly useful for IP's requiring different kernel configurations for different test cases. This command is also used in case of Kernel Panic and Soft Lockups.

e. **Result Compilation:** Platform generates result file containing details of test cases, test case result, known causes of error etc. Result files (TestCaseSummary.csv and CommandSummary.csv) can be viewed in MS-Excel.

f. Platform supports exclusion of test directories. IP's can be excluded from the regression test site if needed.

g. Linux Regression platform is currently based on ethernet as the medium of communication between the Host PC and the target board. TCF supports more than one type of link for communication like serial, usb etc and therefore these can be used in future without changing the platform design.

The test cases for each IP are defined in a text file called the list file. Each IP has its own directory and each IP directory contains four list files for the target platforms

S300 AB, S300 BA, S600 AB, S600 BA respectively The name of the list file should be <SOC NAME> <IP Name>.list
eg: S300 AB i2c.list

The Regression platform picks the list files for all IP's that are to be run in the regression suite for a particular target platform. The test cases are executed in the order as mentioned in the list file. Interactive test cases are an exception to this rule, and they are run in the end as they require human intervention. So if a testcase contains any interactive command it will be executed at the end of regression.

3.2.2 List file format

Each list file contains test cases for the particular IP and each test case consists of multiple test commands. The last command of each test case generates the result of that particular test case.
Test Case syntax:

a. **Test Case LOOP**

   (a) Each test case begins with the keyword "LOOP", followed by a integer number, indicating the number of times, the test case is to be run.

   (b) eg: LOOP 5

b. **Test Case START**

   (a) LOOP keyword is followed by the keyword "START" and the name of the test case. The Name should not contain any white space character and should be max 100 chars. Name should not be put in inverted commas and each test case should have a unique name.

   (b) eg: START ETHERNET 01

c. **Test Case DESCRIPTION**

(a) Each test case body contains the test case description which is captured in the TestCaseSummary.csv test report generated. This gives a brief summary of the test case. Description of the test case inside quotes should not be more than 400 chars and should not contain any inverted commas.

(b) eg: DESCRIPTION "Ethernet Test Case at 100/F Con_guration"

d. **Test Case BODY**

(a) Each test case consists of multiple test commands. These commands can be of four types. Board type command "B", Interactive Command "I", HostPC command "P" and TCF commands "CMD". The B and P type commands are launched on the target board and Linux host PC respectively. The "CMD" denotes the TCF command issued by the client to the agent.

(b) The name of the command should not be more than 100 chars long and should not contain spaces and inverted commas. All the test commands under one test case should have unique names.

(c) All the above commands can be run in a loop, denoted by the Loop No in the syntax below.

(d) All command arguments must be space separated. No inverted commas are allowed in the command line, except at the end to describe the command. The command does not support wildcards like in shells.

(e) The Description at the end of the test command gives a brief summary about the test command. It is captured in the CommandSummary.csv test report generated.

(f) As TCF allows new services to be added, reboot type of command has been added under the process service.

(g) eg: B/P/I <Name> <Loop No> <command name> <arg1> <arg2> <argn> "Description".

(h) eg: P ETH 01 1 ./ethtool -s eth0 autoneg on "This command runs once on PC".

(i) eg: B ETH 02 5 tftp -g -r testfile -l testfile 192.168.1.1 "This command runs five times on board"

e. **Test Case END**

(a) Each test case ends with the keyword END.

(b) eg: Below is an example of a list file containing 2 test cases.

LOOP 1
START TESTCASE ONE
DESCRIPTION "Test Case 1 of test list file"
B test1 1 test board fail.run "This command will run on board once"
P test2 1 test pc segmentationfault.run "This command will run on Host PC once"
B test3 1 test board fail.run "This last test command generates the result of this test casel"
END

3.2.3 Platform Launching

a. Copy the folder 'LinuxRegression' to the filesystem exported via NFS on the Linux Host PC.

b. Ensure that the DHCP and NFS server deamons are ON on the Linux Host PC.

c. Edit the rc file of the filesystem to automatically create dev nodes and to launch the agent on the target after kernel boot. Add the following lines at the end of

the rc file. (Copy the node.sh script from the 'LinuxRegression' folder to the 'etc' folder of the filesystem.)

(a) ./etc/node.sh

(b) cd LinuxRegression

(c) ./agent -L-

d. Boot the Linux kernel on the board. The agent will be automatically launched.

e. On the Linux Host PC, run the following commands

(a) cd LinuxRegression

(b) ./client -a192.168.1.10 -bS300_AB -L <log file name>

(c) -a<IP Address> is the IP of the target board

(d) -b<S300 AB> is the target platform type. It can be one of the following S300 AB, S300 BA, S600 AB or S600 BA

(e) -L<log file name> log file is the TCF log file, if "-" is the log file name, log will be dumped on std output

f. Once the client is launched, the Regression platform begins to execute the test cases. At the end of the Regression, the client exits itself. The test report files TestSummary.csv and CommandSummary.csv are generated in the top directory and can be viewed in MS-Excel.

- **TestCaseSummary.csv:**This file generate the final test case summary. Sr.No. , IP, Name, TestCase Status, Description

(a) IP Name is the name of the IP to which test case belongs.

(b) Test Case is the name of test case as put in list file in START field.

(c) Description is same string as put in DESCRIPTION field of test case body.

(d) Status is the result of last command executed in the test case. Status is PASS or FAIL.

- **CommandSummary.csv :**This file generate the individual command summary of each test case. This file is for debugging purpose only Sr.No., IP, TestCase Command Name, Status Description, Return Value

  (a) TestCase - this is the test case name.

  (b) Command Name - this is the individual command under each test case body.

  (c) Status - this is the status of each command. Status can be of the type Pass /fail.

Chapter 4

Dynamic Power Management

4.1 Power Management for embedded linux

PM software is a crucial component in battery-powered systems, such as PDAs and laptops, because it helps conserve power when the system is inactive. As a simple example, power may be conserved by switching off the display when a system is inactive for some time. Conserving power in this manner extends battery life, so one can work more hours before having to recharge the battery.

Hardware support is vital for power management to work, and software intelligently exercises that support. The degree of power management support available in hardware varies from device to device. Some devices, such as a display, simply provide two power states, on and off. Other devices, like the SA1110 CPU, may support more complex power-saving features, including frequency scaling.

Implementing power management in any system is a complex task, considering that several non-interacting subsystems need to be brought together under a single set of guidelines.

Power Management Implementation Before implementing power management, it is important to understand what hardware support is available for saving power. One of the important goals of power management software is to keep all devices in

their low power states as much as possible[7].

Here is a brief description of the power states.

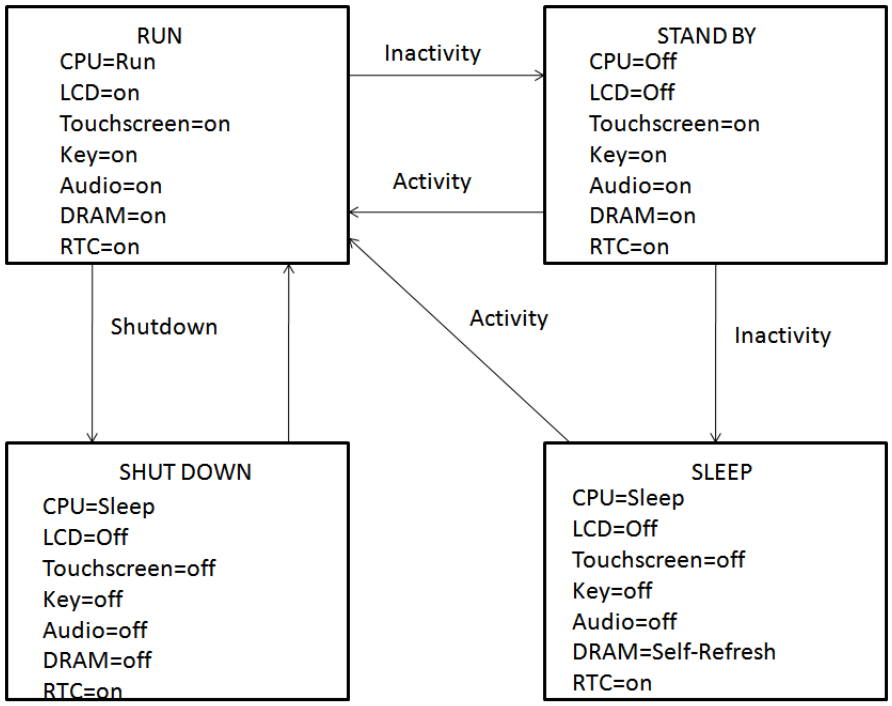

Figure 4.1: a possible power state transition diagram for this PDA.

- **Run state:** system falls into this default state when it reboots. Power consumption is maximum in this state, as all devices are turned on or active.
- **Standby state:** system falls into this state due to inactivity. LCD and display back light are turned off, and CPU clock speed is reduced to save some power.
- **Sleep state:** system falls into this state due to continued inactivity. Power is conserved aggressively by putting the CPU in sleep mode, which in turn powers off most devices. DRAM, however, is put in its self-refresh mode to preserve the machine state (system and application text/data loaded in memory) while the system is sleeping. The system awakens from sleep state when a preprogrammed

event occurs. When it wakes up, it transitions to the run state and machine state is restored.

- **Shutdown state:** system falls into this state when the shutdown command is issued. The system reboots when it exits from this state. This means it is not necessary to preserve the machine state in DRAM, and hence DRAM can be powered off. The shutdown state then represents the lowest power consumption state of all.

The real-time clock is kept on in all power states to retain system time.

4.2 Types of Power Management

Power Management can be of two types:[8]

a. Incorporated in the device itself. Device driver intervention needed.

b. Incorporated in Operating System. This is called Operating System-directed Power Management(OSPM)

Device directed power management is slowly and gradually becoming obsolete in the system.

4.2.1 Why OSPM!?

OSPM! provides some fundamental edge over driver directed power management. Here are few advantages over driver directed power management:

Advantages

a. Moving power management functionality in the OS itself will help all the platforms and devices to utilize it. Power management capabilities are available to all the systems on which the OS is installed. This can be exploited to the maximum by system and firmware designers.

b. This will reduce the industry wide practice of investing in power management algorithms in the device driver/device itself. As such common algorithms will be incorporated in the OS, organizations can think of innovations rather than simple parity.

c. Limitation of implementing of complex power management algorithms in the BIOS has been overcame.

d. Development of **OSPM!** can be done independent to development of OS. This abstraction is a key eye-catcher which has interested many major players

4.3 Dynamic Power Management

Techniques oriented to enable and disable components, as well as adapting their performance to the workload, are called DPM techniques[9].

These ones can make use of several features supported by modern hardware designs, including multiple power states in I/O devices and variable-voltage processors, and can be implemented at different abstraction levels.

Recently, the targeting of DPM strategies towards the operating system (OS) level has gained importance due to its flexibility and ease of use. In fact, because of the OS has an overall view of the system resources and workload trend, it is possible to take customized power management decisions and, as a consequence, achieving significant energy savings.

DPM techniques reported in literature can be classified in three main levels:

a. Application operating system interface

b. Operating system

c. Operating system-hardware interface.

4.3.1 Application operating system interface

At the application-operating system level, it is possible to apply power optimization by integrating the application layer into dynamic power management of devices. These techniques exploit I/O devices to save energy. For example, in a new OS interface is introduced for cooperative I/O that can be exploited by energy-aware applications, while in this is achieved by introducing new system calls which allow interactive applications to inform the OS about future device requests: this enable a proper schedule of the processes and, consequently, a power reduction[10].

4.3.2 Operating System

At OS level it is possible to search for tasks requiring services from hardware components, as sources of power consumptions. OS has detailed knowledge about running tasks, so that this information can be used for power management. A DPM scheme at OS level that adapts power state of hardware components depending on workload, can deal with problems that in other levels cannot be handled and can identify time intervals where I/O devices are not being used and switch these devices to low power state.

4.3.3 Operating system-hardware interface.

A significant effort is spent toward techniques that act at the operating system-hardware level, namely, that try to apply, at the same time, OS techniques based on particular hardware architectures. In this wide class we can distinguish between the specialized hardware that can be typically a microprocessor or a memory. In literature there are several techniques, namely scheduling algorithms, for variable voltage selection microprocessors or page replacement algorithms for Rambus off-chip memory. sectionMethods for power management Three methods that are proposed for power saving in embedded linux.

a. Suspend and resume.

b. CPUfreq scaling.

c. CPUidle.

Before exploring these methods it is good to know about the states of the processor.

4.3.4 States of Processor

4.3.5 C states: Almost all idle

C states are idle states in which the processor will unclock and shut down components to save power. Steps taken by c stats:

a. Stopping the processor clock.

b. Stopping interrupts.

c. Lowering the voltage and frequency.

These states can provide power savings when the system is idle. Remember, the deeper the C state, the more power savings.

4.3.6 P states: In operation

P states are operational states that relate to CPU frequency and voltage. The higher the P state, the lower the frequency and voltage at which the processor runs. The CPUfreq governors use P states to change frequencies and lower power consumption.

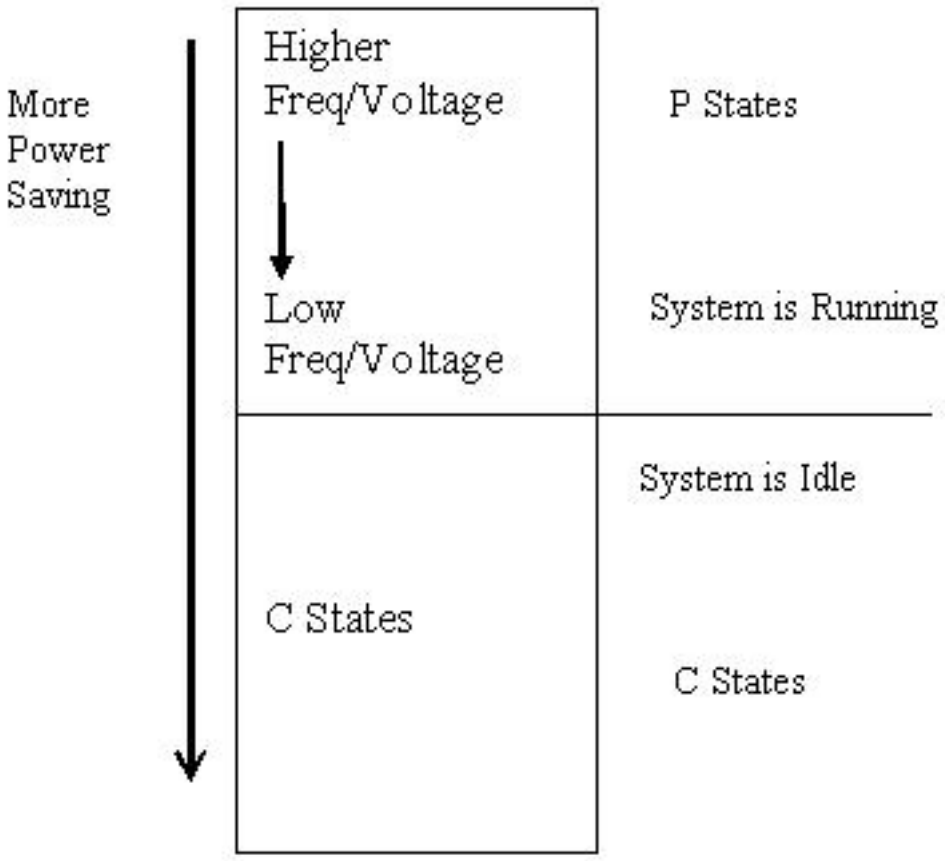

Figure 4.2: States of processor

4.4 Suspend and Resume

It has two methods implemented - Standby and Suspend to Ram.

4.4.1 Standby

This state offers minimal, though real, power savings, while providing a very low-latency transition back to a working system. No operating state is lost (the CPU retains power), so the system easily starts up again where it left off. Here, devices are

put in a low-power state, which also offers low power savings, but low resume latency. Not all devices support low-power state, and those that don't are left on[8].

4.4.2 Suspend to Ram

This state offers significant power savings as everything in the system is put into a low-power state, except for memory, which is placed in self-refresh mode to retain its contents. System and device state is saved and kept in memory. All devices are suspended. In many cases, all peripheral buses lose power when entering STR, so devices must be able to handle the transition back to the ON state.

4.5 NAND Flash Memory

Flash memory has become a powerful and cost-effective solid-state storage technology widely used in mobile electronics devices and other consumer applications. Two major forms of Flash memory, NAND Flash and NOR Flash, have emerged as the dominant varieties of non-volatile semiconductor memories utilized in portable electronics devices[11][12].

NAND Flash, which was designed with a very small cell size to enable a low cost-per-bit of stored data, has been used primarily as a high-density data storage medium for consumer devices such as digital still cameras and USB solid-state disk drives. NOR Flash has typically been used for code storage and direct execution in portable electronics devices, such as cellular phones and PDAs.

NAND Flash was developed as an alternative optimized for high-density data storage, giving up random access capability in a tradeoff to achieve a smaller cell size, which translates to a smaller chip size and lower cost-per-bit. This was achieved by creating an array of eight memory transistors connected in a series. Utilizing the NAND Flash architecture's high storage density and smaller cell size, NAND Flash systems enable faster write and erase by programming blocks of data. NAND Flash is ideal for low-cost, high-density, high-speed program/erase applications, often referred to as data-storage applications.

The NAND FLASH is composed by several storage unit blocks, and every storage unit block which is the least erasing unit includes several storage pages. The storage page is the least unit to write data Compared with normal memory the reading speed of FLASH is faster, but the data must be erased with storage unit block before writing accord page order. There are some initial invalid blocks (bad block) in NAND FLASH that the manufacturing company do not ensue its reliability. The bad blocks are allowed existing in end product because using NAND technology. The bad blocks do not affect the performance of the other blocks, but they should be masked by

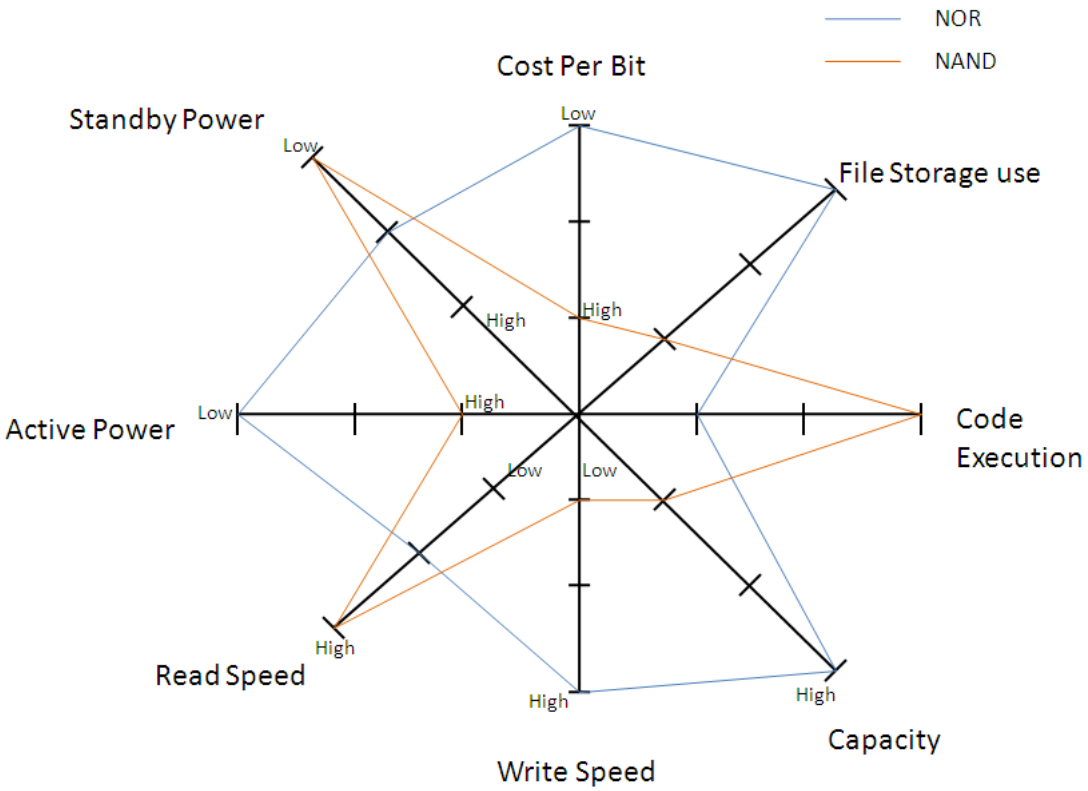

Figure 4.3: Comparision of NAND and NOR memory

address mapping table in system.

When chips produced, all parts except the area that save information of bad blocks are erased (the value is 0xFF). The bad blocks must be distinguished according the initial bad block information when designing system, and build bad blocks table. It needs to compare the block address which going to be operated with bad block address table, jumps if it is bad block. New bad blocks maybe produced when chips running, so these conditions should be considered in order to keep system reliability. If defaults were found in reading state register after writing data or block erasing operation, that means there are bad pages in block i.e. this is a bad block and its information should be saved into the bad block table.

Chapter 5

Implementation

5.1 CPUfreq Scaling

CPUfreq, which changes the processor frequency and/or voltage and manages the processor performance levels and power consumption based on processor load. We can dynamically scale processor frequencies through the CPUfreq subsystem. When processors operate at a lower clock speed, they consume proportionately less power and generate less heat. This dynamic scaling of the clock speed gives some control in throttling the system to consume less power when not operating at full capacity[8].

The CPUfreq structure makes use of governors and daemons for setting a static or dynamic power policy for the system. The dynamic governors can switch between CPU frequencies based on CPU utilization to allow for power savings while not sacrificing performance. These governors also allow for some user tuning so you can customize and easily change the frequency scaling.

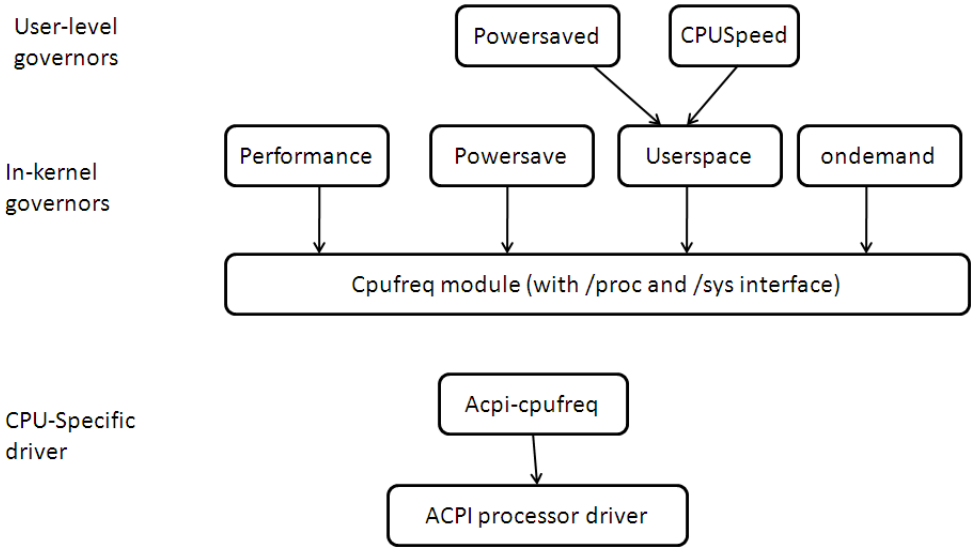

Figure 5.1: cpufreq Infrastructure

5.1.1 Frequency power relation

Current CMOS electronics consumes power in three big areas:

P = P1+P2+P3

leakage - current going either through substrate (under schematics) or not fully closed transistors (through schematics) and depends on voltage, thus this part of the power will be proportional to the square of the voltage:

P1 = IL *UC =U2C/RL

recharging parasitic capacitance of wires and inputs-depends on both frequency and voltage, linear on frequency and square on voltage.

P2 =U2C/RP =U2C*CP*F

shoot-through current - happens during the switch of the CMOS circuit then one transistor is already open while opposite to it is just started to close, and thus, is linear proportional to frequency and square proportional to voltage.

P3 =U2C*F/RS

Summarizing the above, consumed power is proportional to square of core voltage and either constant or linear to frequency, depending on which power consumer on a

chip dominates. Maximum frequency of the CMOS circuit depends on core voltage as well, and thus, to save power we need to decrease the core voltage, but beforehand set frequency to value, allowed at this reduced voltage[12][13].

5.1.2 Governors

There are five in-kernel governors available for use with the CPUfreq subsystem. These governors set the processor frequency based on certain criteria. Some dynamically change the frequency as inputs are changed either by the system or the user[13].

a. Performance governor: Highest frequency The performance governor statically sets the processor to the highest frequency available. We can adjust the range of frequencies available to this governor. As the name implies, this governor's goal is to get the maximum performance out of a system by setting the processor clock speed to the maximum level and leaving it there. This governor does not attempt to provide any power savings by default, although we can tune the governor to change the frequency it selects.

b. Powersave governor: Lowest frequency On the flip side, the powersave governor statically sets the processor to the lowest available frequency. Again we can adjust the range of frequencies available to this governor. The purpose of this governor is to run at the lowest speed possible at all times. Obviously this can affect performance in that the system will never rise above this frequency no matter how busy the processors are.

   In fact, this governor often does not save any power since the greatest power savings usually come from the savings at idle through entering C states. Using the powersave governor will prolong a running process since it will be running

at the slowest frequency; therefore, it will take longer for the system to go idle and get the C state savings.

c. Userspace governor: Manual frequencies Next there is the userspace governor, which allows us to select and set a frequency manually. This governor also works with processor frequency daemons running in userspace to control frequency. This governor is useful for setting a unique power policy that is not preset or available from the other governors.

Note that the userspace governor itself does not dynamically change the frequency; rather, it allows us or a userspace program to dynamically select the processor frequency.

d. Ondemand governor: Frequency change based on processor use

The ondemand governor was the first in-kernel governor to dynamically change processor frequency based on processor utilization. The ondemand governor checks the processor utilization and if it exceeds the threshold, the governor will set the frequency to the highest available. If the governor finds the utilization to be less than the threshold, it steps down the frequency to the next available. If the system continues to be underutilized, the governor will continue stepping down the frequency until the lowest available is set.

We can control the range of frequencies available, the rate at which the governor checks utilization on the system, and the utilization threshold.

e. Conservative governor: A more gradual ondemand Based on the ondemand governor, the conservative governor is similar in that it dynamically adjusts frequencies based on processor utilization; however, the conservative governor

behaves a little differently and allows for a more gradual increase in power. The conservative governor checks the processor utilization and if it is above or below the utilization thresholds, the governor steps up or down the frequency to the next available instead of just jumping to the highest frequency as ondemand does[10][9].

We can control the range of frequencies available, the rate at which the governor checks utilization on the system, the utilization thresholds, and the frequency step rate.

5.2 Algorithm

Based on workload modify frequency Based on the workload of the CPU frequency can be modified. First calculate the amount of time CPU is idle. And based on the idle time calculate the load of the CPU[13].

```
If(load > up_threshold)
If(policy-> cur=policy->max)
Return;
else
assign policy->max as a new freq;   //increase the value to the max.
Else if If(load < up_threshold)
Freq=(policy->max-load)/up_threshold       //assign it as a new freq.
```

5.2.1 Steps for modifying frequency

a. Step1 Define the threshold values for load:
   MIN_UP_THREASHOLD
   MAX_UP_THREASHOLD

DEF_UP_THREASHOLD

b. Step2 Define values for sampling rates
MIN_SAMPLIN_RATE
MAX_SAMPLING_RATE
TRANSITION_LATENCY_LIMIT

c. Step3 Calculate CPU idle time
For calculating CPU idle time get the total CPU time by the function get_cpu_jiffies().
CPU idletime is the (total_cputime - cpu_busytime).
For calculating the cpu_busytime add all the time spend in user space, System space, interrupt, soft interrupt and cpu stealing cycle.
So final
idletime = total_cputime - cpu_busytime.

d. Step4 Set the different frequency for jiffies with power save option.

e. Step5 Calculate load

f. Step6Compare load and adjust frequency according to the workload.

```
if (load > dbs_tuners_ins.up_threshold) {
    /* if we are already at full speed then break out early */
if (policy->cur == policy->max)
                return;
if (policy->cur == policy->max)
```

```
                                return;
        } else {
              int freq = powersave_bias_target(policy, policy->max,
                                CPUFREQ_RELATION_H);
                 }
                     return;
               }

     if (load < (dbs_tuners_ins.up_threshold - 10)) {

unsigned int freq_next=(policy->cur*load)/(dbs_tuners_ins.up_threshold-10);
               __cpufreq_driver_target(policy, freq_next, CPUFREQ_RELATION_L);
               if (!dbs_tuners_ins.powersave_bias) {
                    __cpufreq_driver_target(policy, freq_next,
                                        CPUFREQ_RELATION_L);
               } else {
                int freq = powersave_bias_target(policy, freq_next,
                                        CPUFREQ_RELATION_L);

                     }
               }
```

5.2.2 Prerequisites

First check that CPUfreq is enabled or not. One quick way to check if CPUfreq is already enabled is to look in the /sys filesystem. If you see the cpufreq directory listed under /sys/devices/system/cpu/cpu*/cpufreq/, then your system currently has CPUfreq enabled.

If you do not see this directory listed, follow these instructions to ensure that you have the required pieces.
First, make sure that processor can support frequency scaling. Next, look at kernel config file. The following options are located in the CPU Frequency scaling section of the config file:
CONFIG_CPU_FREQ
This option must be set to y to make use of the kernel's CPU frequency scaling.
CONFIG_CPU_FREQ_GOV_PERFORMANCE
CONFIG_CPU_FREQ_GOV_POWERSAVE,
CONFIG_CPU_FREQ_GOV_USERSPACE
CONFIG_CPU_FREQ_GOV_ONDEMAND,
CONFIG_CPU_FREQ_GOV_CONSERVATIVE
These options are for each of the available CPUfreq governors. To use a governor, set the config option to y or m. If you set the option to y, that governor's module will be built into the kernel. If you set the option to m, you will have to load the module yourself for each boot by issuing one or all of the following commands:
Modprobe cpufreq_performance
modprobe cpufreq_powersave
modprobe cpufreq_userspace
modprobe cpufreq_ondemand
modprobe cpufreq_conservative

Alternatively, if you chose m, you can have the module loaded at boot time by adding the governor modules to /etc/rc.local. Also note that you can set either the userspace or the performance governor to the default by setting either
CONFIG_CPU_FREQ_DEFAULT_GOV_USERSPACE or
CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE to y.

5.3 Flashing Utility Design

This section explains the directory structure of Utility in more details. The main folder Flashing Utils contents are as below[14][15].

a. Binaries: This folder contains all the binary images required to flash the board. It also contains the DDR2 driver and firmware image which are downloaded to the board automatically when we click on the connect button on Flashing Utility window.

b. Configurations: This folder is the default location for keeping the stored configurations for later use. These configurations define the programming window options eg. which image has to be programmed to which partition etc.

c. Scripts: This is the default location for U-Boot command scripts which can be run on the board. It contains a default test scripts. The required scripts may be generated on similar pattern.

d. Setup: This folder contains the files required to perform the initial setup on Windows Host

e. Tools: This directory contains 2 executables which are used while normal running of Flashing Utils. USB send.exe is used to send the DDR2 driver and firmware when connect button is clicked and ukermit.exe is used to send image files over a COM port via Kermit protocol.

   (a) src: This directory contains the source code for the 2 executables contained in the Tools directory

f. SPEAr-Utils.tcl: This is the actual utility written in TCL language which opens the GUI to interact with the user.

5.3.1 Setting up memory partitions

Click on the Setup menu and Setup Memories Partitions

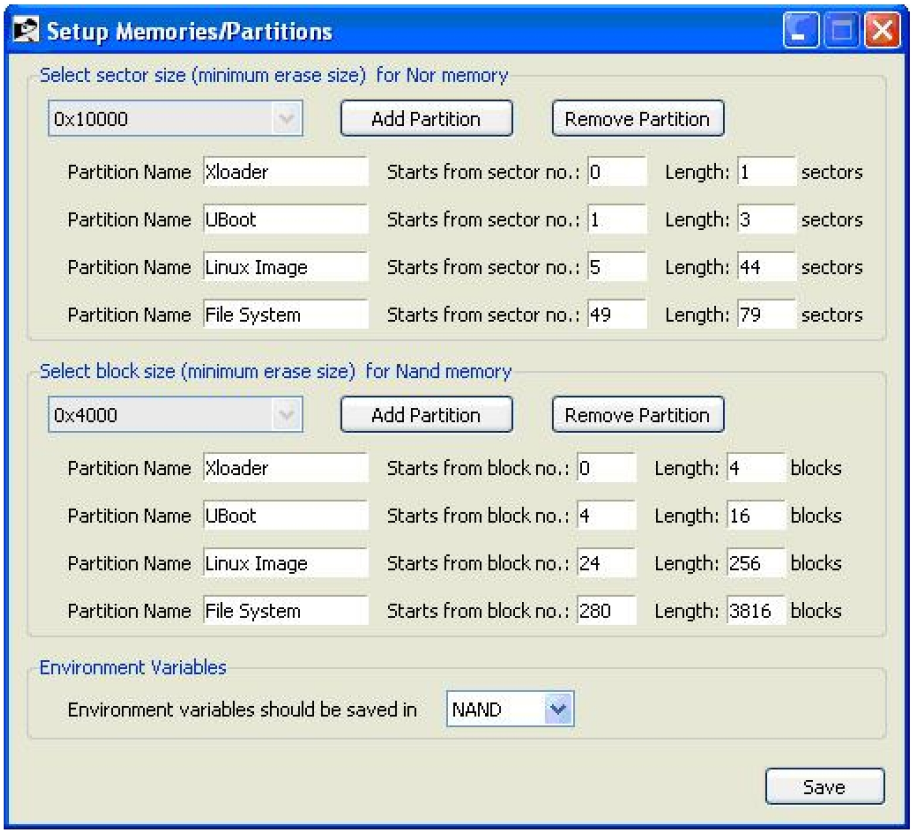

Figure 5.2: Setup memory partition

The partitions can now be added or removed. 0x10000 and 0x4000 is the default erase size (Sector size for Serial NOR Flash and Block Size for NAND Flash device). This erase size can be selected once all the partitions in the specific memory are removed. Also, select the location for environment variable[14][15].

Note: Since the firmware is different for saving environment variables in different memories, the board needs to be rebooted and reconnected to make the new option work.

This is a onetime setting and the partition setup is saved in a file once the Save button is clicked. It can then be accessed or modified by clicking on Setup menu -> Setup Memories/Partitions.

Note: This step is necessary for performing any erase or program operation on flash as all the operations are performed on partitions only.

```
proc SetupWindow {} {
    global mems size Partitions ShowParts
    global sb


    set es(Nor) {
      0x0 0x10000 0x40000
    }
    set es(Nand) {
      0x0 0x4000 0x10000 0x20000
    }


      if {$Partitions($i) > 0} {
        for {set j 1} {$j <= $Partitions($i)} {incr j} {
          addPart $f.lab $i old
        }
      }
    }
```

```
}

proc saveSetup {w} {

foreach mem $mems {
      set max 0
      for {set i 1} {$i <= $ShowParts($mem)} {incr i} {
        set var [expr $from($mem$i)+$len($mem$i)]
        if {$var > $max} {
          set max $var
        }
      }
      for {set j 0} {$j < $max} {incr j} {
        set check($j) 0
      }
      for {set i 1} {$i <= $ShowParts($mem)} {incr i} {
        if {$sanity == "FAIL"} {
          break
        }
        set j $from($mem$i)
        for {set k 0} {$k < $len($mem$i)} {incr k} {

      incr j
        }
      }
    }

    if {$sanity == "PASS"} {
      if {[info exists lastenvopt]} {
```

```
        foreach mem $mems {
        while {$i < $p} {
          incr i
          set line "$names($mem$i)@$from($mem$i)@$len($mem$i)"
          puts $conf $line
        }
      }

proc delPart {p mem} {
     if {$i > 0} {
       if {$i == 1} {
       }
     }
     destroy $p.$i
}
```

5.3.2 Erasing Partitions

Click on the Erase button Select the partitions to erase and click on erase button to erase the selected partitions[15].

```
proc EraseSec {p} {
     for {set i 1} {$i <= $Partitions(Nor)} {incr i} {
       if {$erase(Nor$i) == "on"} {
         set eraseall(Nor) off
         set erase(Nor$i) off

         set cmd "erase 1:$from(Nor$i)-[expr $from(Nor$i) + $len(Nor$i) - 1]"
```

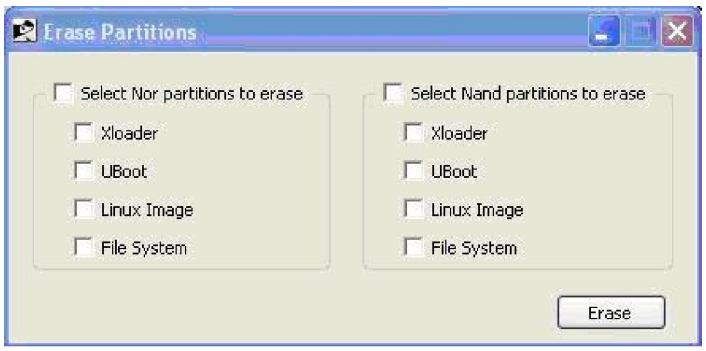

Figure 5.3: Erase memory partition

```
 }
    }

for {set i 1} {$i <= $Partitions(Nand)} {incr i} {
      if {$erase(Nand$i) == "on"} {
        set fromadd [expr $from(Nand$i) * $size(Nand)]
        set fromadd [format "%x" $fromadd]
        set length [expr $len(Nand$i) * $size(Nand)]
        set length [format "%x" $length]

        set cmd "nand erase 0x$fromadd 0x$length"
        }
    }
```

5.3.3 U-boot Through Script File

It provides an additional feature that user can run U-boot commands through script file. For that user have to write a script file or a text file that contains all the u-boot commands in sequence.

To load the script file click on "Run Script". It asks from where to get the script file and then load the file. It launches command written in the script file or text file one by one. Make sure that the script file is written properly.

```
proc runscript {} {

   if {[string compare $file ""]} {
      if {![catch {set conf [open "$file" r]}]} {
        set contents [read $conf]
        close $conf

        set lines [split $contents "\n"]
        set opstatus "Running Script"
        for {set i 0} {$i < [llength $lines]} {incr i} {
          set line [lindex $lines $i]
          if {[string compare $line ""]} {
           }
    }
```

5.3.4 Selection of SOC

This utility supports four different SPEAr devices.

a. SPEAr600

b. SPEAr300

c. SPEAr310

d. SPEAr320

```
runCmd {} {

      set command ""
      set checkcmd [string range $cmd 0 [expr [string first " " $cmd] - 1]]
      if {$checkcmd == ""} {
        set checkcmd $cmd
      }
      for {set i 0} {$i < [llength $checkcmdlist]} {incr i} {
        if {$checkcmd == [lindex $checkcmdlist $i]} {
          set endstr " "
          break
        }
```

5.3.5 Programming Partitions

Click on program button

User can select appropriate SoC according to the SPEAr device going to be used.

Select the partitions to program through check-buttons and select the binary files for each selected partition. Uncheck buttons of the NAND partition program.Click on program binaries button to write the flash.

The programming configuration can be saved in a file for further use. The configuration saved as defConf.conf is default configuration that opens when Program button is clicked. Overwrite this configuration according to your PC paths.

Note : The COM port selected in the main window should be the gadget serial COM port number. It can be seen from the device manager window.

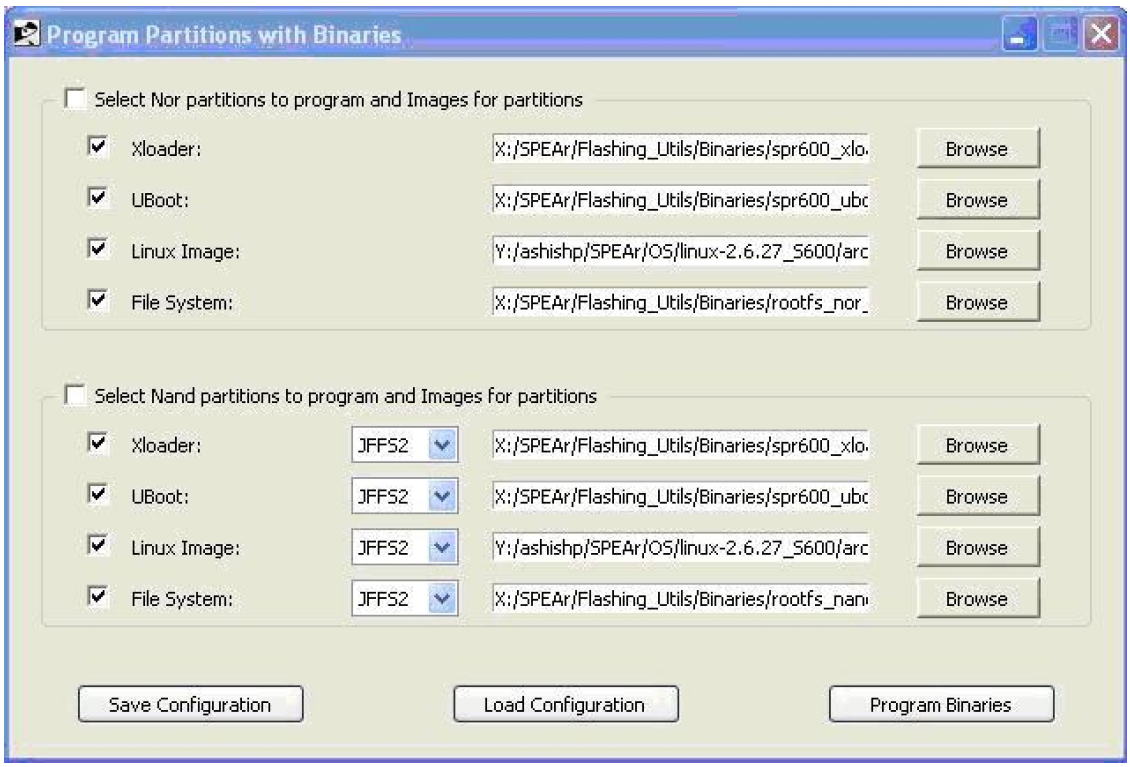

Figure 5.4: Program memory partition

```
proc ProgramWindow {} {
1.  Set the memory type NOR or NAND want to flash.
2. Check the box which partition want to flash Example: File Syatem, xloader,ubo
3. Set the type of partition type if NOR then only jffs2 and if NAND then it can
4. Write the data on the particular mtdblock address using write command.
}
proc flashImages {f w} {


for {set i 1} {$i <= $Partitions(Nor)} {incr i} {
       if {$prog(Nor$i) == "on"} {
           set filelength [file size $image(Nor$i)]
```

```
          if {$length >= $filelength} {
            set filelength [format "%x" $filelength]
            set opstatus "Uploading file \"$fname\""
            catch {ukermit $image(Nor$i)} id
            set cmd "cp.b 0x800000 0x$fromadd 0x$filelength"


for {set i 1} {$i <= $Partitions(Nand)} {incr i} {
      if {$prog(Nand$i) == "on"} {
        set fromadd [expr $from(Nand$i) * $size(Nand)]
        set fromadd [format "%x" $fromadd]
        set length [expr $len(Nand$i) * $size(Nand)]


 set filelength [file size $image(Nand$i)]

            catch {ukermit $image(Nand$i)} id

            if {$rootfs(Nand$i) == "YAFFS2"} {
              set cmd "nand write.yaffs2 0x800000 0x$fromadd 0x$filelength"
            } else {
              set cmd "nand write.jffs2 0x800000 0x$fromadd 0x$filelength"
            }
```

5.4 Bad Block Management

The FLASH chip uses the data management of block and byte. When a block of storage area became invalid, replace it with the block of backup storage area. If there is no bad block in actual storage chips, the logic address correspond with physical address completely. If the physical block which corresponds to logic block become invalid, a block of reserved storage area will be distributed to replaces it, thus the physical storage address is made to in continuity. In this condition, if we still make the logic address corresponded to physical address completely, will brings great complexity to manage data storage address. If the logic address and physical address mapping mutually, will reduce address management complexity. Particularly, a mapping table of logic address to physical address should be built, use continuous logic address when manage data, transfer the logic address to physical address through the mapping table only when access FLASH chips. The main function of the table is to make bad block replacing convenient when storing data, so we call it bad block replacement table[16][11].

We build 2 bad block tables in data buffer in our design (bad block replacement table A, B), as the bad block replacement tables of host FLASH memory and backup FLASH memory separately. The size of every bad block replacement table is 4KB. Every bad block list item occupy 2 byte, so every bad block replacement table contains 2048 items and corresponds to a storage unit of 128KB. Their data structure is shown as table 1[11].

The left side of table is the number of bad block replacement table, and it is also the logic number to access FLASH memory. The right side is 2 byte list item content-the physical block number corresponds to logic block number. If there is no bad block in FLASH chips, the physical block number equals logic block number; if the physical block number which corresponded to the logic block number is a bad block, the physical block number is the replacement physical block number. Such as the first item of

lOGIC NUMBER	*PHYSICAL NUMBER*
0000	0000
0001	1024
0002	0002

Table 5.1: Data Structure Of Bad Block Replacement Table

table, the corresponded logic block number is 0. The 0th block of FLASH chip works normally, so the table item content is 0[16].

The second item of table which logic block number is 1, but the first block of FLASH chip is bad, so we need to replace it with 1024th block which is the first fine block of backup FLASH chip. The second table item content is modified to 1024, show that the later access to the first logic block will be all mapped to the 1024th block.

5.4.1 Bad Block Singed Table

After built the bad block replacement table, a bad block signed table should be built to coordinate with it. Therefore, we can find the fine blocks with shortest time when manage bad blocks and improve efficiency. As same as bad block replacement table, two bad block signed tables should be built to correspond to the main FLASH and the backup FLASH.

PHYSICAL NUMBER	*HEALTH CONDITION*
0000	0000
0001	0FFH

Table 5.2: Structure Of Bad Block Signed Table

The BBM algorithm is defined by system modules external to the NAND device, which can be hardware and/or software components. To further complicate matters, various system components can contribute a portion of the larger algorithm. These components can be from multiple suppliers. Obtaining a comprehensive BBM algorithm specification from a sole source may not be achievable. Components that may contribute to the BBM scheme include, but arent limited to: commercial, open-source, or proprietary flash file system software and drivers, microcontrollers or chipsets with integrated NAND controller hardware, and hardware IP core NAND controllers.

To help with specifying BBM algorithms for production programming, six areas are important. These include:

a. Bad Block Replacement Strategy

b. Partitioning

c. Error Correction Codes **ECC!**

d. Spare Area Placement

e. Free Good Block Formatting

f. Dynamic Metadata

There are other aspects of bad block management, such as wear leveling, block reconditioning, and garbage collection that are important for the target system to implement. However, only the above listed areas are a requirement during factory programming.

5.5 Bad Block Manage Algorithm

BBM The bad blocks of FLASH chip includes several types of initial bad block, areas bad block, program bad block, read bad block etc. any type of bad block would lead

to a condition that the written data in bad block can not be read out correctly. In order to simplify logic design, we use uniform readout checking method to all types of bad blocks. In actual implication, in order to reduce start time, we proposed a new real-time bad block recognized and replaced method compare to normal replacement algorithm. When power-up originally, we do not make FLASH check itself. We assumed that all blocks in FLASH are fine. When recording data, we should erase the block before write data on the first page of it[16].

If erase failed, the block is considered as bad block and then replaces it with backup

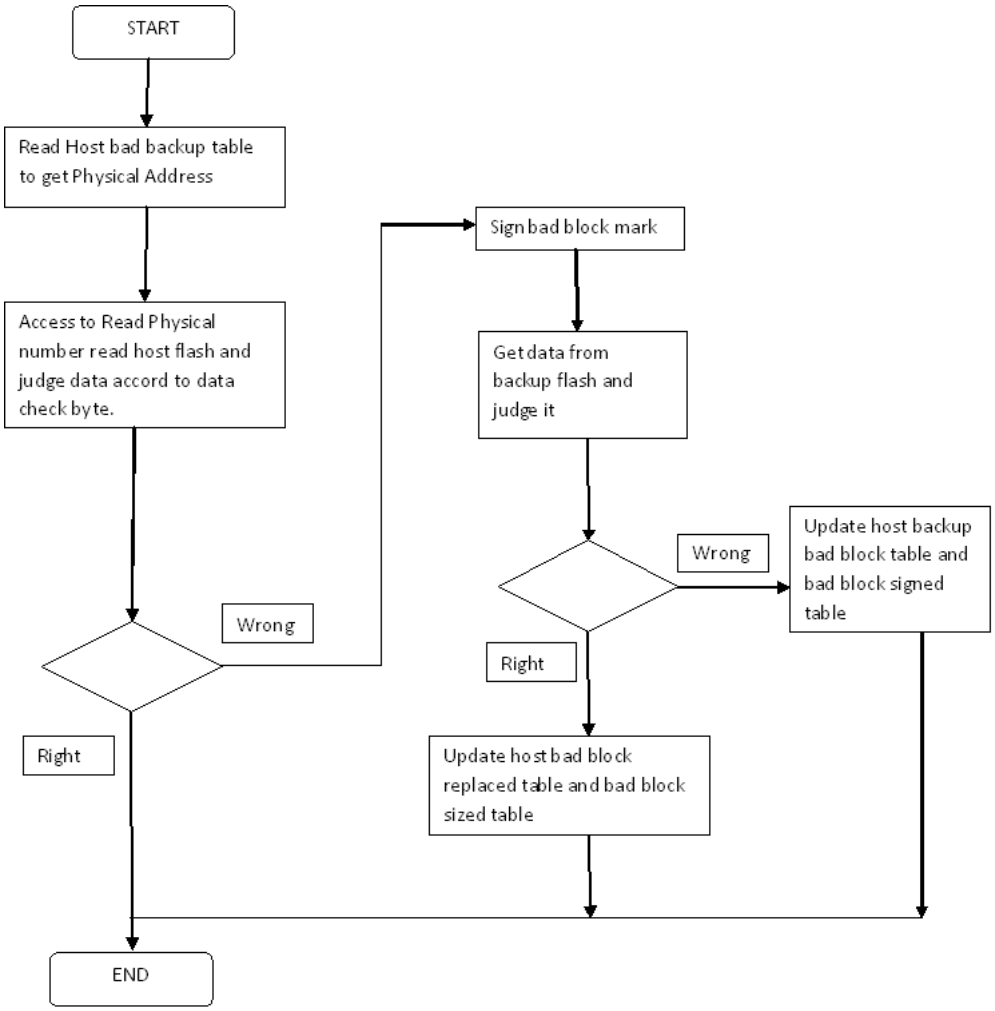

Figure 5.5: Replacement Algorithm

block. Then erased again, if failed too, continue to replace with backup blocks until the data can write into the replacement bad block. In order to provide enough redundancy blocks, we select larger capacity FLASH chips according to the requirement of storage capacity and supply of goods. The bad block replacement process chart is shown as figure.

Chapter 6

Testing and Results

6.1 Testing of Power Management

6.1.1 Test setup:

- SPEAr Board: SPEAr600,SPEAr300,SPEAr320
- Temperature controller(used to vary temp from -10 to 140)
- Voltage and Current analyzer(used to measure current and voltage at different jumper)
- Linux HOST PC
- DDR Testing utility

Analyzer is used to provide external voltage supply and measure the current at different jumper. Connecting this analyzer to the SPEAr board and boot the board and measure the current consume in normal mode.Also measure the current in different three situation:

- When DDR test running.

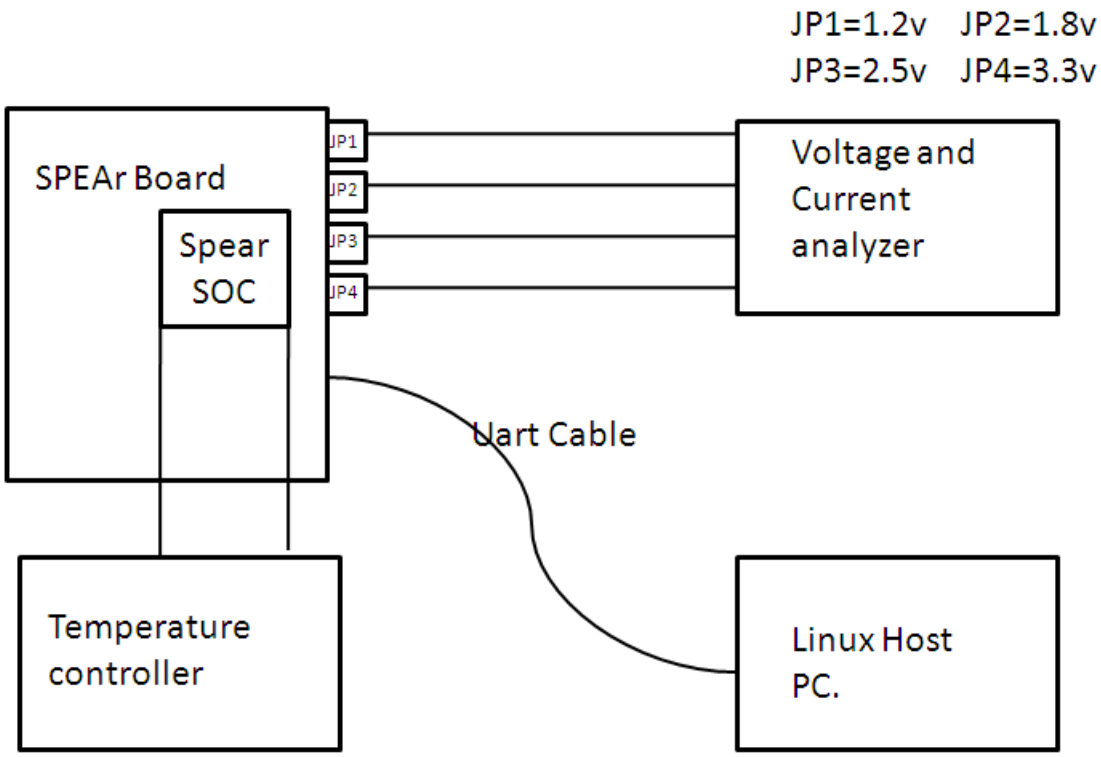

Figure 6.1: Power Management Test Setup

- When Copying data from pen-drive.
- Suspending and resuming process.

6.1.2 Using CPUfreq

After completing Power management testing in normal mode recompile linux kernel with enabling cpufreq option. To enable cpufreq in kernel image run following commands:

make menuconfig;

select cpufreq option:

make uImage;

Repeate the above steps with different governors ondemenad,powersave.performance.

echo powersave¿/sys/devices/system/cpu/cpu-0/scaling-governors

Testing Results For SPEAr-300

Operation	*JP1(1.0v)*	*JP2(1.8v*	*JP3(2.5v)*	*JP3(3.3v)*
Linux Prompt	226	47	25	7
Copy process	235	58	26	7
Power Save Mode	153	42	25	7
On Demand Mode	190	42	25	7
Suspend	140	32	25	7
Resume	190	42	25	7

Table 6.1: Power Management testing on SPEAr-300

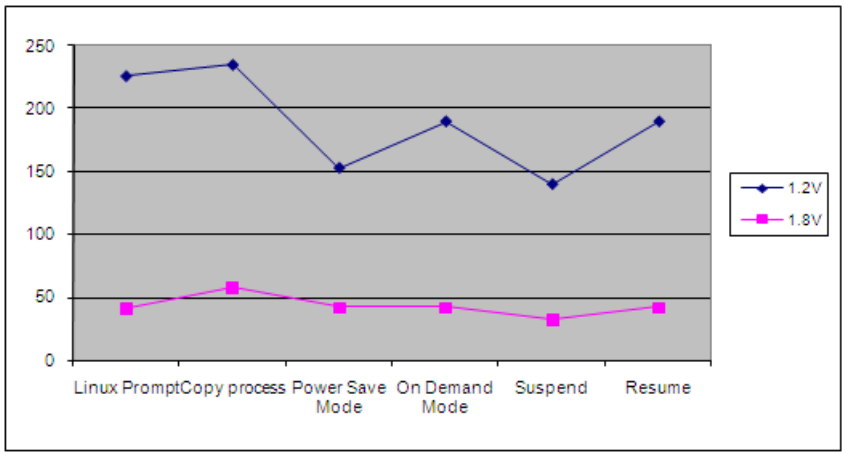

Figure 6.2: Power Management chart for SPEAr-300

Testing Results For SPEAr-320

Operation	*JP1(1.0v)*	*JP2(1.8v*	*JP3(2.5v)*	*JP3(3.3v)*
Linux Prompt	233	134	26	37
Copy process	240	140	26	37
Power Save Mode	162	134	26	37
On Demand Mode	190	134	26	37
Suspend	116	77	25	37
Resume	190	134	25	37

Table 6.2: Power Management testing on SPEAr-320

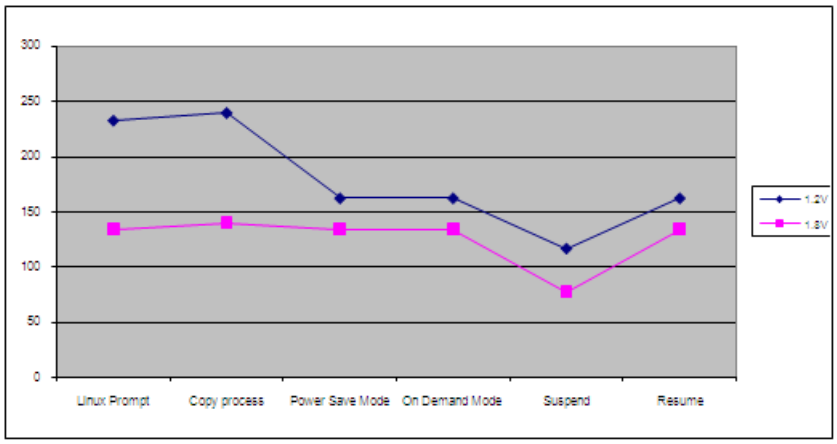

Figure 6.3: Power Management Chart for SPEAr-320

Testing Results For SPEAr-600

Operation	*JP1(1.0v)*	*JP2(1.8v*	*JP3(2.5v)*	*JP3(3.3v)*
Linux Prompt	340	46	10	10
Copy Process	380	66	10	10
Power Save Mode	270	46	10	10
On Demand Mode	300	46	10	10
Suspend	217	36	10	10
Resume	300	46	10	10

Table 6.3: Power Management testing on SPEAr-600

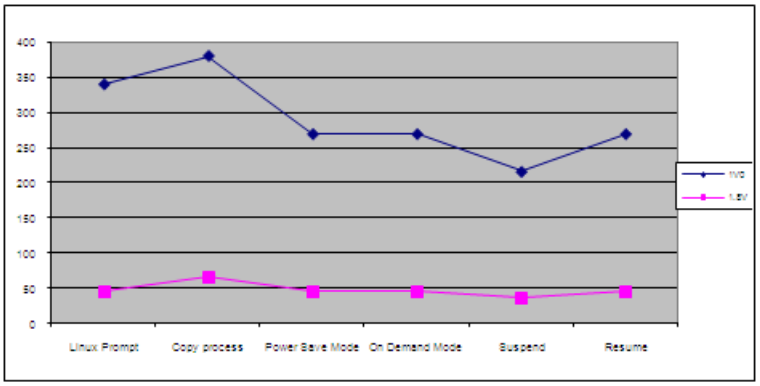

Figure 6.4: Power Management chart for SPEAr-600

6.2 Performance of NAND Flash Memory with Bad Bloack Management Algorithm

The performance measurement has been performed using:

- Hardware: ARM926EJS (333MHz), STMicroelectronics NAND 512W3A2CZA6 flash.
- Test File-system on the NAND Flash: YAFFS2 and JFFS2 (for /dev/mtdblock7)
- Kernel: linux-2.6.27.
- Driver: DMA was disabled

Result for YAFFS2The steps for testing NAND Flash memory is as follow:-
Mount YAFFS2 fs and write/read to a file. Sequence is as follows:

- mount t yaffs2 /dev/mtdblock7 /mnt
- time dd if=/dev/zero of=/mnt/file.bin bs=4K count=8192 (Write the file)
- umount /mnt
- mount t yaffs2 /dev/mtdblock7 /mnt
- time dd if=/mnt/file.bin of=/dev/null bs=4K count=8192 (Read file back)
- umount /mnt
- mount t yaffs2 /dev/mtdblock7 /mnt
- rm /mnt/file.bin
- umount /mnt
- Repeat for bs = 8, 16, 32, 64K and count=4096, 2048, 1024, 512.

Size	*Count*	*MB*	*Write time*	*MBPS*	*Read time*	*MBPS*
4K	8192	32	49.5	0.64	21.9	1.46
8K	4096	32	49.7	0.64	21.9	1.46
16K	2048	32	49.6	0.64	21.9	1.46
32K	1024	32	49.3	0.65	21.9	1.46
64K	512	32	49.6	0.64	21.9	1.46

Table 6.4: Performance of NAND memory

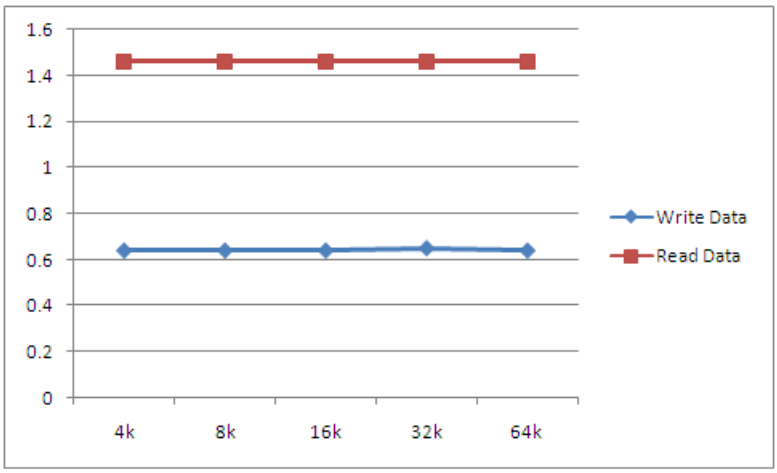

Figure 6.5: NAND Performance chart

Test result for Spear-600 NAND Performance Chart: In order to detect the initial bad blocks (generated during factory production) and to handle run time bad blocks (generated by program/erase usage).It remaps a bad block to one of the reserved blocks so that the data contained in a bad block is not lost and new data writes on a bad block is avoided.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

Dynamic power management is an effective means for system-level design of low-power embedded systems. Dynamic power management is already widely applied to system design. The experiments were conducted with different ST Spear architecture, with different temperature. Experimental results shows amount of power that can be save when the system is sitting ideal for some time or the load on the system varies. The implementation of Dynamic Power Management algorithm increases battery life for Embedded system.

Flashing utility and bad block management algorithm improve performance of NAND flash in Embedded system.Flashing utility used to program the various embedded system flash memory partitions.By using this utility user can program memory partition at run time according to user requirement.Bad block Management algorithm improvers the over all read and write time of the memory.

7.2 Future Scope

The current ondemand governor depends on the idle/busy statistics collected at the scheduler ticks. If at the tick instance the CPU was idle, then whole tick is considered idle and vice-versa. But, if we can do a microaccounting of idle time then we get a more accurate number of time spent idle and time spent doing useful work. The kernel can do the micro-accounting by noting the time of entry and exit of idle routine and interrupts.

ukermit and usb_send are two application used in Flashing utilization. Using these two utility data is flash in NOR or NAND memory and behind these data transfer USB host is used as a medium.But the data transfer rate is slow as compare to USB host. So optimization of flashing utility can be possible.

Appendix A

ST SPEAr^R Architecture

ST Microelectronics customizable processor family, called SPEAr^R, supplies a powerful digital engine that offers the possibility of designing special user functions with very low development time and cost. The new family is based on an ARM core architecture that maximizes hardware and software performances, it includes an advanced bus system and IPs for connectivity and memory interfaces. The user functions can be embedded in the configurable logic block[3].

SPEAr^R concept provides high performances SoC and full custom design capabilities with a quick development cycle.

A.1 SPEAr^R Technology

SPEAr^R products are SoC with embedded functional blocks and innovative design model that makes a project customization easy, reliable, quick, and affordable.The typical **ASIC!** design models allow the designer to put together functional blocks according to a project requirements. SPEAr^R enables designers to embed functional blocks (standard or custom IP) within a pre-designed, tested, and validated system architecture featuring leading ARM processors, peripheral/memory controllers and state of the art connectivity.

Adapting the project requirements to SPEAr^R architecture is easy, fast, and reliable. The customizable logic accepts any digital project requirements by interconnecting metal and via layers, combining "logic tiles" with a very high integration yield. Moreover the customizable logic allows the design of a single platform device to respond to changes in the customer's specification.

A.2 SPEAr^R architecture

Of course SPEAr^R family components are different: single/dual cores, customizable logic size, connectivity and so on, but have been designed sharing some important architectural concepts and therefore providing scalable solutions:

a. State of art ARM cores (from ARM9 to Cortex A9) single/dual cores

b. Very good memory interface; especially for the DRAM path. Multiport memory controllers, providing buffers for each SoC internal path (main IPs are connected via dedicated bus), are used to solve latency issues and optimize memory access.

c. Rich set of connectivity IPs (USBs, Ethernet etc.) having dedicated busses for the memory controller ports connection.

d. HW accelerators and HMIs (JPEG codecs, crypto engines, LCD controllers with touchscreen capabilities etc.)

e. Internal multilayer bus matrix to avoid bottlenecks and latency issues accessing "low speed peripherals"

f. A set of legacy IPs (UARTs, timers, ADC etc.)

g. The customizable block with SRAM resources. SRAM blocks (single and dual ports) can be configured to act as IPs buffers (FIFOs, local memories et.) or

directly accessible by the ARM core like other "Master peripherals".

Each memory cut has dedicated address/data/control paths connected to the customizable logic so the customization process can freely create the required memory sizes (different depths and widths).

Since each memory cut is independent, the total provided bandwidth is really impressive.For example SPEAr600 provides 28 singleport and 12 dualports SRAM cuts (32bits data wide), so running them at 166MHz it can achieve 276 Gbits per second (34.5GBytes per second: [28 + 2 x 12] x 4 x 166MHz).

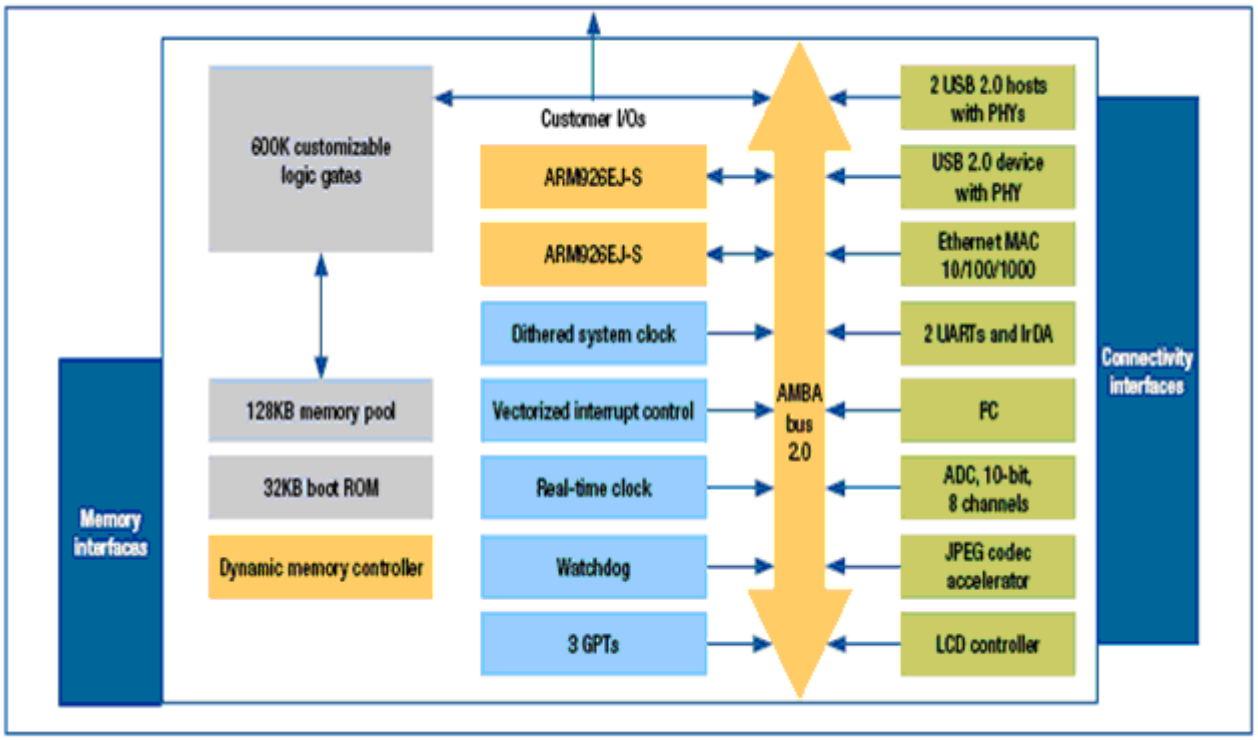

Figure A.1: SPEAr Architecture

Features

a. Processor: ARM926EJ-S running at 333MHz

b. 32 Kbytes of Instruction cache, 16 Kbytes of Data cache

c. 8-Kbyte Data-TCM (Tightly Coupled Memory), 8-Kbyte Instruction-TCM

d. 3 USB2.0 ports (two hosts and one device supporting high speed mode)

e. Ethernet 10/100 MAC

f. 16-channel 8-bit A/D converter

g. I2C interface, 3 UARTs

h. SDRAM memory interfaces at 133MHz supporting DDR and SDR

i. SPI interface supporting serial FLASH/ROM

j. 1 full USB-dedicated PLL and one dithered system PLL

k. 200-kgate (ASIC equivalent) of configurable logic connected to four banks of 4 KBytes SRAM each

l. A Real Time Clock, Watchdog and 4 general-purpose timers complete the SoC structure

m. Supports a wide range of operating systems, including Linux, Nucleus, uItron, and Vxworks.

Website References

[1] http://www.linuxjournal.com/article/6699

[2] http://en.wikipedia.org/wiki/EmbeddedLinux

[3] http://embedded-system.net/spear-basic-customizable-arm-based-soc-stmicroelec
html

[4] http://www.ibm.com/developerworks/linux/library/l-embl.html

[5] http://www.embedded-computing.com/articles/singh/

[6] https://oprofile.sourceforge.net

[7] http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq.html

[8] http://acpi.sourceforge.net

[9] http://www.datio.com/nand/nandflash.asp

[10] http://tali.admingilde.org/linux-docbook/gadget/ch05.html

[11] http://www.omen.com/knt.html

[12] http://www.st.com/stonline/stappl/cms/press/news/year2006/p2104.
htm

[13] http://blog.datalight.com/tag/bad-block-management

References

[1] Carl van Schaik Ben Leslie and Gernot Heiser. A portable user-mode linux for embedded systems. *National ICT,Australia, Sydney, Australia*.

[2] Alain Mosnier. Embedded/real-time linux survey. July 2005.

[3] ST Microelectronics. *SPEAr-300 User manual*.

[4] Pratyush Anand. Testing strategy for spear power managment. *Department: CPG-CSD, Noida,ST Microelectronics*.

[5] William E. Cohen. Tuning programs with oprofile. *WIDE OPEN MAGA-ZINE*, 2004.

[6] Deepika Dhamija Karun Saraswat. Linux regression platform. *Department: CPG-CSD, Noida,ST Microelectronics*.

[7] A. Bogliolo G. A. Paleologo, L. Benini and G. D. Micheli. Policy optimization for dynamic power management. *In Design Automation Conference*, pages 182–187, 1998.

[8] Alessandro Bogliolo Giovanni De Micheli, Luca Benini. Dynamic power management of embedded systems.

[9] Lattice Semiconductor. Dynamic power management in an embedded system. *A Lattice Semiconductor White Paper*.

[10] IBM and MontaVista Software. Dynamic power management for embedded systems.

[11] SAMAUNG. *BAD BLOCK MANAGEMENT Application Note*.

[12] BPM Microsystem. Understanding nand flash factory programming. October, 2008.

[13] Alexey Starikovskiy Venkatesh Pallipadi. The ondemand governor. *Intel Open Source Technology Center*.

[14] Armando Vipin Kumar. Flashing utility requirements. *ST Internal*.

[15] Vipin Kumar Shiraz Hazhim. Bootrom spear300. *ST Internal*.

[16] ST Microelectronics. *Nand Bad Block Replacement method*, September 16, 2009.

Index

Bad Block Management, 49
BBM Algorithm, 51

CPUfreq Algorithm, 35
CPUfreq Scaling, 31

Dynamic Power Management, 24

Flashing Utility, 39
Frequency power relation, 32

Governors, 33

Introduction, 1

Linux Regression, 14

Oprofile, 12
OProfiler installation steps, 13
OSPM, 23

Power Management, 21

Resume, 27

SPEArR architecture, 64
Suspend, 27

Test Result, 54