申请上海交通大学工程硕士学位论文

Android 智能手机的低功耗设计与实现

论文作者：陈实凡

学　　号：1120332068

交大导师：邓倩妮

企业导师：杨昌桦

院　　系：计算机科学与工程系

工程领域：计算机技术

上海交通大学电子信息与电气工程学院

2016 年 9 月

Thesis Submitted to Shanghai Jiao Tong University

for the Degree of Engineering Master

LOW POWER DESIGN AND IMPLEMENTATION OF THE ANDROID SMARTPHONE

M.D. Candidate : Chen Shifan

Student ID : 1120332068

Supervisor(I) : Deng Qianni

Supervisor(II) : Yang Changhua

Department : Computer Science & Engineering

Speciality : Computer Technology

School of Electronic Information and Electric Engineering
Shanghai Jiao Tong University
Shanghai,P.R.China

Sep, 2016

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其它个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□ 在 __年解密后适用本授权书。

本学位论文属于

不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

Android 智能手机的低功耗设计与实现

摘 要

近几年，由于移动互联网技术与移动通信技术快速的发展，Android 智能手机也得到了快速的普及，在全球市场占有率已超过 75%，Android 总用户数已超过 Windows 操作系统，成为最受用户欢迎的操作系统，但续航能力不足是 Android 系统的一大缺点。由于智能手机硬件不断升级，功能与使用场景也更加丰富，导致系统功耗也在不断增加，但智能手机电池容量增长非常有限，如何通过系统低功耗设计降低智能手机功耗是一个值得研究的问题。

论文基于高通 MSM8953 硬件芯片平台与 Android 软件平台进行系统低功耗设计与实现，通过唤醒锁管理、后台清理、联网管理和省电模式四个关键模块，在不影响用户正常使用与体验的前提下进行系统监控与管理，及时释放 CPU 资源，使系统在空闲时能够顺利的进入低功率模式，从而降低系统功耗达到省电的目的。

论文主要工作内容包括如下：

1) 通过对 Android 系统的唤醒锁进行管理，在不影响用户正常使用的前提下及时的释放 PARTIAL_WAKE_LOCK 类型的唤醒锁，避免系统不能及时进入睡眠状态而导致额外耗电。

2) 后台进程清理的目的是清理在后台异常占用 CPU 的进程，以避免 CPU 一直处于高频率的运行状态而造成不必要的耗电。

3) 应用联网管理是通过对 IPTable 进行设置，禁止未经过用户允许的应用程序在灭屏状态下联网，但在亮屏状态下恢复应用的联网权限，从而不影响用户的正常使用。

4) 省电模式为用户提供一个极端省电的系统，在省电模式下会关闭极大部分可能耗电的设备，关闭后台应用与进程，手机上只保留联系人、通话、短信以及闹钟最基本的四个功能，已杜绝其他可能耗电的情况，以满足用户超长续航的需求。

最后，论文对系统省电管理方案进行了测试与验证，测试在各使用场景下的平均电

流，计算出了手机在各场景下的使用时长。同时分别在 wifi 与 4G 状态下测试样机的平均待机电流，用测试数据验证了优化后在 wifi 与 4G 状态下待机下的待机电流较优化前分别降低了 11%与 13%，达到了系统省电管理的预期效果。

关键词 电源管理，低功耗，省电，安卓

LOW POWER DESIGN AND IMPLEMENTATION OF THE ANDROID SMARTPHONE

ABSTRACT

In recent years, due to the mobile Internet technology and mobile communication technology development, the popularity of Android smartphones also got rapid, in the global market share has exceeded 75%, Android total number of users has exceeded the Windows operating system, one of the most popular with the users of the operating system, but insufficient battery life is a big drawback of the Android system.Due to escalating smartphone hardware, function and usage scenario is also more abundant, cause the system power consumption is also increasing, but the smartphone battery capacity growth is very limited, how to use low-power system design reduce the power consumption of the smart phone is a problem worthy of studying.

Paper based on qualcomm MSM8953 chip hardware platform and Android software platform for low-power system design and implementation, through the wake lock management, clear background, network management and power saving mode of four key modules, in does not affect the normal use of users with experience under the premise of system monitoring and management, timely release of the CPU resources, make the system would be able to in your spare time into low power mode, so as to reduce the power consumption of the system to achieve the purpose of energy saving.

The primary work in this thesis are presented as follows:

1) through the study of the management of Android wake lock, in does not affect the normal use under the premise of user timely release PARTIAL_WAKE_LOCK type wake locks, avoid system not to sleep in time and cause additional power consumption.

2) is the purpose of the background process cleaning process of abnormal CPU i

n the background, in order to avoid the high frequency operation of the CPU has bee n caused by the unnecessary power consumption.

3) the application of network management is based on set up iptables, prohibited without the user allows the application to screen state connected to the Internet, but i n the bright screen to restore the application under network access, which does not af fect the normal use of the user.

4) the power saving mode to provide users with an extreme save electricity syste m, in power saving mode will be closed may greatly part of power equipment, close the backend application and process, only keep on your mobile phone contacts, calls, t ext messages, as well as the alarm clock the four basic functions, power consumption conditions might have put an end to the other, to meet the needs of users for long l ife.

Finally, the thesis tested the system energy saving management plan and verificati on, test under different usage scenarios of the average current, calculate the use of m obile phones in all scenarios.At the same time respectively in wifi and 4 g state the average standby current test prototype, with test data validate the optimized under the wifi and 4 g in the state of standby on standby current was reduced by 11% and 1 3% respectively than before optimization, achieve the expected effect of system energy saving management.

Keywords: Power Manager, Low Power, Power Saving, Android

目 录

第一章 绪 论 1

1.1 选题背景与研究意义 1
1.2 相关技术与研究现状 3
1.2.1 硬件低功耗设计 3
1.2.2 软件低功耗设计 4
1.2.3 智能手机功耗研究现状 6
1.3 研究内容与意义 7
1.4 论文组织结构 8

第二章 电源管理 9

2.1 LINUX 电源管理 9
2.1.1 APM 与 ACPI 9
2.1.2 设备电源管理模型 13
2.1.3 Linux 休眠与唤醒 14
2.2 ANDROID 电源管理 14
2.2.1 Android 电源管理结构 15
2.2.2 电源管理服务 16
2.3 ANDROID 电源管理新特性 17
2.3.1 唤醒锁（wake lock） 17
2.3.1 Early Suspend 和 Late Resume 17
2.4 本章小结 19

第三章 总体设计与平台架构 20

3.1 总体设计 20
3.2 硬件平台 21
3.2.1 MSM8953 芯片平台概述 21
3.2.2 硬件平台架构 22
3.2.3 Big.Little 处理器架构 24
3.3 软件平台 25
3.3.1 Android 操作系统概述 25
3.3.2 Android 软件平台架构 25
3.3.3 Android 应用四大组件 27
3.4 手机功率系统 27
3.4.1 手机供电系统 27
3.4.2 系统功率树状图 28
3.4.3 CPU 低功率模式 29
3.4.4 系统低功率模式 30
3.5 本章小结 31

第四章 关键模块设计与实现 32

4.1 唤醒锁管理 33
4.1.1 唤醒锁概述 33
4.1.2 唤醒锁管理策略 33
4.1.3 设计与实现 34
4.2 后台进程清理 35
4.2.1 Android 进程概述 35
4.2.2 后台清理策略 36
4.2.3. 设计与实现 37
4.3 应用联网管理 39
4.3.1 联网管理概述 39
4.3.2 联网管理策略 39
4.3.3 联网管理设计与实现 40
4.4 省电模式 42
4.4.1 省电模式概述 42
4.4.2 省电模式策略 43
4.4.3 设计与实现 43
4.5 本章小结 44
第五章 测试与验证 45
5.1 测试工具与环境 45
5.1.1 Power Monitor 使用 45
5.1.2 绘制曲线图 46
5.2 测试数据与分析 48
5.2.1 场景功耗测试 48
5.2.2 后台耗电测试 49
5.3 本章小结 50
第六章 总结与展望 51
6.1 论文工作总结 51
6.2 未来展望 51
参考文献 53
致 谢 55
攻读学位期间发表的学术论文 56

第一章 绪 论

1.1 选题背景与研究意义

2008 年 9 月，谷歌发布了 Android 操作系统的第一个版本，随后 HTC 生产出了第一部 Android 操作系统手机 G1[1]。由于谷歌的开源策略并公开了 Android 源码，大量的厂商加入到 Android 阵营，促使 Android 系统的发展非常迅速。Android 在国内也受到欢迎，带动着国内从功能机到智能机的转型。加上国内运营商网络的快速发展和推广，从 2G 到 3G，再到现在的 LTE,移动网络也越来越成熟，使得智能机迅速地普及起来，移动终端用户数也不断增长，截至 2015 年 11 月份国内手机上网用户总数已超过 9.05 亿[2]。手机也慢慢变成大家生活中不可或缺的一部分，公交地铁、吃饭聚会上出现地“手机党”也越来越多，与台式机相比，移动设备更加便携，触屏操作也非常简单实用，方便随时随地使用移动设备，极大的丰富了使用场景。同时由于手机硬件不断地升级，处理器性能和频率、屏幕分辨率和尺寸等都在提高，导致手机的功耗也相应增加。而手机上的应用程序和功能也越来越丰富，用户每天使用手机的平均时长也在不断增加，所以手机用的更多与手机耗电更大的矛盾日益突显，用户也开始抱怨智能手机电池越来越不经用，不能满足用户需求，“一天一充”甚至“一天多充”的现象非常普遍，出门带着充电宝和边充电边使用手机的应用场景也很常见。手机耗电和续航能力不足的问题逐渐成为智能手机未来发展的一道障碍，使得移动设备用户不能充分享用到移动互联网的便捷，续航问题严重影响了用户体验。降低智能手机的功耗并提升续航能力也成为手机产品的一大买点，是各大芯片厂商和手机制造商迫切需要解决的问题。

智能手机与台式机不同，完全采用内置电池供电，受限于移动设备的体积大小，内置电池体积容量也非常有限，现在主流智能手机内置电池容量大概在 2500mAh 到 4000mAh 之间，在不增加电池体积的情况下想要显著地提升电池容量很困难，几乎行不通，所以只能采用其他方法来提升手机的续航能力。主要的方法是降低手机功耗，通过降低智能手机使用时的平均电流，从而达到降低手机功耗

和提升续航时间的目的。

手机耗电可以分为两个方面：硬件耗电与软件耗电。硬件耗电是指组成智能手机的各种元器件在使用过程消耗电量，主要的硬件耗电单元有 CPU、屏幕、GPU、射频、天线、WIFI 和传感器等，各器件的耗电量也和具体地使用情况有关，例如打电话用的比较多，手机射频的耗电占比就回增加。手机使用过程中各硬件耗电占比如图 1-1 所示，CPU、屏幕和数据网络是耗电占比最多的三个耗电单元，因为它们也是手机上最常用的三个部分，只要手机亮屏，CPU、屏幕和网络基本就在运行状态，所以 CPU、屏幕和网络这三部分总耗电量一般占到整体耗电量的 50-75%。

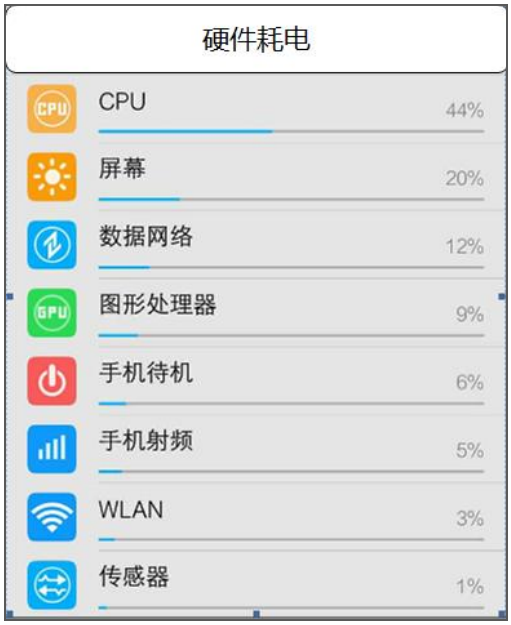

图 1-1 硬件耗电

Fig.1-1 Hardware power compression

软件耗电本质上也是硬件耗电，手机应用程序软件中的各种功能实际也是通过硬件实现，应用程序会使用到手机上的各种硬件资源。例如拍照软件，在拍照的过程中会使用 CPU、GPU、屏幕、背光、摄像头、闪光灯和传感器等器件，固。手机在关机状态，所有硬件和软件都是关闭状态，此时基本没有耗电；当手机运行的时候，操作系统和应用程序会使用到 CPU、内存、屏幕和其他一些硬件去满足用户的使用功能，所以会产生一定的功耗。各软件使用到的硬件资源不同，所以使用各软件的功耗也不一样。如图 1-2 所示，是手机在常用的使用场景下的平均电流大小。打游戏的时候手机平均电流最大，达到了 600mA，因为游戏会占用很多 CPU、GPU 和屏幕的资源，导致功耗也较大；听音乐的时候手机平均电流较

小，仅 100mA，因为音乐播放器可以挂在后台播放，在放音乐的时候，手机屏幕和背光等也是关闭状态，占用的 CPU 资源也比较小，固功耗偏小。假如手机电池容量为 3000mAh，充满电的情况下一直打游戏只能续航 5 个小时，而充满电的情况下能听 30 个小时的音乐。应用程序的耗电除了具体实现的功能，也和自身程序的设计和内部结构有关系，差的应用程序软件没有合理的利用并及时释放硬件资源，可能导致系统出现耗电异常，对应用程序的耗电方面进行监控和管理也很有必要研究的课题。

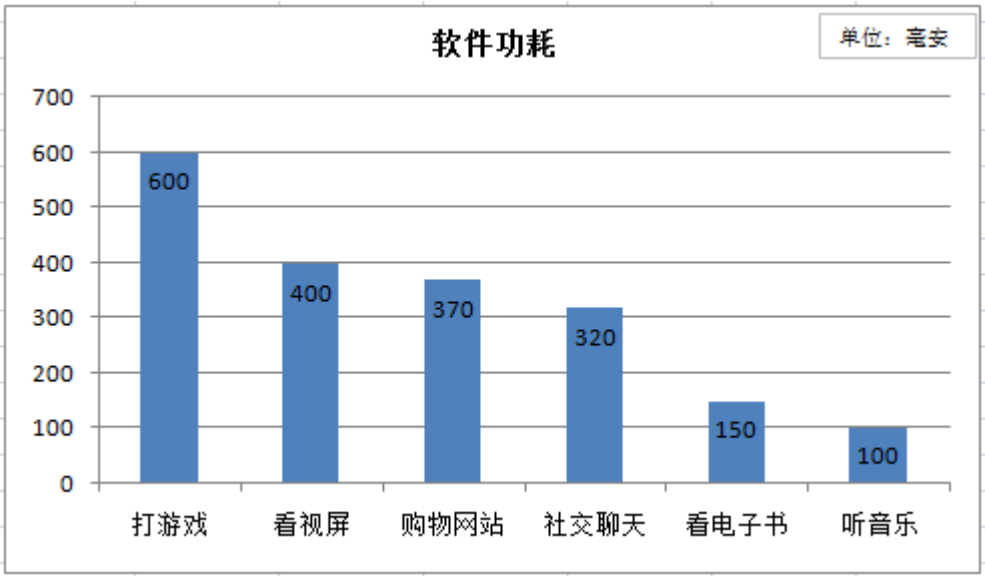

图 1-2 软件功耗

Fig.1-2 Software power compression

1.2 相关技术与研究现状

智能手机续航不足的问题，在电池容量方面很难有重大突破和提升的情况，只能下功夫去研究省电方面的方向，对于智能手机的省电研究主要也分为以下两个研究方向：硬件低功耗设计和软件低功耗设计。手机芯片厂商和硬件厂商主要在研究硬件地低功耗设计，使硬件在实现相同功能的条件下功耗更低，而操作系统厂商和手机制造商更多的是在操作系统和软件应用层面去进行省电管理的研究，通过减少系统和应用程序一些不必要地耗电，从而降低手机整体功耗，提升续航，下面对国内外相关地硬件低功耗设计和软件低功耗设计研究现状进行详细介绍。

1.2.1 硬件低功耗设计

1） 芯片级

CPU 芯片处理器是手机的一个重要组成部分，也被称为智能手机的“大脑”，负责运算和处理数据。CPU 是手机的一个主要耗电单元，固通过降低 CPU 的功耗，也能在一定程度上降低手机的整体功耗。CPU 主要由 CMOS 集成电路组成，CMOS 集成电路的功耗分为两部分：CMOS 状态改变产生的动态功耗与漏电流产生的静态功耗。动态功耗跟工作电容 C,工作电压 V 和工作频率 F 均为正比关系，动态功耗 $P=CV^2F$。静态功耗跟漏电流 I 与工作电压 V 相关，静态功耗 P=IV，在目前先进的制造工艺条件下，输入阻抗与输出阻抗均能达到很大的值，从而使得漏电流趋近与 0，故静态功耗 P 也趋近于零。在芯片级低功耗设计中，几乎可以忽略静态功耗的损耗，主要考虑如何降低 CMOS 的动态功耗[3]。

对于芯片级的低功耗设计，芯片厂商主要通过两种途径去降低 CMOS 的动态功耗。一方面通过新的芯片架构设计，例如多核技术，通过采用多核的方式分配计算任务，从而降低单个核的工作频率，达到降低动态功耗的目的；另一方面通过不断提升芯片制造工艺，缩小晶体管的体积，在芯片中集成更多的东西，推出更小纳米制程工艺的芯片，降低工作电压。高通公司推出的骁龙 625（MSM 8953）芯片，采用了八核技术，芯片制造达到了更小的 14nm 制程工艺，并通过数字信号处理器（DSP）减少 CPU 的任务负载量，实现更低的功耗和更快的处理速度[4]与它的上一代处理器骁龙 617（28nm）相比，晓龙 625 处理器的性能更加强大，更重要的是处理器场景功耗能够降低 20%-30%，整体功耗节省达到了 35%。

同时为了弥补续航不足，需要经常充电的问题，各大厂商也在积极研究快速充电技术。高通最新发布的 Quick Charge 3.0 已是第三代快充技术，提供 5V 到 20V 的充电电压，提高了充电效率，是普通充电技术的 4 倍，只用 30 分钟就将一部智能手机的电量从 0 充到 70%以上，可以有效的缩短用户充电时间[6]。自从 2012 年发布了 Quick Charge 1.0 快充技术以来，使用快充的用户已超过 3 亿。

1.2.2 软件低功耗设计

2) 系统级

对于 Android 操作系统耗电与续航的问题，也是谷歌公司比较关注的，每年推出的 Android 新版本一直在对 Android 系统中存在的问题进行持续改进与优化，同时新的 Android 版本也会包含一些新的特性和功能。最近推出的 Android

M 版本也在电源管理部分做了优化与改进，并引入了新特性。在 Android M 中提供了 2 种延长手机续航的功能，对空闲的设备和应用引进了新的省电管理机制：Doze 和 App Standby 模式[7]。

1、打盹模式（Doze Mode）：如果手机在非充电情况下保持灭屏状态达到一定的时间，系统就会自动进入到打盹模式（Doze）。在该模式下，系统将保持在睡眠状态，但会在一定的时间间隔内短暂的进入到正常状态，以便应用程序能够进行同步操作，同时使系统能够执行任何待定的操作。

2、应用准备模式（APP Standby Mode）：若用户在一段时间间隔内没有使用和触碰一个应用程序，系统将确认该应用处于空闲状态，将使该应用进入到应用准备模式。如果智能手机不在充电状态，应用准备模式下应用程序将被断开网络连接，同时系统会挂起这些应用的同步操作和任务。

3) 应用级

现在的智能手机上通常会安装几十个甚至上百个 APP（Application，应用程序），除了手机厂商系统内置的一些 APP，用户也会安装部分 APP，很多是由第三方公司或个人开发的，并没有统一的标准和规范，大多数的应用开发者只关注功能的实现，没有过多的考虑对系统资源的合理利用和对系统功耗的控制。某些 APP 在运行的时候会使 CPU 运用在过高的频率或者对系统资源造成浪费，严重的还会导致手机系统电流偏大，造成手机的异常耗电甚至发烫等问题。另外某些网络应用即使没有被使用，也会在后台偷偷唤醒系统连接网络和传输数据，也会造成耗电。其中某厂商便推出过一种对齐唤醒机制来减少后台应用频繁唤醒系统次数，并减少耗电。

正常的唤醒机制下，应用程序通过闹钟服务设置定时器，到达定时时间，通过 RTC 调用将系统唤醒并执行相应操作，由于每个应用程序设置的唤醒时间不一样，可能每分钟都有一个应用唤醒系统，这样会导致系统不能正常休眠，从而导致了异常耗电。而齐唤醒机制，原理就是为没有设置在同一个时间点唤醒的应用程序设定一个集中唤醒时间点，假如设定的集中唤醒时间为 5 分钟，那么在这个 5 分钟之内，统计有请求唤醒的应用，但不会立即执行唤醒操作，直到集中唤醒时间点，唤醒所有发出过请求的应用。对齐唤醒机制能够有效减少待机状态下系统频繁被唤醒的情况，延长系统待机的时长，达到省电的目的[8]。

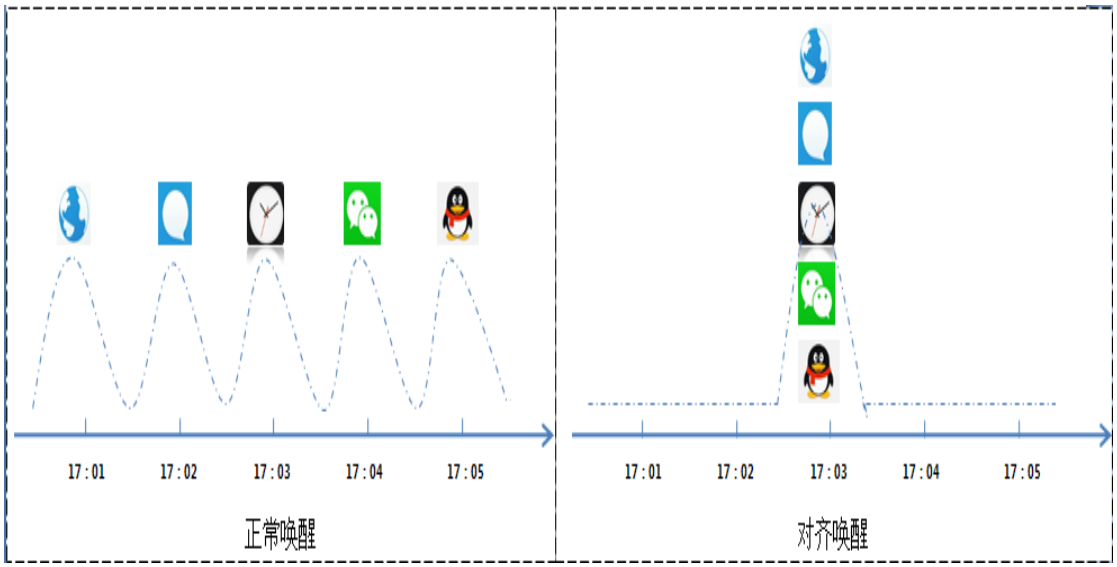

图 1-3 正常唤醒与对齐唤醒

Fig.1-3 Normal wake up and Aligned wake up

如图 1-3 所示，假如在 17：01 到 17：05 之间，有 5 个应用程序分别设定了定时器对系统进行唤醒，每个应用的唤醒时间恰好相隔 1 分钟，那么 5 分钟内就回对系统唤醒 5 次，频繁的唤醒系统会减少系统休眠时间，导致费电。而采用对齐唤醒机制，会把这 5 分钟时间内的 5 次唤醒合并为一次唤醒，从而减少了系统被唤醒的次数，增加系统待机的时间。

1.2.3 智能手机功耗研究现状

在国内工信部和运营商对智能手机的功耗都有严格的要求与标准，例如国内的入网测试和入围测试都要求手机的功耗必须达到规定标准，否则就不合格。除了芯片厂商和系统厂商在不断的优化手机功耗问题外，国内手机手机制造商与应用开发者也面临着用户对功耗和续航问题的投诉与反馈，也在急迫的寻找解决方案。目前在手机上已有不少省电管理应用，如电池医生、省电王、省电大师等，它们主要功能有电池状态显示、应用耗电排行、省电设置与提示、一键省电等，这些功能能够为用户提供一部分电池与耗电方面的信息，同时提示用户进行一些省电设置与操作，对减低手机整体功耗有一定的作用，但需要用户进行相关操作，仅为应用级的省电管理，对降低系统功耗的效果不明显。论文的省电管理方案是基于 Android 系统级的电源管理，在不影响用户正常使用与体验的前提下，通过系统服务在后台对唤醒锁进行管理，智能清理后台进程、应用联网管理并结合省电模式，在系统层进行省电优化与功耗管理，提高硬件的使用效率，并减少与避

免软件方面的一些不必要的耗电，达到减低手机整体功耗与增加手机续航时间的目的。

1.3 研究内容与意义

论文首先介绍了选题背景和研究意义，然后分别从硬件与软件方两个向分析了当前 Android 系统智能手机耗电情况与现状，并介绍了国内和国外手机硬件低功耗设计与硬件低功耗设计方面的相关技术。在硬件低功耗方面主要是芯片厂商对于芯片级的低功耗设计，软件低功耗方面主要是在系统和应用程序层面的监控和管理，使系统能够及时进入休眠，并对应用程序的耗电异常行为进行有效的管控，论文所作的主要工作内容如下：

1. 针对当前 Android 智能手机普遍存在续航不足的问题，进行了深入的分析，一方面由于智能手机电池容量有限，另一方面，由于手机功能越来越丰富，硬件不断的升级，安装的软件应用程序也越来越多，增加了相关的耗电从而导致续航不足的问题。论文结合了国内外省电管理和功耗管理方面的相关研究现状，介绍了硬件和软件方面的一些省电方案和策略。

2. 深入研究了 Linux 电源管理和 Android 电源管理的机制，分析了台式机和移动设备间存在的一些差异，总结了 Android 平台在电源管理实现机制上的变化与改进。

3. 详细介绍了高通 MSM8953 芯片处理器平台、Android 操作系统体系结构和手机功率系统。

4. 完成了基于 Android 系统的低功耗方案设计与实现。构建一套了低功耗管理机制，通过对唤醒锁进行有效管理、对后台异常占用 CPU 资源的应用程序及时清理，并结合应用程序联网管理和省电模式，从多个方面进行省电管理以减少系统的耗电量，达到提升手机续航能力的目标。

5. 搭建手机功耗的测试环境，对该基于 Android 系统的低功耗方案进行了测试与验证，并对当前的工作进行了总结和展望。

论文研究的意义在于如下几个方面：

1.Android 是目前最受用户欢迎的手机操作系统，全球市场占有率超过 75%，是目前用户量最多的操作系统。而 Android 手机续航能力不足问题一直为用户所诟病，降低功耗和提升手机续航能力也是手机厂商和芯片厂商的重点研究方向，

能为用户带来更便捷的移动互联网体验。

2. 论文结合了 Linux 和 Android 的电源管理机制，基于 MSM8953 芯片平台提出了一种低功耗方案的设计与实现，通过唤醒锁管理、后台异常清理，应用联网管理和省电模式多种机制相互结合，并在实际项目开发中应用了该方案，通过测试数据来体现该方案的实际应用效果，提升了手机产品的续航能力，并对 Android 平台开发人员有一定的借鉴价值。

1.4 论文组织结构

论文总共分为六个章节进行论述，首先对当前智能手机中普遍存在的续航问题进行分析，介绍了手机的硬件耗电与软件耗电，提出了降低手机功耗和提升续航能力的解决方案，研究了 Android 和 Linux 的电源管理机制，设计和实现了基于 Android 软件平台、高通 MSM8953 硬件平台的一套省电管理方案，并在实际项目上进行了测试与验证，文章组织结构编排如下：

第一章：论文针对智能手机续航能力不足的现状进行分析，总结了智能手机耗电方面存在的问题,结合硬件低功耗设计和软件低功耗设计的实际应用情况，说明低功耗设计在智能手机中的重要意义。

第二章：通过对标准的 Linux 电源管理机制和 Android 原生的电源管理机制进行研究，对比 Linux 与 Android 电源管理的差异，并详细说明了 Android 电源管理中的 Early Suspend、Late Resume 和锁机制。

第三章：系统低功耗的总体设计方案。基于高通 MSM8953 芯片硬件平台，Android6.0 软件平台，提出了一种省电管理的机制,将唤醒锁管理、后台异常清理，应用联网管理和省电模式相结合对手机进行监控管理，达到降低功耗，提升续航的目的。

第四章：关键模块的设计与实现。详细介绍了对唤醒锁进行有效管理、对后台异常占用 CPU 资源的应用程序及时清理、应用程序联网管理和省电模式的设计与实现方案。

第五章：介绍了 Power Monitor 的使用方法，搭建电流测试环境，使用 Power Monitor 对手机进行功耗和电流测试，给出测试数据。

第六章：根据实际应用经验总结了低功耗设计方面的工作成果,并提出下一步的研究方向。

第二章 电源管理

Android 手机操作系统是采用 Linux 的内核，Android 电源管理机制也是在基于 Linux 的电源管理基础上做了改进。本章详细介绍了 Linux 的电源管理，包括 APM 与 ACPI、Linux 设备电源管理模型和 Linux 休眠与唤醒机制，同时介绍了 Android 电源管理的层级结构、唤醒锁、Early Suspend 和 Late Resume 机制。

2.1 Linux 电源管理

2.1.1 APM 与 ACPI

由于传统的台式机和服务器上都有 BIOS（Basic Input Output System，基础输入输出系统）系统，通过 BIOS 系统完成硬件初始化、硬件检测和引导操作系统，所以最早的电源管理方法是基于 BIOS 系统的电源管理。APM（Advanced Power Management，高级电源管理）和 ACPI（Advanced Configuration and Power Interface，高级配置和电源接口）是电源管理中的 2 个工业标准，APM 是基于 BIOS 的电源管理，ACPI 是在 APM 基础上进行了改进并基于操作系统的电源管理。APM 与 ACPI 都在电源管理中占有重要的作用，下面会对 APM 与 ACPI 电源管理方法进行详细介绍。

1. APM

APM 是由微软和英特尔在 1992 年提出的一种基于 BIOS 的电源管理标准，最初在 IBM 兼容计算机上使用，能够对计算机处理器和设备电源进行管理，随后也用在 Window 操作系统和 Linux 内核中，1996 年发布了 APM 1.2 版本后没有再继续更新，微软从 Vista 开始也不在支持 APM 了[9]。

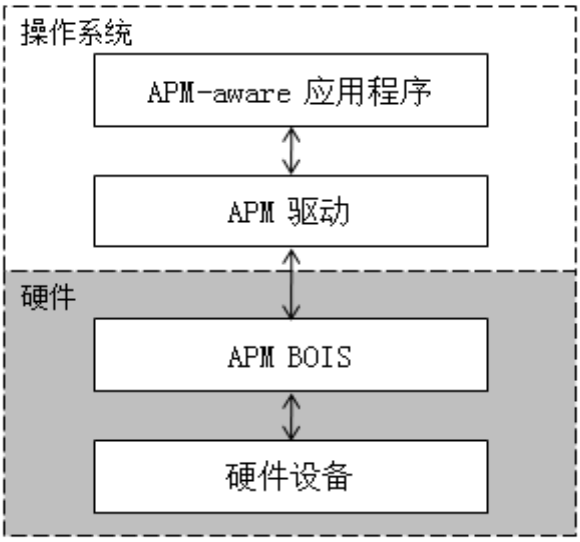

图 2-1 APM 框架图

Fig.2-1 APM Block Diagram

如图 2-1 所示，APM 层级结构从上往下分为 APM-aware 应用程序、APM 驱动、APM BIOS 和硬件设备 4 个层级。在硬件层 APM 中共定义了就绪、待机、挂起、休眠和关闭 5 种电源状态，而 APM BIOS 中定义了 Stanby、Suspend、Resume 和低电量通知等 12 种电源管理事件；在操作系统层 APM 驱动中共定义的 21 电源管理相关的函数，用于检测和修改硬件层的电源管理状态，APM-aware 应用程序通过接口调用 APM 驱动中的函数，触发 APM BIOS 中对应的电源管理事件，然后切换成对应的电源状态，硬件设备根据电源状态执行相应的电源管理。其中 APM 中定义的 5 种电源状态分别如下：

1. 就绪状态：计算机或设备在就绪状态处于完全加电的状态，就绪状态下系统随时可以使用，但不区分活动和空闲的情况

2. 待机状态：就绪状态下，计算机或设备在一段指定的时间间隔内没有活动且处理器处于空闲时会自动切换到待机状态。待机是一种降低功耗的中间状态，在待机状态计算机能够保存所有数据和操作参数，当发生硬件中断或访问受控设备会让计算机返回到就绪状态。

3. 挂起状态：通过 BIOS 或 BIOS 软件进入挂起状态，挂起状态比待机状态功耗更低，能够保存数据和操作参数，但在挂起状态计算机不能进行运算。

4. 休眠状态：休眠状态能够保存计算机或设备的完整状态并关闭电源，进

入休眠状态时，计算机会创建一个休眠文件保存自己的状态，从休眠状态恢复时，BIOS 执行正常的开机流程，然后读取休眠文件让计算机恢复到休眠前的最后状态，能够减少开机时间。

5. 关闭状态：计算机或设备在关闭状态时处于不通电且不活动的状态，关闭状态可能保存数据和参数，也用可能不保存。

作为最早的电源管理方法，APM 也存在一些缺点。首先由于 APM 是基于 BIOS 的，不同的计算机厂商 BIOS 方案不同，导致个 BIOS 上的 APM 实现也不相同，缺乏一致性。其次，修改电源管理的设置需要进入到 BIOS 系统，非常不方便。再次，BIOS 介于硬件和操作系统之间，用户只和操作系统打交道，BIOS 无法了解用户的操作和系统的状态，不能有效的进行电源管理。

2. ACPI

ACPI 是由微软、英特尔和东芝等联合在 1996 年提出的一种基于操作系统的配置和电源管理标准，提供 OSPM(Operating System-directed configuration and Power Managerment)的核心接口，定义了操作系统、BIOS 和硬件之间的工作接口。APM 是基于 BIOS 的电源管理，是由 BIOS 来进行电源管理的几乎所有控制，限制了操作系统的电源管理能力，同时由于不同硬件厂家都采用各自的 BIOS，也导致 BIOS 与硬件不易兼容[10]。这也是 ACPI 与 APM 的最大区，ACPI 是基于操作系统的电源管理，更加灵活，可扩展性也更好。ACPI 与 APM 也不能相互兼容，同一种设备只能采取其中的一种策略，ACPI 相比 APM 有更多的优点，微软从 Windows 98 操作系统中开始支持 ACPI，另外大部分的 Linux 家族操作系统 CentOS、ubuntu、OpenBSD 等都支持 ACPI。

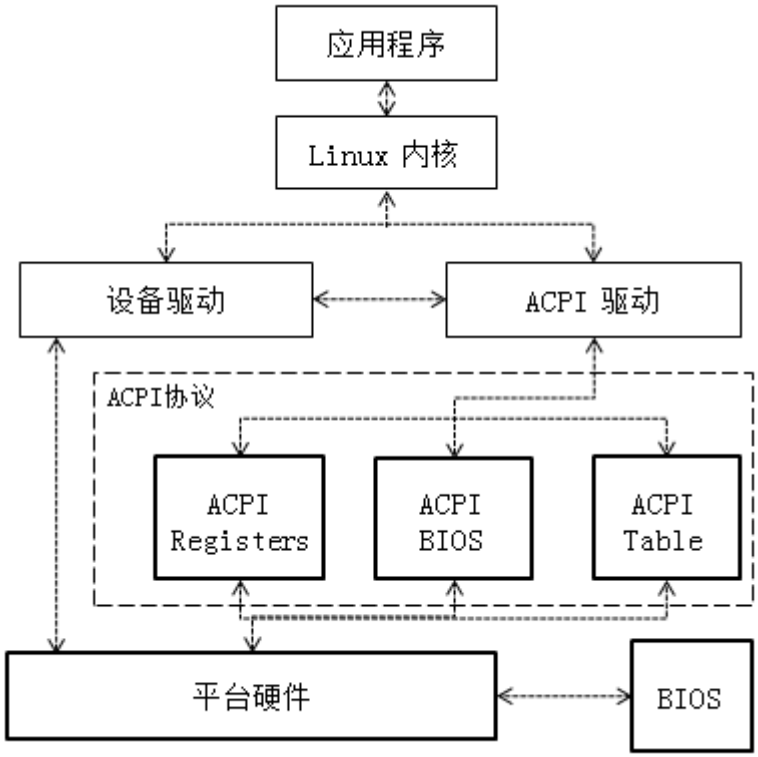

图 2-2 ACPI 框架图

Fig.2-2 ACPI Block Diagram

ACPI 系统结构图如图 2-2 所示：其中 ACPI 协议包含 ACPI Table、ACPI Registers 和 ACPI BIOS 等 3 个部分，它们各自的作用如下：ACPI Table：主要提供 AML（ACPI Source Language，ACPI 定义的硬件语言）用来表示与硬盘的接口；ACPI Registers：与 ACPI Table 一样，也是表示与硬盘接口的一部分；ACPI BIOS：主要提供启动、睡眠、唤醒和重启等操作的接口，又被称为 ACPI System Firmware。另外 ACPI 还定义了六种工作状态，S0、S1、S2、S3、S4 和 S5，各种工作状态的定义如下：

a) S0 运行状态（running）:系统正常工作状态，设备全部都上电，功耗最高的状态；

b) S1 挂起（standby 或 POS，Power on Suspend）： 通过 CPU 时钟控制器使 CPU 处于停止运行状态，但其他设备仍然正常工作。

c) S2 挂起（Suspend）：与 S1 相比停止了 CPU 的供电，只有主机通电；

d) S3 挂起到内存（Sleep 或 STR，suspend to RAM）：系统把当前状态保存到 RAM 中，只有 RAM 保持通电，系统处于底耗电状态。通过按任意键，系统从 R

AM 中读取保存信息并快速恢复到之前状态。

e) S4 挂起到硬盘（Hibernation 或 STD，suspend to disk)：系统把当前状态保存到硬盘中，然后关机。通过按电源键，系统重新启动并直接从硬盘读取数据，恢复之前的状态。

f) S5 关机（shutdown)：关闭所有设备电源。

2.1.2 设备电源管理模型

Linux 理论上可以支持几乎所有的、不同功能的硬件设备，这也是 Linux 的一个优点，由此也导致 Linux 内核中有大部分的代码都是设备驱动相关的，而且伴随着硬件不断的升级换代，设备驱动代码量还要继续增长。这么多代码进行管理必须要建立一套有效的规则，否则代码将会杂乱无章。Linux 的管理机制就是对内核代码进行抽象分层，把公共部分抽象出来，并建立统一的设备驱动模型，把非共性的东西使用接口去实现，不同的硬件只需要实现设备驱动模型的接口。通过设备驱动模型既能降低开发难度，也符合高内聚、低耦合的设计思想。电源管理作为 Linux 内核中的一个模块也不例外，也同样采用了设备驱动模型的设计，其中 Linux 设备电源管理有两种模型，系统睡眠模型和运行时电源管理模型，驱动设备通过其中任意一种模型都可以进入到低功耗状态：

1．系统睡眠模型（System Sleep model)：

系统睡眠模型就是当系统切换到低功耗状态的时，驱动设备也切换到低功耗状态，驱动设备作为系统的一部分随着系统级别的状态变化而变化。Linux 低功耗状态有 S3 挂起到内存（suspend to ram）和 S4 挂起到硬盘（suspend to disk)，系统如果切换到 S3 或 S4 状态，驱动设备、程序和总线等也通过执行相应的休眠或唤醒方法，切换设备状态与系统保持一致的状态。通过驱动程序管理硬件的休眠和唤醒，减少系统的开销，达到降低功耗的目的。

2．运行时电源管理模型（Runtime Power Management model)：

该模型与系统睡眠相对，设备状态不用与系统级别的状态保持一致，系统在运行状态设备也能独自进入低功耗状态，但是设备间不能进行单独的控制，例如：设备在系统运行状态下进入了休眠，当系统级别状态发生迁移时设备必须做出特殊处理。只有当所有子设备进入到休眠状态，父设备才能进入休眠，否则父设备不能休眠。在系统运行状态，如果有大部分的设备进入了休眠状态，这时即使系

统并没有进入低功耗状态，但是系统的耗电量也非常肖，效果也与系统的低功耗状态类似，固某些驱动程序可以利用运行时电源管理模型让系统进入类似低功耗的省电状态[11]。

2.1.3 Linux 休眠与唤醒

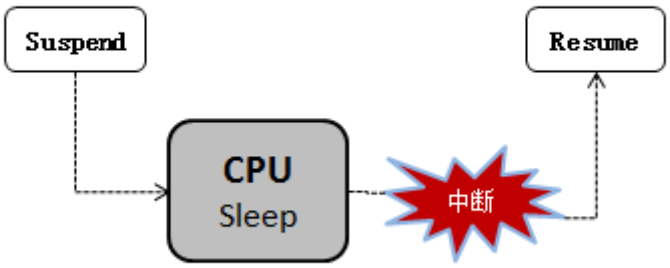

图 2-3 Linux 休眠与唤醒

Fig.2-3 Linux Suspend and Resume

如图 2-3 所示，Linux 休眠与唤醒是一对相反的方法，即 suspend 与 resume 方法，通过 suspend 方法休眠系统，通过 resume 唤醒系统。系统从运行状态进入到睡眠状态，会通知设备驱动调用 suspend 方法，使设备进入到 suspend 状态，具体情况各个系统有所区别，通常 suspend 状态是一种低功耗状态；通过中断使系统退出睡眠状态，同时会去调用 resume 方法唤醒相应的设备，使设备进入到唤醒状态[12]。suspend 与 resume 通常也是成对发生的， 由于以前的 Linux 设备一般都是台式机或服务器，由外部电源持续供电，固仅通过 suspend 与 resume 就能满足简单的电源管理需求。

2.2 Android 电源管理

由于Android系统使用的是Linux的内核，而电源管理模块是内核中的部分，一般由内核实现，固 Android 的电源管理实际上就是 Linux 电源管理为基础，并考虑到智能手机与台式机的一些差异，进行了改进与优化。主要是因为智能手机电池容量有限，比台式机更加注重低功耗与省电设计，同时手机的操作习惯与台式机操作习惯也有很大差异，Android 在 Linux 内核的电源管理的基础上主要增加了唤醒锁、Early Suspend 和 Late Resume 等机制。

2.2.1 Android 电源管理结构

在本节将详细介绍 Android 操作系统的电源管理层级结、各层的作用以及相互之间的关系。

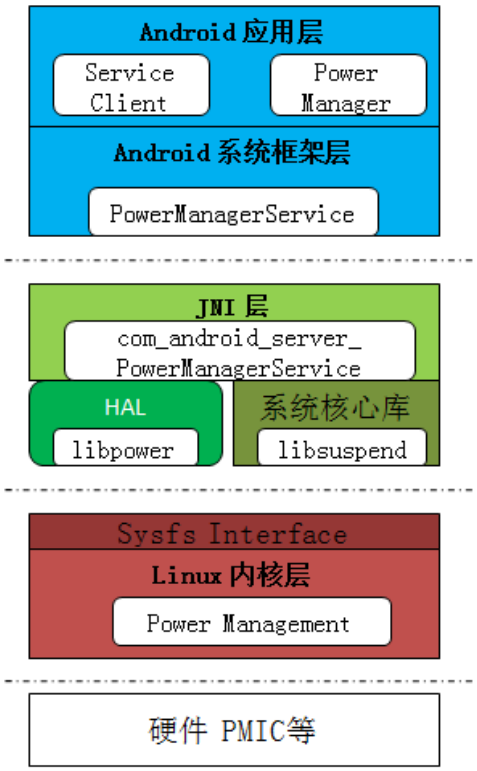

图 2-4 Android 电源管理框架图

Fig.2-4 Android Power Management Block Diagram

如图 2-4 所示，Android 电源管理结构中各层的作用分别如下：

a) Android 应用层：在应用程序主要是通过调用应用层本身的电源管理（Power Manager）和框架层的电源管理服务（Power Manager Service）来实现电源管理的相关功能。

b) Android 系统框架层：框架层主要是 Power Manager Service 这个远程服务，为应用层提供 api 接口，供应用程序使用。

c) JNI（Java Native Interface，java 本地接口）层：Java Native 指的是本地方法，当在方法中调用非 java 代码或用 java 直接调用其他语音（如 C/C++）操作计算机硬件时需要声明为 native 方法。JNI 层的作用是由框架层的电源管理服务中的本地方法调用的本地函数或本地函数调用电源管理服务方法的

本地空间，JNI 层电源管理服务的本地函数定义在 frameworks/base/services/core/jni/com_android_server_power_PowerManagerService.cpp 文件中。

d) HAL（Hardware Adaptation Layer）层：硬件适配层，也称为硬件抽象层。Android 增加的一层，主要作用是向下屏蔽 Linux 硬件驱动实现的细节，向上提供硬件访问的统一接口，HAL 层抽象从用户空间到内核空间的硬件设备访问，目的是让其可以适用于各种设备。

e) 系统核心库：Android 电源管理的 Early Suspend 和 Late Resume 操作也是通过 libsuspend 库去触发的。

f) Sysfs Interface：提供 sysfs 虚拟文件系统的相关接口，也是用户空间与内核空间通信的主要方法。

2.2.2 电源管理服务

电源管理服务（Power Manager Service）是 Android 系统的核心服务之一，主要负责上层的电源管理事件的操作，如控制系统待机、调节屏幕亮度、开关背光和键盘灯等。应用程序不能直接的调用 Power Manager Service 方法，而是通过 2 种方法间接的去调用 Power Manager Service 方法。一种是通过 Power Manager，Power Manager 为应用程序提供了公共接口去间接的调用 Power Manager Service 方法；另一种通过 Power Manager Service 的代理对象引用去调用[13]。

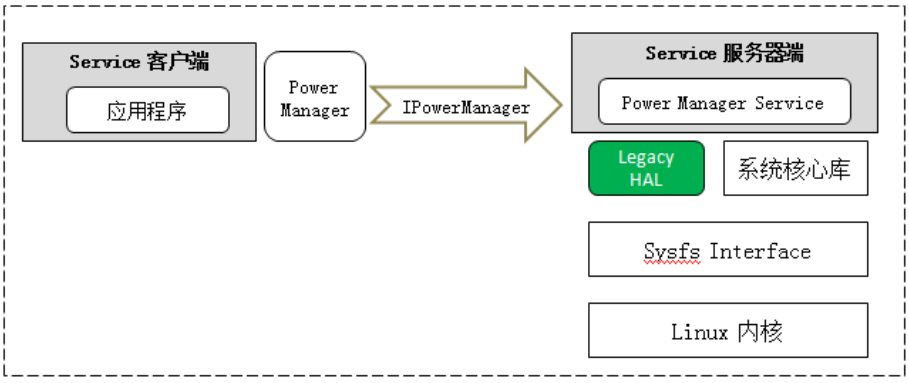

图 2-5 电源管理服务

Fig.2-5 Power Manager Service

如图 2-5 所示，应用程序与电源管理服务的关系有点类似客户端与服务器，应用程序相当与客户端，电源管理服务相当于服务端，客户端与服务端直接通过 RPC 接口 IPowerManager 进行通信，服务端控制电源管理相关操作，并将对应的信息传递至本地空间与内核空间。

2.3 Android 电源管理新特性

2.3.1 唤醒锁（wake lock）

唤醒锁（wake lock）是 Android 电源管理加入的新特性。wake lock 是一种锁机制，应用程序开发人员够通过申请 wake lock，来控制 CPU、LCD 和键盘灯的状态，通过阻止系统进入休眠来确保应用程序能够正确的执行，只有当所有的 wake lock 都被释放，系统经过判断不存在 wake lock 后才会进入到休眠状态。通过 wake lock，应用程序能够很容易的进行电源管理，有多种类型的 wake lock 标记，通过相应的 wake lock 标记类型来控制系统状态进行不同的切换[14]。wake lock 不同的标记它们实现功能均不同，主要有如下几种类型。

a) **FULL_WAKE_LOCK**：保持 CPU 运行；屏幕和键盘灯保持最大亮度状态；

b) **SCREEN_DIM_WAKE_LOCK**：保持 CPU 运行；只有屏幕保持显示状态，但有可能是灰的；键盘灯为关闭状态；

c) **SCREEN_BRIGHT_WAKE_LOCK**：保持 CPU 运行；只有屏幕保持最大亮度状态；键盘灯为关闭状态；

d) **PARTIAL_WAKE_LOCK**：保持 CPU 运行；但屏幕和键盘灯可能是关闭的；

e) **ACQUIRE_CAUSE_WAKEUP**：WAKE LOCK 并不会点亮屏幕，只是在亮屏状态下保持屏幕亮屏状态，通过与 ACQUIRE_CAUSE_WAKEUP 一起使用，能够主动点亮屏幕。

f) **ON_AFTER_RELEASE**：在 WAKE LOCK 被释放时重置计时器，作用是延长亮屏，减少 WAKE LOCK 切换时的闪烁。

2.3.2 Early Suspend 和 Late Resume

与 Linux 电源管理相比，除了增加了唤醒锁机制， Android 操作系统还增加了 Early Suspend 和 Late Resume 两种中间状态，以便更合理的对移动设备进行电源管理，降低耗电量[15]。

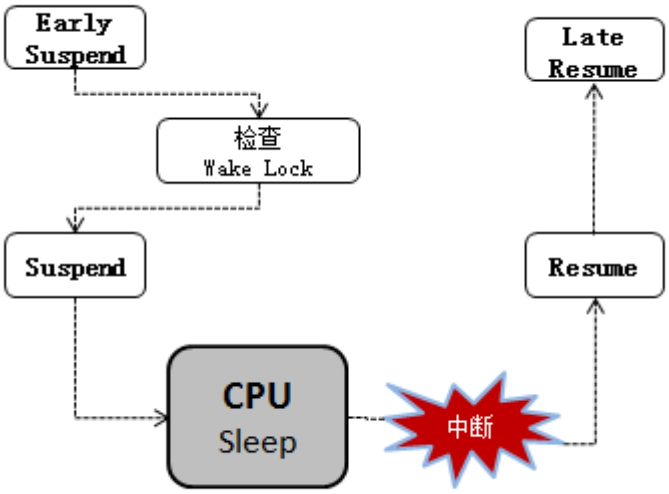

图 2-6 Android Early Suspend 与 Late Resume

Fig.2-6 Android Early Suspend and Late Resume

Early Suspend：顾名思义，Early Suspend 是增加在 suspend 之前的一个状态，与 linux 系统不同，Android 不是从运行（running）状态直接进入到挂起（Suspend）状态。当智能手机熄灭显示屏的时候，与屏幕相关的一些设备也应该被关掉，如屏幕背光、触摸屏和一些传感器等等。但屏幕熄灭并不代表系统就可以进入休眠状态，持有 wake lock 的应用程序还在后台运行任务，如电话、音乐播放器和网络下载等，直到所有的 wake lock 都被释放，系统才能真正进入到休眠状态。所以 Android 中加入 Early Suspend 机制，在屏幕熄灭时，把 Early Suspend 属性的设备都关掉，可以节省很大一部分耗电。

Late Resume：Late resume 是和 Early Suspend 配套的机制，增加在 resume 后面，就是当唤醒源对 CPU 进行唤醒，首先会调用 Resume 执行标准的 linux 内核唤醒流程，但不会像 Linux 一样激活所有中断，只会激活其中指定的中断，已节省耗电量。通过对唤醒源进行判断，假如是要求对屏幕进行唤醒的操作，如来电话和按电源键，那么就会远程调用 PowerManagerService 修改/sys/power/state 中状态，通过执行 late resume 打开在 Early Suspend 中关闭的设备，如点亮屏幕、背光和触摸屏等；否则，不会执行 Late resume，在 linux 内核唤醒流程执行完后，系统再次进入休眠状态。

2.4 本章小结

本章主要介绍了Linux电源管理和Android电源管理的相关技术背景和层级结构等，介绍了Android平台上的新特性，唤醒锁、和Late resume是和Early Suspend的实现机制，为论文的系统低功耗设计与实现提供指导。

第三章 总体设计与平台架构

系统低功耗的总体设计方案必须基于软硬件平台架构，否则就可能出现不兼容情况或导致系统性问题，从而达不到预期的省电效果。论文所采用的高通 MSM 8953 硬件芯片平台是高通公司今年推出的全新手机处理器平台，14nm 制程，8 核 A53 架构，主频 2Ghz，是高通骁龙 600 系列的主打产品，硬件性能和功耗较上一代处理器平台有显著提升。正是由于 MSM8953 是新的芯片平台，故没有类似的低功耗设计方案提供参考，在前期需要详细的了解平台特性并进行大量的预研工作。软件平台采用的是谷歌公司 2015 年发布的 Android N 版本，与上个版本 Android L 相比有较大的改动，系统低功耗设计方案也要适应 Android N 的新特性。本章将详细介绍系统低功耗的总体设计方案与软硬件平台架构。

3.1 总体设计

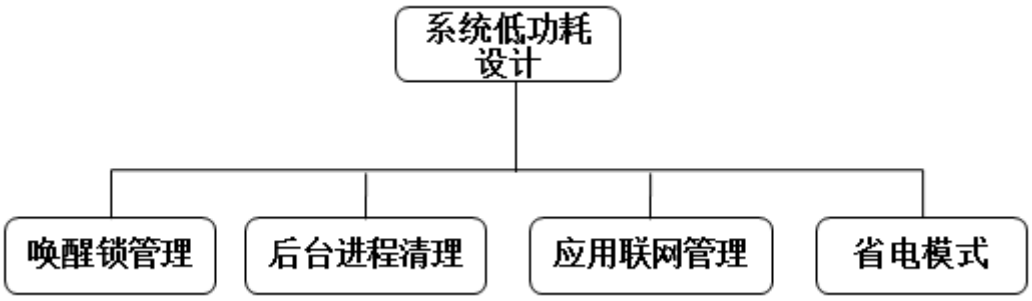

图 3-1 总体框架图

Fig.3-1 System Block Diagram of Architecture

如图 3-1 所示，系统低功耗设计分为唤醒锁管理、后台清理、联网管理和省电模式四个关键模块，各模块的功能和作用分别如下：

唤醒锁管理是通过对应用申请的 WakeLock 进行监控，不允许第三方应用程序长时间的占用唤醒锁，及时的释放非特殊情况下应用申请的 WakeLock，解决应用程序导致系统休眠异常的问题，使系统在不影响系统正常运行的情况能够及时的进入休眠状态，达到降低功耗的目的。

后台清理是指清理挂在系统后台并异常占用了 CPU 资源的应用程序和进程，导致 CPU 在未执行任务的情况下还运行在较高的频率，增加了系统的耗电量。后台清理分为亮屏状态下后台清理和灭屏状态后台清理两种情况。亮屏状态下的后

台清理主要通过后台异常清理功能实现，监控后台应用的 CPU 占用率并设置一个阀值，对于后台 CPU 占用率长时间超出阀值范围的应用，认定为在后台异常运行，将异常应用筛选出来并进行清理。灭屏状态下的后台清理以灭屏后系统能够正常休眠为目标， 在接收到灭屏广播后对后台进程进行检测并清理异常的后台程序和进程，可以有效的解决灭屏状态下应用后台耗电异常问题。后台清理功能致力于解决系统应用程序存在的后台耗电行为，通过多种手段来实现，同时也充分考虑了该功能自身运行时对系统休眠和用户正常使用的影响，加入拉延时执行的机制，并针对可能出现的问题做了适当的处理。

联网管理主要是对 IPTable 的设置实现对应用程序的网络管理，在收到灭屏广播后保存当前应用程序的网络连接状态，并进行恰当的处理禁止非白名单应用联网，应用在亮屏状态下恢复保存的应用程序网络连接状态。通过联网管理减少应用程序在灭屏状态下联网和唤醒系统的行为，从而减低系统功耗。

省电模式为用户提供一种极度省电的情景模式，类似飞行模式，推荐在手机低电量的情况下使用。当手机低电量时进入省电模式后系统会关闭大部分的功能与服务，仅保留电话、短信、联系人和闹钟四个基本功能，以达到最大化省电的目的，在低电量的情况下尽量延长手机续航时间，避免手机电量耗尽关机。

3.2 硬件平台

3.2.1 MSM8953 芯片平台概述

AP（Application Processor，应用处理器）和 MP（Modem Processor，调制解调器）是手机芯片平台上最重要的两个组成部分。AP 应用处理器负责运算和数据处理，做为主控 CPU,运行 Android 操作系统和响应应用程序请求，完成手机上的大部份功能，而 MP 调制解调器主要任务是处理语音通话业务和数据上网业务，AP 和 MP 构成了手机双处理器架构。以前 AP 和 MP 是分别独立的两块芯片，随着技术的不断发展，使得手机芯片的集成度越来越高，目前的趋势是单芯片化，将 AP 应用处理器、MP 调制解调器以及一些外围器件都集成在一块芯片上，合二为一，能够有效减小芯片的总体积，有利于降低整体的硬件功耗。

高通 MSM8953 芯片，外部型号名为骁龙 625，是骁龙 600 系列中第一款采用 14nm 制程的芯片，CPU 采用的是八核 Cortex-A53 架构[16]，分为 4 个大核和 4 个小核，大核的二级缓存为 1MB，小核的二级缓存为 512KB，每个核的最大频率均

为 2GHz。

MSM8953 芯片结构如图 3-2 所示，橙色的两个模块是手机芯片中最重要的 AP（应用处理器）和 MP（调制解调器），其他的主要模块有 CAMSS(照相机)、VIDEO(视屏)、MDSS(显示屏)、Graphics（GPU，图像处理器)、LPASS(功耗管理)、WCNSS(蓝牙/wifi)和 Memory（内存)。

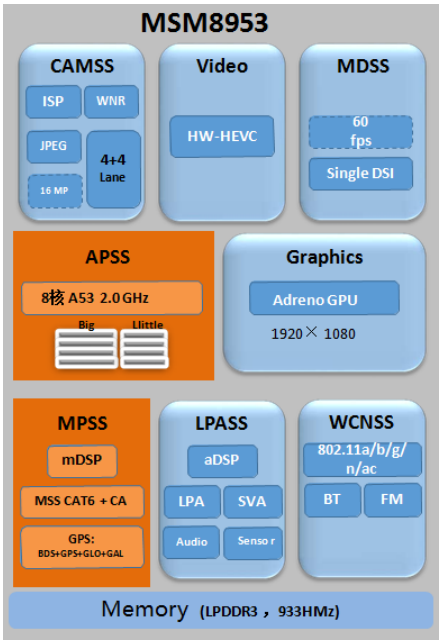

图 3-2 MSM8953 结构图

Fig.3-2 MSM8953 Structure Diagram

3.2.2 硬件平台架构

目前在智能手机芯片市场竞争也非常激烈，主流的芯片厂商有高通、联发科、三星猎户座和华为海思等。其中高通芯片由于性能与基带功能强大，功耗控制也较好，所占市场份额也最大。本文采用的硬件芯片平台为高通骁龙 625（Snapdragon 625）八核处理器，内部型号名 MSM8953，属于高通骁龙 600 系列，是高通公司今年推出的中端主打芯片产品。采用 14nm 制程，主频 2Ghz，在性能及功耗方面都有很好表现，多家手机厂商都在研发搭载 MSM8953 芯片处理器的手机产品，并将在今年陆续推出。与高通上一代处理器比较，MSM8953 在性能及功耗方面表现更好。

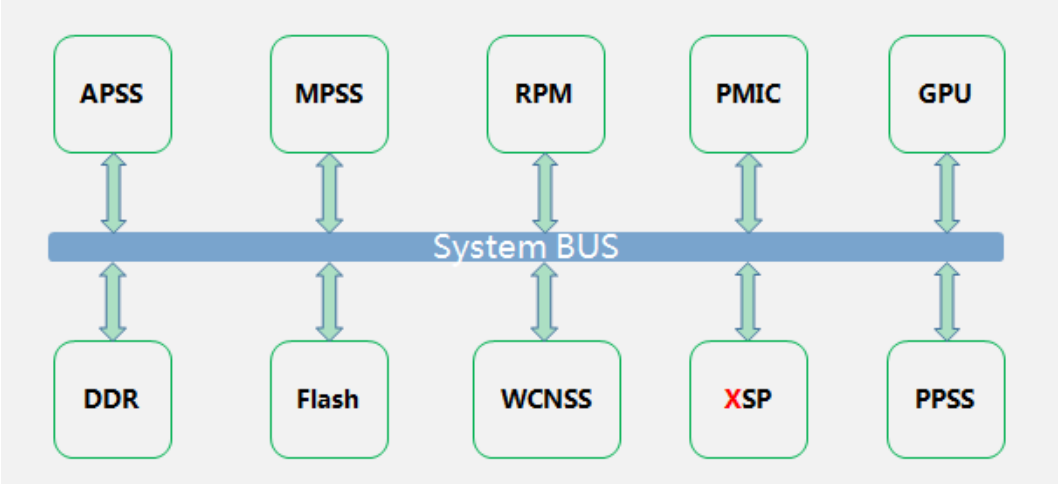

图 3-3 硬件框架图

Fig. 3-3 Hardware Block Diagram

如图 3-3 所示，MSM8953 芯片硬件框架图，手机芯片内部分为多个独立的模块，各个模块也是一个独立的子系统，各子系统之间通过系统总线进行通信，其中各子系统的功能与作用分别如下：

1. APSS（Application Processor Subsystem）：应用处理器子系统，简称 AP，AP 处理机是基于 ARM 架构，通常由一个或多个核心组成[16]。MSM8953 的 AP 采用的 8 核 Cortex-A53 架构，主要作用是运行 Android 操作系统及应用程序，是手机系统不可或缺的部分。

2. MPSS（Modem Processor Subsystem）：调制解调处理器子系统，简称 MP，也叫做基带芯片[25]，通常由一个核心组成，采用高通自主的 ARM 架构，主要负责语音通话以及数据上网业务等相关的功能，MP 与 AP 也是手机芯片中最重要的两个模块。

3. RPM（Resource Power Manager）：资源电源管理，也是基于 ARM 的处理器架构，负责管理及分配系统电源和时钟。

4. PMIC（Power Management IC），电源管理集成电路，集成各种电源（DC-DC/LDO），满足其它各个部件的电源需求。

5. GPU（Graphics Processor Unit），图形图像专用处理器，用于图形图像的绘制和处理。

6. DDR SDRAM（Double Date Rate Synchronous Dynamic Random Access Memory），同步动态随机存取存储器，简称 DDR 或内存，也是计算机的重要组成部分，主要负责系统运行过程中的数据存储，特性是存取速度快，但掉电后数据

不能保持。

7. Flash（Flash Memory），也称闪存，主要负责软件代码及数据的持久存储，与 DDR 相对，特性是存取速度慢，但掉电后数据能够继续保持。

8. WCNSS（Wireless Connectivity Subsystem），处理蓝牙及 Wifi 相关协议和数据等。

9. XPU（专用处理单元），其特点是功能职责单一。如 ISP 负责处理图像，A DSP 负责处理音频数据或者 sensor 数据[20]。

10. PPSS（Peripheral Subsystem），外围设备子系统，负责给各种外设提供接口，如 GPIO，I2C，SPI 等。

3.2.3 Big.Little 处理器架构

处理器（CPU）、存储器（DDR）与外围设备（I/O）是计算机的三大核心组成部分，处理器包含运算核心和控制单元，主要负责执行指令和处理数据，处理器的运算速度与能力也在很大程度上决定了计算机性能的好坏。处理器内核设计厂商主要有 ARM、Inter 和 MIPS，Android 系统分别都支持这 3 种架构的处理器架构，但 Inter 主要在台式机和服务器上非常普及，MIPS 主要应用在嵌入式领域，而 Android 平台主最受欢迎的是 ARM 架构处理器，ARM 在 Android 的市场占有率超过 95%。

提升处理器性能的主要方法是更新架构和升级制程工艺。如 Inter 在 2007 年提出“Tick-Tock”的处理器升级模式[17]，Tick、Tock 分别代表制程与架构，“Tick-Tock”表示按年份分别对处理器的制程和架构轮流进行升级，即每一年推出新制程的处理器产品，第二年再推出新架构的处理器产品，每两年为一个硬件升级周期，不断的对处理器产品进行升级。移动应用处理器（AP）的发展趋势也是在不断的更新架构和提升制程，ARM 架构的发展从 Cortex A8，到 Cortex A 9,再到 Cortex A15；制程工艺也在不断提升，从 28nm 提升到 14nm；时钟频率也从 1GHz 提升到了 1.5GHz，再到 2.5GHz；应用处理器的性能能也在不断的提高，已经能够达到并超越了一些台式机的处理器性能，但是同样也碰到了台式机 CPU 发热与耗电的问题， 时钟频率难以继续提升，所以芯片厂商开始从多核中寻找答案。

多核技术是指在一个处理器中集成两个或两个以上的处理器内核，支持多个

核处理指令，有效分散操作处理。多核的发展是从 2 核到 4 核，再到 8 核，目前大多旗舰产品采用的都是 8 核处理器。内核越多，理论上耗电亮也越大，但是多核可以通过统一的管理，不用在运行中同时启动所有的核，一般只开启 1~2 个核，在需要时再开启其他核，耗电量不如单核那么高，还能实现比单核更高的性能。

处理器性能越高，耗电量也越大，性能与耗电量间存在一定的比例关系，很难实现提升性能却不增加耗电量。因此 ARM 采用了 Big.Little 架构[18]，即将多核分为 Big 内核与 Little 内核。Big 内核提供高性能，负责处理视屏、拍照和游戏等需要较高 CPU 负载的操作，而 Little 提供低耗电，在处理通话、听音乐等不需较高负载的操作时尽量使用 Little 内核，通过合理的将任务分别分配给 Big 与 Little 处理，达到技能提供高性能又尽量节省了耗电量的目的。

3.3 软件平台

3.3.1 Android 操作系统概述

2003 年，安迪鲁宾创建了 Android 公司，开始开发手机操作系统[19]。2005 年 7 月谷歌收购了 Android 公司，并在 2007 年成立了一个全球性的联盟组织——开放手机联盟 OHA（open handset alliance），最初成员包含手机制造商、手机芯片厂商和移动运营商共 84 家。2008 年 9 月，谷歌发布了 Android1.0 版本，随后，HTC 生产了全球第一款搭载 Android 操作系统的智能手机 G1。2010 年 10 月 Android 达到了首个里程碑，谷歌宣布通过官方认证的 Android 应用程序已超过 10 万个。2011 年， 在全球使用携带 Android 操作系统的设备用户总数已超过 1 亿，日均新增长的 Android 设备用户达到 55 万。当年 8 月份， Android 操作系统全球市场份额跃居全球第一，首次超过诺基亚的塞班操作系统，成为最受欢迎的操作系统。2013 年 9 月，Android 操作系统已推出来 5 年了，全球采用 Android 系统的设备总熟练已超过 10 亿台，同时采用 Android 平台的智能手机在全球市场份额达到了 78.1%。

3.3.2 Android 软件平台架构

如图 3-4 所示，Android 操作系统采用分层的设计思想，软件系统架构可以分为五层[21]，由下而上分别为 Linux 内核层（Linux Kernel）、硬件抽象层（HAL，Hardware Abstraction Layer）、系统运行库层（Libraries 和 Android Runtime）、应用框架层（Application Framework）和应用程序层（Applications）。

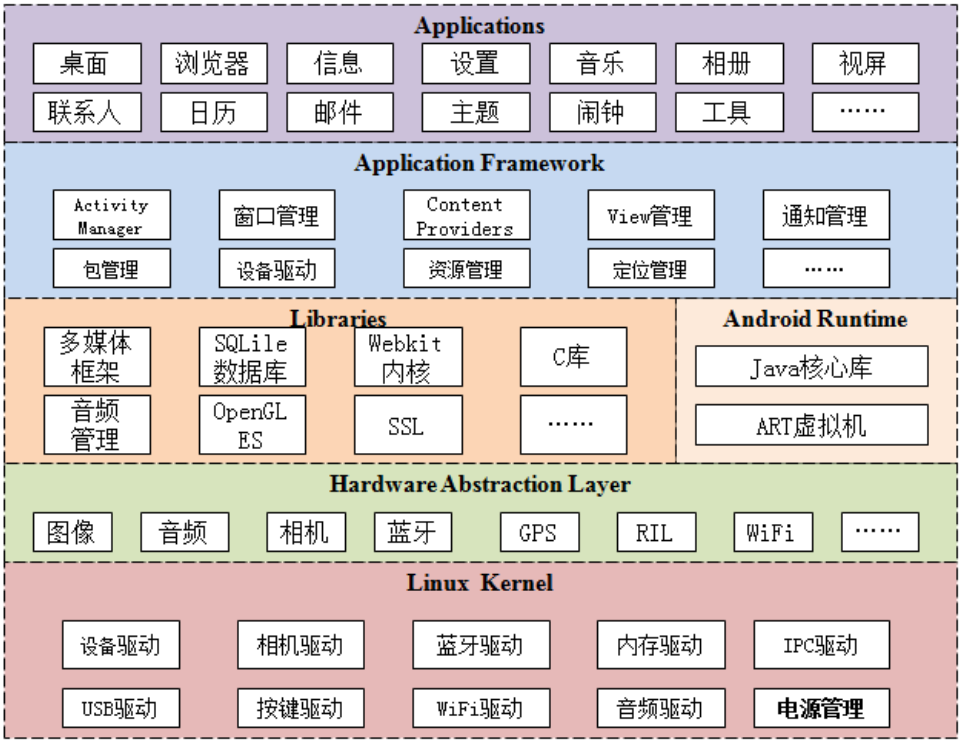

图 3-3 Android 系统架构

Fig.3-3 Android System Architecture

Android 操作系统中各层的作用分别如下：

1. **Linux Kernel**：Linux 内核层，为 Android 提供核心系统服务与驱动程序，如设备驱动、相机驱动、内存管理、进程管理和电源管理等。

2. **Hardware Abstraction Layer(HAL)**：也称硬件抽象层，介于内核与操作系统之间。向下屏蔽 Linux 硬件驱动实现的细节，向上提供硬件访问的统一接口；面向对象的设计方法重点就在抽象，目的是让其可以适用于各种设备。

3. **Libraries and Android Runtime**：系统运行库层，包括程序库和 Android 运行库两部分。程序库中包含一些 C/C++库，提供给 Android 内部的组件使用，为应用框架与程序提供服务，有系统 C 库、媒体库、浏览器内核和 SQLite 数据库等。Android Runtime 主要包含 java 的核心库和虚拟机，java 核心库提供 java 语音的基础功能，如数据结构、工具、数据库和网络等；art 虚拟机是 Android4.4 版本发布的，用来代替之前的 Dalvik 虚拟机，用于提升应用程序启动速度和性能。

4. **Application Framework**：应用框架层，主要是提供 API 和开放接口，供应用程序层和应用程序开发人员调用系统框架中的函数和功能。

5. **Applications**：应用程序层，即通过在操作系统中安装各种应用程序，

来满足用户的各种需求。

3.3.3 Android 应用四大组件

Android 操作系统中的应用程序都是由组件组成，而组成应用程序的组件主要有四种，也称为四大组件，它们分别是 Activity、Service、Content provider 和 Broadcast receiver。这四大组件在 Android 操作系统中有非常重要的作用。

Activity：主要用是提供一个界面并与用户进行交互，是应用程序的基本功能单元，扮演前台界面的角色。但实际上 Activity 本身并没有界面，而是通过 setContentView(View)方法将 UI 放到 activity 创建的窗口上。一个应用程序通常包含多个 Activity，每个 Activity 都可以互相调用其他 Activity，其中必须有一个 Activity 被指定为主 Activity，应用程序启动时会首先显示主 Activity。

Service：即 Android 服务，没有界面，无法直接与用户进行交互，通过用户或者其他应用程序启动，运行在后台。优先级比前台应用低，但比后台其他应有优先级高。比如在手机锁屏状态下听音乐，就是在后台运行音乐播放器的服务。

Content provider：内容提供者主要作用是给其他应用程序共享数据，通过内容提供者把应用的数据共享给其他应用程序访问。分为系统的与自定义的，例如联系人数据与图片数据都是系统的。应用程序通过实现 ContentProvider 能提供数据给其他应用访问，通过实现 ContentResolver 来操作其他应用数据或应用程序在内部共享数据。

Broadcast receiver：广播接收者是一种消息型组件，主要作用是在不同的组件或不应用之间发送消息，是一种应用程序间传输信息的机制，分为静态广播和动态广播两种。

3.4 手机功率系统

3.4.1 手机供电系统

要进行手机低功耗的设计，就必须了解手机系统功率的情况，上面分别介绍了智能手机硬件平台与软件平台的架构，接下来将详细介绍手机内部是如何进行供电以及手机功率概述。

图 3-4 手机供电系统

Fig.3-4 Mobile Power Supply System

手机供电系统如图 3-4 所示，通常由充电器、充电芯片（ChargerIC）、电池、电源管理芯片（PMIC）和子系统模块等五个部分组成。充电器的作用是连接电源和手机，给手机电池充电。ChargerIC 是充电控制芯片，主要作用是保持充电过程中的电流和电压稳定，确保充电安全。充电过程通常是恒流恒压充电，分为恒流充电和恒压充电两个阶段，第一个阶段是恒流充电阶段，以恒定电流电流进行充电，此时电压会不断的上升，充电速度比较快，当电压达到预定值时转为第二阶段恒压充电，此时电池充电电量约为 70%，充电电源保持预定值不变，避免电压继续上升导致发热及安全问题；恒压充电阶段，充电电压保持不变，随着充电的电量不断增多，电流会逐渐降低，当电池电量充满时，充电电流也趋于 0，自动停止充电。电池的作用是存储电量，因为手机不带外接电源，电量都存储在电池里面，通过电池给 PMIC 供电。PMIC 的作用是对硬件进行电源管理，再将电量合理的分配给手机中的各子系统模块，主要的子系统模块有应用处理器子系统、modem 子系统、照相机子系统、和音频/视频子系统等等。

3.4.2 系统功率树状图

PMIC（Power Management IC，电源管理芯片），主要作用是为手机中各个子系统提供电源和时钟[22]，它内部分又为电源管理核心（PM core IC）和电源管理接口设备（PM interface IC）两个部分。电源管理核心负责提供输出电源管理功能和大部分的系统时钟，为射频、天线、音频、WIFI/蓝牙、处理器和外围设备等提供电源和时钟，电源管理接口设备负责一些输入电源管理功能，有充电芯片的部分功能，同时提供如显示屏、触屏和背光等模块的用户接口[23]。、

系统功率树状图如图 3-5 所示，显示了电流的流动方向：电池→PMIC 调节器→各子系统。PMIC 是电池和子系统之间的主要接口，所有用于电力系统部件的调节器（SMPS 和 LDO）都在 PMIC 内，这些调节器为子系统提供并维持所需电压和电流。

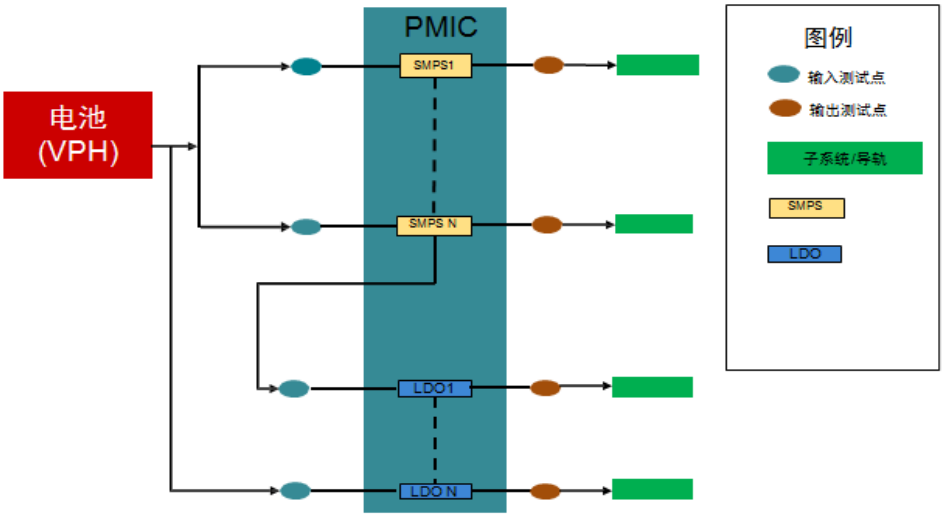

图 3-5 系统功率树状图

Fig.3-5 Power Tree Diagram of System

开关模式电源(SMPS)是一种将开关式稳压器合并，从而在电压差较高的情况下提高效率的电子式电源，SMPS 有以下 2 个优点：1.因为开关晶体管在充当开关时耗散了较低的功率，所以效率更高；2.因为开关模式电源的效率高，所以热发生较低，所以功耗更低。

低压差调节器(LDO)是一种 DC 线性调节器，可以操作较小的输入－输出差分电压。LDO 适合在调节器的电压差较低的情况下使用，较高的电压差意味着较多的损耗和散热。

功率树状图也显示了功率细分通道与输入/输出测试点，通过测量测试点的电压值用于调试功率问题以发现导致功率较大的子系统。

3.4.3 CPU 低功率模式

CPU 处理器共有五种低功率模式，处理器通过进入其低功率模式以实现省电目的。根据 CPU 空闲的持续时间，即处理器运行下一次计划任务前可用的时间量决定处理器要进入哪种合适的低功率模式，CPU 低功率模式共有以下几种：

软件等待中断(SWFI)：也称作软件时钟门控，是最低水平的节能模式，仅涉及对处理器的时钟供给进行门控，不需要 RPM 的介入便可终止处理器。

保持：时钟被门控并设置为较低的保持电压，在该模式中处理器的时钟已被门控并且处理器的电源电压降到保持电压等级，有助于降低功率泄漏。

快速电力休眠：也称作所有内核的时钟门控，该模式提供了软件执行透明化形式的 CPU 核心逻辑的电力休眠和恢复机制。快速电力休眠代替了传统的时钟门控低功率模式，使得功率泄漏变得更低。

独立式电力休眠：也称作没有 RPM 通知的时钟和电力休眠，处理器的时钟和电源在没有 RPM 的援助下被切断。因此，处理器可以在不通知 RPM 的情况下进入电力休眠，也可以在不涉及 RPM 的情况下唤醒。

电力休眠：电力休眠是指对处理器的电源进行门控，该模式需要通知 RPM 并由 RPM 援助对资源进行电力休眠，仅当 RPM 将处理器从电力休眠中提出时处理器才能唤醒。

3.4.4 系统低功率模式

不仅处理器有低功率模式，系统也有低功率模式。只有当所有处理器均已进入电力休眠的低功率模式，并且通过投票通知 RPM 系统进入睡眠，则系统才可以进入低功率模式。根据处理器投票以及自当前时间到下一计划唤醒的可用时间，RPM 将决定进入系统低功率模式以实现省电的目的，系统低功率系统睡眠模式和深度睡眠两种模式。

系统睡眠模式（XO 关闭）：XO 是向系统不同功能块供应时钟的来源，系统睡眠模式会关闭 XO，基本上会禁用对系统的时钟供给以实现省电的目的。当进入系统睡眠模式，MPM（Modem Power Manager，Modem 电源管理）会接管整个系统，主要的功能是进行中断管理并负责唤醒系统。执行 XO 关闭时，CXO 始终开启，只有其缓冲器关闭，CXO 时钟生成在 PMIC 上仍保持开启并用于生成 32kHz 睡眠时钟，这也是 MPM 等始终开启的部件所必须的时钟来源。

深度睡眠模式（VDD 最小化）：在系统睡眠模式下继续降低电压仍可以节省一些漏电流，因此也更加节省功率，深度睡眠模式是系统所能实现的最深层的低功率模式。当进入深度睡眠模式时， VDD 被最小化，虽然系统仍然保持着所有硬件和软件状态，但芯片在该状态下不会运作（除了检测 VDD 最小化的唤醒中断/计时器超时），这时内存并不能进入电力休眠，因此使用了较低的电压以维持内存中的内容等。在进入深度睡眠模式之前必须保证系统已进入睡眠模式，即只有进入系统睡眠模式之后，才有可能进入系统深度睡眠模式。

3.5 本章小结

本章不但介绍系统低功耗的总体设计方案、MSM8953 硬件平台以及 Android 软件平台，除了对芯片架构、处理器架构、操作系统架构和应用程序组件做了详细的介绍外，还详细介绍了手机功率系统，包括手机供电系统、系统功率树状图、处理器低功率模式和系统低功率模式四部分内容。本章也是 Android 系统低功耗设计的基础，通过对系统有效的监控和管理，使处理器和系统及时的进入到低功率模式，达到降低功耗和省电的目的。

第四章 关键模块设计与实现

上一章介绍了系统低功耗设计的总体方案、软硬件平台和手机功率系统，说明手机内部是如何进行供电，本章将说明如何进行省电以及关键功能模块的设计与实现。

手机低功耗设计实际上也是对 CPU 性能调优，通过 CPU 性能调优策略降低手机功耗。首先 CPU 处理器是手机的大脑，CPU 控制着整个操作系统的运行状态以及其他部件的状态，故 CPU 实际上也控制着整个手机的系统功耗，同时 CPU 自身也是手机上的一大耗电单元，CPU 运行时耗电量占手机整体功耗的 30%以上。CPU 的功率主要通过 CPU 运行频率控制，CPU 内部有频率调节器，调节器能够决定当前 CPU 频率，根据 CPU 当前的实际负载合理的调节 CPU 频率，在满足正常使用的前提下尽量降低 CPU 频率，从而降低 CPU 的功率达到省电的目的。CPU 支持各式各样的调节器，如交互式、请求式和性能调节器，主要的调配模式有以下几种：

1. 请求式调节器：该调节器根据电流利用率设置 CPU 频率。

2. 性能式调节器：该调节器将 CPU 的频率设置为界定的最低和最高频率范围内的最高值。

3. 交互式调节器：该调节器与请求式调节器相似，用于对延迟比较敏感的工作负载，根据 CPU 占用率从 DCVS 表格中动态选择合适的频率。请求式和交互式调节器之间的差别只是用于确定 CPU 频率的参数不同。

Android 系统的一个优点是允许应用在后台运行，但是如果缺少有效的系统监控和管理，一方面会导致应用不合理的占用 CPU 资源，造成资源浪费；另一方面应用在后台运行，有可能导致系统一直不能正确的进入睡眠状态，极大的增加系统耗电。为了避免上述问题，系统低功耗设计方案通过唤醒锁管理、后台清理、联网管理和省电模式四个功能模块来实现对系统有效的管理和监控，及时清理不规范的应用及进程，一方面确保 CPU 运行在正常的频率，另一方面确保及时释放系统资源，使系统在未使用的情况下及时进入睡眠模式，达到降低功耗和省电的目的。

系统低功耗设计中的另一个难点是如何在不影响用户正常使用和体验的前

提下达到省电的功能。唤醒锁管理需要准确的区分那些申请是由用户发起的、那些申请是应用程序自己发起的,管理行为不能影响到用户的正常使用；后台清理也需要甄别出在后台异常运行的应用程序,如果清理了正常的后台程序会导致功能异常，例如音乐和下载都是在后台运行；联网管理功能也是一样，只在灭屏状态禁止一部分应用程序联网，但是在灭屏状态下用户也可能在打电话、听音乐和下载，需要针对这些应用场景做相应的处理；省电模式推荐在低电量的情况下使用，延长续航时长，避免手机因电量耗尽而关机。接下来将介绍各功能模块的详细设计与实现。

4.1 唤醒锁管理

4.1.1 唤醒锁概述

Android 的唤醒锁机制能够阻止系统进入休眠，系统没有对应用程序申请唤醒锁进行严格的管理，而用户使用过程中也不会知道那些应用程序申请了唤醒锁，甚至不知道系统是否正常进入休眠，固这一块缺乏管控。由于 Android 又是一个开发的操作系统，任何人都可以自己开发应用程序，所以存在这种现象，即某些第三方应用没有合理的使用唤醒锁机制，例如并非必要情况申请 WakeLock 或持有 WakeLock 未及时释放，甚至还有些恶意应用程序，在灭屏状态下通过唤醒锁的设置唤醒系统，达到其后台运行的目的，偷偷的执行任务。而且随着锁机的功能越来越多，用户安装的应用程序也很多，或多或少都会存在一些应用程序没有合理的试用唤醒锁，导致系统不能正常进入休眠，从而造成了耗电异常。为了避免这些情况，需要对唤醒锁进行有效的管理，论文通过分析唤醒锁的类型，在不影响应用正常工作的前提下对其申请的唤醒锁进行处理,让应用及时释放唤醒锁，从而使系统能够正常休眠[24]，达到节省电量的目的。

4.1.2 唤醒锁管理策略

Android 系统采用唤醒源来进行电源管理，上层可以使用唤醒锁 PowerManager.WakeLock 来控制系统保持唤醒。在灭屏待机状态下,应用程序要在后台运行,通常会首先设置一个用来唤醒系统的闹钟，待系统被唤醒后，再设置一个 PARTIAL_WAKE_LOCK（以下称 PARTIAL）类型的唤醒锁来保持 CPU 的运行，使系统不能再次正常进入休眠，从而达到其后台运行的目的。

在程序设计中，应用通过 acquire 函数获取唤醒锁来保持系统唤醒，通过 release 函数释放唤醒锁来使系统休眠。 acquire 函数的实现在 PowerManager 里，在使用时会传递唤醒锁的类型 Flag，应用 uid 和包名等参数，因此可以根据对这些参数的判断，制定唤醒锁的处理策略。唤醒锁管理策略如下：在灭屏状态下，如果获取的是 PARTIAL 类型的唤醒锁，在灭屏到获取唤醒锁的这段时间内系统休眠过，且获取唤醒锁的应用程序不在白名单内，则只允许该唤醒锁保持 400 毫秒。通过这个策略及时的释放唤醒锁，避免应用程序一直持有唤醒锁导致系统不能进如休眠。

判断系统在灭屏到获取唤醒锁的这段时间内是否休眠过的原因，是为了避免手机灭屏时仍然需要运行，比如存在下载任务或录音时，却不合理的将 WakeLock 释放，从而影响系统正常工作。若系统没有休眠过，则说明仍然有必要的任务正在运行；否则说明系统任务已经完成或无任务运行，此次唤醒中可以释放该唤醒锁。白名单是用于保护某些重要应用程序或任务申请的唤醒锁不被处理，如播放音乐和打电话时的唤醒锁不能被处理。由于某些网络应用在接收到消息后，需要 300ms 左右才能处理完消息的提醒，例如震动和响铃，固将唤醒锁的保持时间设置为 400ms。

4.1.3 设计与实现

(1). 唤醒锁处理流程

当应用调用 PowerManager 中的 acquire 函数申请唤醒锁时，系统会接着调用 acquireWakeLock 函数，而对唤醒锁的处理就是在 acquireWakeLock 函数中实现，依据唤醒锁管理策略，只针对 PARTIAL 类型的唤醒锁进行处理。首先，通过 isScreenOn 函数判断当前手机屏幕状态，若为亮屏，则不做处理；否则，继续进行下一步。判断唤醒锁是否为 PARTIAL 类型，只处理 PARTIAL 类型的唤醒锁，对其他类型的唤醒锁不用做处理。若为唤醒锁为 PARTIAL 类型，继续判断从灭屏到申请唤醒锁的这段时间内系统是否休眠过，通过自定义的系统属性 daemonService.sleep.flag 的值判断的。若系统没有休眠过，说明系统可能还在执行其他任务，不作处理；否则，判断申请唤醒锁的应用是否在白名单中，对非白名单中的应用申请的 PARTIAL，唤醒锁设置在 400 毫秒后自动被释放。通过发送 400ms 的延时 mHandler，触发对该唤醒锁的释放，完成处理工作，使系统能够及时休

眠。白名单的作用是用于保护电话、多媒体和一些重要的系统服务申请的唤醒锁不被处理，避免系统不能正常运行。对唤醒锁的处理流程如图 4-1 所示。

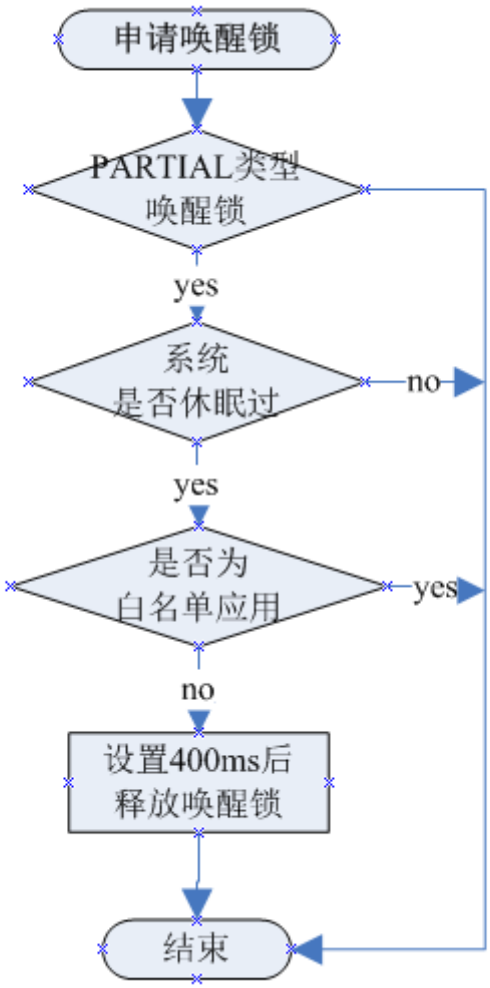

图 4-1 唤醒锁管理流程图

Fig.4-1 Wake Lock Management Diagram

4.2 后台进程清理

4.2.1 Android 进程概述

在 Android 系统中，应用程序都是运行在 java 虚拟机环境中，每个应用程序启动时，系统会先在内存中创建一个虚拟机实例来运行这个应用程序，每个虚拟机实例都作为一个进程，每个应用程序都运行在各自的虚拟机环境中。正常情况下，应用程序无法直接访问其他应用程序的数据和资源，通过该机制确保应用

程序相对安全和独立的运行空间，同时虚拟机也具有进行线程管理、堆栈管理、异常管理和垃圾回收等功能。

当用户退出某个应用程序时，系统不会立即关闭该应用程序的进程，而只是将该应用程序进程挂在系统后台，即系统实际上并没有关闭该应用程序。只有当系统内存空间不足时系统进行内存回收，才会彻底关闭一些挂在后台的应用程序，释放内存资源。这样设计的目的是加快应用程序再次启动的速度，用空间换时间，例如初次启动一个应用程序，需要进行虚拟机实例化并分配内存空间和进程来运行该应用程序，会有不少时间开销；而将应用程序挂在后台，在下次启动该应用程序的时候就能立刻显示出 UI 界面，且不需要再次分配内存空间和进程，能够极大的加快应用程序的启动速度。

应用程序进程挂在后台也会占用 CPU 资源，且某些应用程序在后台长期占用 CPU 资源没有及时释放，甚至有些恶意程序会在后台设置定时器，通过唤醒锁定时唤醒系统并发送数据，导致系统不能正常休眠，从而出现手机高耗电与耗电异常的情况。针对应用程序在挂在后台没有及时的释放并占用大量 CPU 资源导致的高耗电与耗电异常的情况，本方案通过检测后台应用程序的 CPU 占用率，来判断应用程序是否正常运行，并在不影响用户正常使用的前提下，合理的对异常后台应用程序进行筛选并清理，保证系统亮屏时后台无耗电异常的应用，灭屏后能够正常进入休眠，从而有效降低 CPU 的占用率和评论，达到降低手机功耗提升续航能力的目的。

4.2.2 后台清理策略

为了合理的监控后台应用程序的 CPU 占用率，要考虑到亮屏状态和灭屏状态下的差异，一般亮屏状态用户都是在正常的使用手机及应用程序，而灭屏状态下手机也可能在运行音乐或网络下载等任务，为了不影响用户正常的使用和操作，也将后台清理分为亮屏和灭屏两种情况进行。亮屏状态下的清理以前台应用程序切换后台的事件做为触发，当用户重应用程序界面切换到桌面时，即所有运行的应用进程都会被挂到后台，这时发出系统广播，启动清理服务执行亮屏清理流程；灭屏状态以灭屏广播做为触发，当收到系统灭屏广播时，说明手机将进入灭屏状态，此时启动灭屏清理服务对后台应用程序及进程进行清理。

亮屏状态下，当接收到广播信号时，不会立刻启动监控服务，而是设置一个

时间阀值，例如 400ms，当等待 400ms 后再启动监控服务进行后台清理过程。延时启动清理服务有两方面的原因：一方面是给刚被切换到后台的应用预留足够的时间，完成切换过程必要的工作；另一方面是为了防止监控服务处理时占用 CPU 资源，影响到前台应用的启动速度。亮屏后台清理策略如下：通过预先对应用程序的 CPU 占用率设置一个阀值上限，启动清理服务后会统计各后台应用程序的 CPU 占用率，筛选出 CPU 占用率超过阀值上限的应用程序名单，依次清理这些应用，若筛选出的结果为空，则表明不存在 CPU 占用率异常的应用程序，不用进行清理操作。

灭屏状态下，当接收到灭屏广播信号时，同样不会立刻启动后台进程清理服务，此处处理与亮屏状态下的流程一样，也是延时 400ms 后再启动后台进程清理服务。灭屏后台清理策略如下：通过在系统服务框架下实现了一个实时存在的系统服务，并设置黑名单应用列表，通过网络同步黑名单的应用程序数据。同时监听手机的灭屏广播 android.intent.action.SCREEN_OFF，当发现手机锁灭屏时，延时启动后台进程清理的服务并读取黑名单应用程序列表，同时进行筛选 CPU 占用率异常的应用程序，筛选完成后，对异常占用 CPU 的应用程序和黑名单应用程序统一进行清理，kill 掉这些异常的应用程序和进程，确保手机灭屏状态下使能够正常进入休眠，避免存在应用的后台异常耗电行为。

4.2.3. 设计与实现

亮屏后台清理是在亮屏状态下执行，而灭屏后台清理只在灭屏状态下执行，同时都考虑到避免影响用户正常使用的情况，下面分别介绍亮屏状态下后台清理和灭屏状态下后台清理的设计与实现。

1. 亮屏状态下后台清理

当接收到 android.intent.action.ChangetoBackHome 广播时，说明有前台应用切换到桌面的动作，此时就会延时启动处理服务。当服务启动后，需要获取当前后台各应用程序的 CPU 占用率，通过构造用于获取 CPU 占用率的 top 命令，执行 top 命令，其返回结果即是各个应用程序的 CPU 占用率。根据设置 CPU 占用率的上限阀值，筛选出高于上限阀值的应用程序。筛选完成后，开始处理筛选结果。先判断此次筛选结果是否为空，若本次筛选结果内容为空，则说明本次处理无占用率高于上限阀值的应用程序，因此不需要进行清理；若筛选结果不为空，

则将此次筛选结果假如到清理任务队列中，依次对这些后台应用程序 CPU 占用率异常的进程进行清理，亮屏状态下后台清理流程如图 4-2 所示：

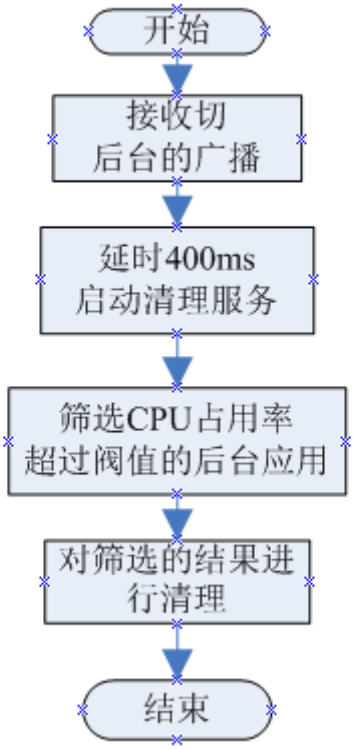

图 4-2 亮屏状态下后台清理流程图

Fig.4-2 Cleaning Process Diagram of SCREEN ON

(2). 灭屏状态下后台清理

接收到灭屏广播后，延时 400ms 启动清理服务后，读取应用黑名单列表，同时通过 top 命令检测并筛选 CPU 占用率异常的应用程序，将黑名单中的应用名单与筛选出的结果都加入到待清理列表中，然后遍历系统正在运行的进程，通过 forceStopPackage 函数清理列表所有应用中相应的进程，其中灭屏状态下对异常占用 CPU 应用程序清理过程同亮屏后台清理部分。由于某些应用程序还会存在守护进程，以便于应用程序被关掉后继续重新启动，所以方案中也对带清理应用程序的守护进程进行清理。操作过程如下，通过 ps 命令筛选所有应用进程，然后依次判断进程是否为待清理应用程序进程的守护进程，即进程 uid 是否与待清理应用程序进程相同，若存在守护进程，则通过”kill -9”命令进行清理。因此，

通过 forceStopPackage 函数与 kill 命令的配合，完成了将获取的待清理应用程序列表中的进程彻底的清除。灭屏状态下后台清理流程如图 4-3 所示：

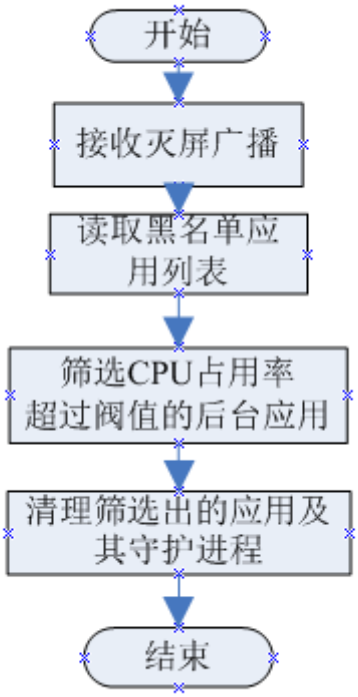

图 4-3 灭屏状态下后台清理流程图

Fig.4-3 Cleaning Process Diagram of SCREEN OFF

4.3 应用联网管理

4.3.1 联网管理概述

针对智能手机灭屏待机时存在应用程序通过网络频繁唤醒系统，造成待机过程耗电异常的现象，对此问题通过实现灭屏状态下对应用的联网行为进行有效的管理，实现系统在休眠时对应用程序不合理的网络行为进行有效控制，从而达到省电的目的。

4.3.2 联网管理策略

考虑到应用程序的不同的使用场景，某些应用在灭屏时很少或不被用户使用，联网管理的策略是让用户在系统菜单中设置灭屏时可以进行联网的应用程序白名单列表，在灭屏的状态下，不禁止这些用户设置的白名单应用程序的联网行为，而对应非白名单中的应用程序，在灭屏是禁止联网，以避免一些应用程序在

灭屏时通过网络频繁唤醒系统的情况。禁网流程如下，在系统接收到灭屏广播后，先获取用户设置的禁止灭屏联网的应用程序列表，然后通过对 IPTable 的设置实现对应用程序列表中的应用进行禁网处理，实现应用程序灭屏禁网功能；在系统接收到亮屏广播后，再次设置 IPTable 应用程序列表中被禁网的应用，允许它们在亮屏状态下联网，并验证应用程序的网络状态是否被正确恢复，以避免影响手机的正常使用。

4.3.3 联网管理设计与实现

在 Android 上，当屏幕状态发生改变时，系统会将屏幕状态发生变化的消息通过发送系统广播给相关的应用，通知它们针对屏幕状态的变化进行相应的处理。例如，灭屏时，系统会发送灭屏广播 android.intent.action.SCREEN_OFF，收到该广播后就知道屏幕将进入灭屏状态，系统可以关闭背光等并进行相应的处理；而亮屏时，系统会发送亮屏广播 android.intent.action.SCREEN_ON，通知系统打开按键灯、背光并进入到 keyguard 锁屏界面等等。联网管理主要也是通过系统广播进行处理与实现的，分别针对对灭屏广播对应用进行禁止联网处理与针对亮屏广播恢复应用网络设置的两部分处，详细的设计与实现流程如下。

1. 灭屏广播的处理流程

对系统灭屏广播 android.intent.action.SCREEN_OFF 进行监听，当系统灭屏时，会接收到灭屏广播信号，即可以开始进行应用禁网处理流程。首先通过系统的包管理服务 PackageManger 获取手机所有的应用程序与对应的系统 UID 列表，然后再获取系统内的保存的应用白名单，这样就能得到非白名单内的所有应用 UID 列表，该列表中的应用程序就是需要在灭屏状态下进行禁网限制的。

禁止应用程序联网是通过的方法是通过对 iptables 进行设置，首先在 startWhenScreenOff 函数中调用 getInitialIptableState 函数，执行”iptables -L -n”命令并解析返回后数据，将该数据与用户当前的应用联网管理状态保存下来，作为亮屏后再恢复用户当前联网状态的基础。然后在灭屏的情况下调用 FirewallApi.startWhenTimeReady 进行灭屏时的设置，将非白名单内的所有应用 UID 列表作为参数传给 applyIptablesRulesImpl 函数进行设置 IPTable，完成对非白名单内应用程序的禁网。

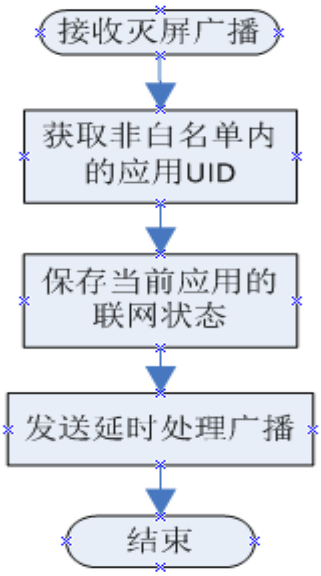

图 4-4 灭屏状态下后台清理流程图

Fig.4-4 Cleaning Process Diagram of SCREEN OFF

2．亮屏广播的处理流程

通过对系统亮屏广播 android.intent.action.SCREEN_ON 进行监听，当接收到亮屏广播后，首先对亮屏设置前提条件进行判断。判断包括以下内容：1、当前是否为系统第一次开机启动发出的广播，因为首次开机时之前没有对应用进行网络管理，固也不需要恢复用户联网管理状态；2、灭屏时禁止应用联网的设置是否执行了，因为存在用户在短时间内反复灭屏和亮屏操作的特殊情况，需要对禁网操作也做延时处理，避免灭屏禁网与亮屏恢复动作出现冲突，要在完成灭屏禁网流程的情况下才对接收到的亮屏广播进行恢复网络操作。当以上 2 个判断条件都满足时，才能进行接下来的操作。调用 FirewallApi.startWhenScreenOn 恢复用户联网设置，将之前保存有用户联网管理状态的文件作为为参数去调用 applyIptablesRulesImpl 函数恢复 IPTable 的设置，还原用户的联网管理的状态，最后调用 FirewallApi.checkAfterScreenOn 验证恢复的结果是否正确。通过 getInitialIptableState 函数执行”iptables -L -n”命令并解析返回数据，将获取的数据与灭屏后保存的用户联网状态数据相对比，若与之前用户联网状态数据不同，则说明恢复失败，需再次执行恢复动作，否则确认恢复成功。

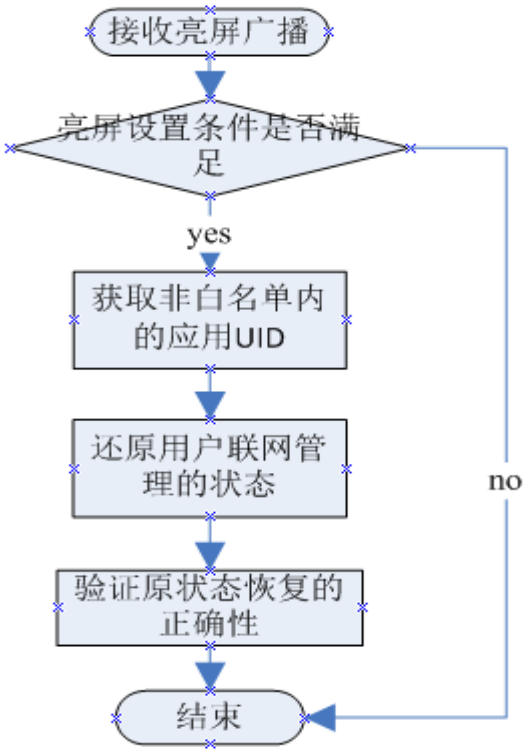

图 4-5 亮屏状态下后台清理流程图

Fig.4-5 Cleaning Process Diagram of SCREEN ON

4.4 省电模式

4.4.1 省电模式概述

省电模式是一种用户模式，类似飞行模式，用户进入省电模式时系统会强制关闭大部分耗电的服务，只保留最常用的短信、通话、时钟、联系人等功能，达到最大化省电的目的。省电模式的实现初衷在用户在不使用手机的时候，可以只保留手机最基本的功能，最大化的提升续航时间。以前有一些方案也是定义几种简单的省电模式供用户快速选择，根据不同的模式快速的设置屏幕亮度等级、自动灭屏时间和铃声大小等以达到省电的目的，但这种方案还较为粗浅，较大的耗电项比如屏幕亮度也只有切换为较低亮度才能真正实现省电，其他的耗电项比如蓝牙、wifi 等的耗电其实并不多，但是直接降低屏幕亮度却会影响一些用户的正常使用，为用户所诟病，各种模式的省电效果并不明显，因为 Android 系统耗电大的时候，多数是因为应用程序在后台运行和定时启动唤醒 CPU 等。而省电模式能能够切实做到省电的效果，它完全断绝了手机内其他应用程序的影响，使得

用户在不需要使用应用程序的时候，关闭大部分的器件，仅保留电话、短信、联系人和闹钟等四个基本功能，此时智能手机相当于一个最古老的功能机，能够最大化的提升待机续航时间。

4.4.2 省电模式策略

表 4-1 省电模式下硬件状态

Table4-1 Hardware Condition on Power Saving Mode

硬件	状态
CPU 频率	限频，660MHZ 以下
网络选择	2G 网络
锁屏时间	15s 自动锁屏
红外传感器	关闭
按键灯	关闭
触屏反馈	关闭
拨号提示音	关闭
蓝牙	关闭
WLAN	关闭
GPS	关闭
NFC	关闭
移动网络	关闭

省电模式有 2 个入口，一种是用户主动选择从普通模式进入到省电模式，另一种是手机电量低于 10%时主动提示用户进入到省电模式。当进入到省电模式时，会发送一个系统广播，通知系统切换到省电模式并做一系列操作和处理，如设置省电模式的标识、发出通知广播、设置屏幕和关闭后台进程等操作，同时系统也会对手机上的硬件状态进行设置，以节省电量，省电模式下硬件的状态如表 4-1 所示。

退出省电模式时会恢复之前的系统状态和设置，例如恢复屏幕亮度值、锁屏时间设置、触屏反馈、移动网络和蓝牙/WLAN/GPS 等等，但省电模式下正常运行的四个模式，电话、信息、联系人、和闹钟时钟不需要做任何改变。

4.4.3 设计与实现

省电模式其他的模式不同，在省电模式下系统会进入另一个界面[25]，同时会杀死大多数进程和应用，关闭所有可能耗电的设备，省电模式会牺牲手机上的大部分功能，只保留联系人、通话、短信以及闹钟最基本的四个功能，以满足用户超长续航的需求。省电模式详细流程如图 4-8 所示：

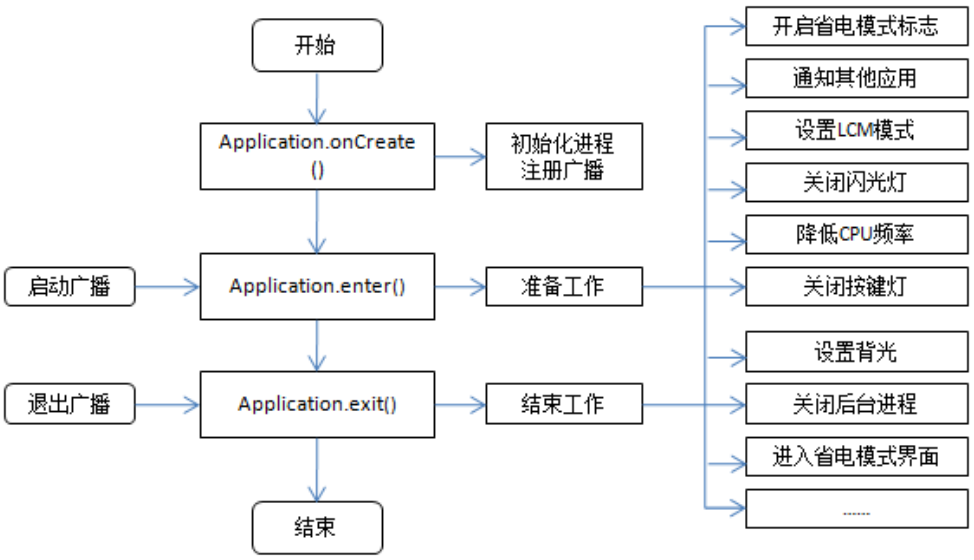

图 4-8 省电模式流程图

Fig.4-8 Power Saving Mode Diagram

4.5 本章小结

亮屏状态下，后台异常清理功能实现了后台应用的监控，并对其中异常占用 CPU 的不良应用进行清理，完成了亮屏状态下后台耗电的管控。灭屏状态下，联网管理结合 WakeLock 省电模型可以很好的解决系统休眠时异常唤醒导致的耗电过多问题。灭屏清理功能作为最终处理方法，对某些顽固的应用进行清理，保证系统正常休眠。通过多种手段的处理，完成了灭屏状态下后台耗电的管控。亮屏状态和灭屏状态下的各种管控手段交替配合，形成互补，确保了手机无耗电异常的情况，提升系统的整体续航能力，达到系统省电管理的目的。

第五章 测试与验证

为了验证省电管理设计方案的实际效果，本章将对省电管理的方案进行功耗测试，通过测试数据给出结论。首先介绍 Power Monitor 的相关使用，然后使用 Power Monitor 连接手机，进行常用应用场景和后台耗电的测试，对测试出来的数据进行统计和分析，验证省电管理的作用与实际效果。

5.1 测试工具与环境

5.1.1 Power Monitor 使用

1. Power Monitor 介绍

Power Monitor（功耗测试仪）可以监控手机电压和电流的变化，并且用曲线图展示所获取到的电压和电流数据，有助于测试和分析，对手机功耗测试很有帮助。Power Monitor 如图 5-1 所示。

图 5-1 功耗测试仪

Fig.5-1 Power Monitor

2. 连接仪器

Power Monitor 前端红色和黑色接口分别为电源的正和负极接口，可以为手机提供指定电压的电源，但手机收没有接口直接与 Power Monitor 连接，一般是从手机电源的正负极端口飞线出来，然后分别与 Power Monitor 正负极端口连接，如图 5-2 所示。这样就将手机与 Power Monitor 连接好了，然后用 USB 线将 Power Monitor 与台式计算机连接起来，并插上电源，Power Monitor、待测手机和台式计算机三者就连接好了。

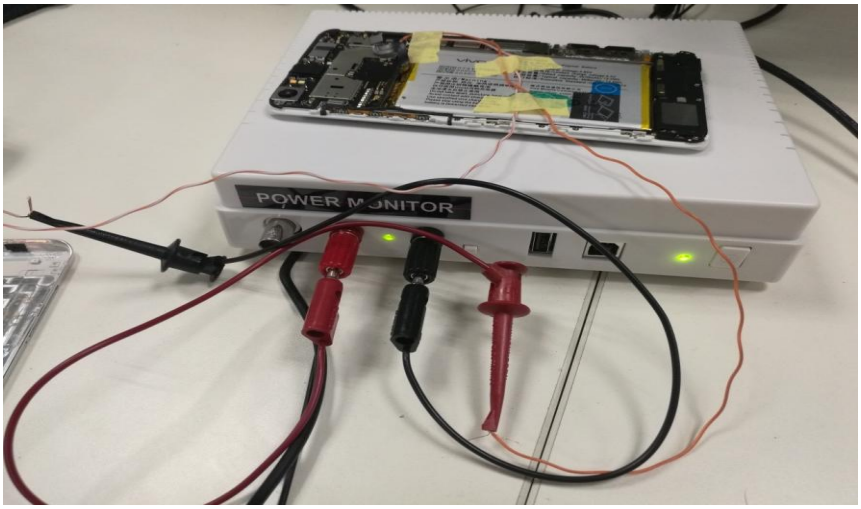

图 5-2 连接手机

Fig.5-2 Connect with Mobile

3. 设置电压

打开 Power Monitor 前还需要先在台式计算机上安装好 power monitor 的设备驱动程序和客户端软件（PowerTool）。都安装好后，在台式计算机上能识别到功耗测试仪设备，然后打开 PowerTool，可以正式开始对手机进行测量了。

在 PowerTool 客户端软件界面右上角对供电电压进行设置。如图 5-3 所示，首先点击 Set Vout 按钮进行电压设置，可以手动输入电压值，但必须在 2.01V 到 4.55V 之间，精度为 0.01V，否则会弹出警告窗口。设置好电压后点击 Vout Enable 按钮便开始对手机进行供电，这样手机就能在 PowerMonitor 的供电下正常使用了。同样的通过点击 Vout Disable 按钮会断开 PowerMonitor 供电，为避免仪器损坏，在断开手机与 PowerMonitor 的连接前要先点击 Vout Disable 停止供电。

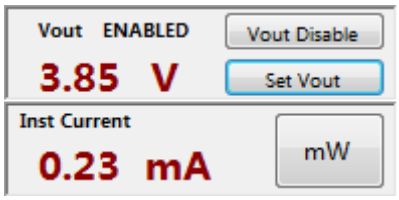

图 5-3 设置电压

Fig.5-3 Voltage Setting

5.1.2 绘制曲线图

1. 开始绘制

在 PowerTool 的右下角找到 RUN 按键，点击 RUN 后 Measured Power Data

就会开始进行数据采样并绘制曲线，如图 5-4 所示。

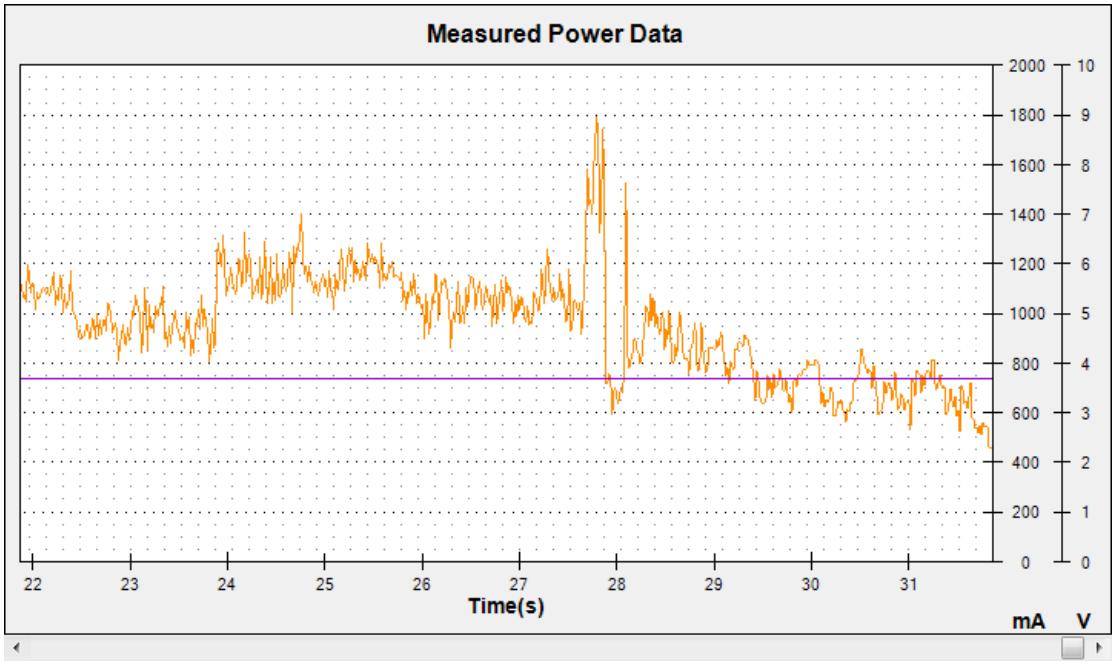

图 5-4 绘制曲线图

Fig.5-4 Measured Power Diagram

2. 调节曲线显示

在绘图过程中也可以对横纵坐标的比例进行调整，在 PowerTool 软件中找到如图 5-5 所示区域，设置时间和电流等。

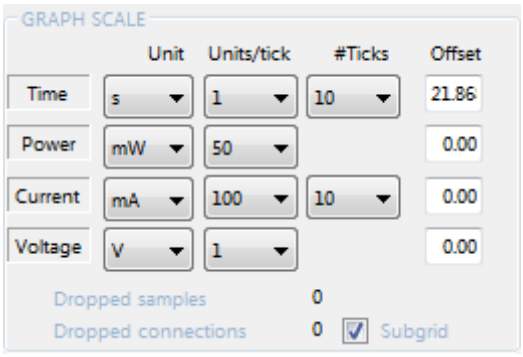

图 5-5 设置电压

Fig.5-5 Power Monitor

3. 选中区域

如果想要放大某一部分数据，可以鼠标圈住数据所在区域。此时图形的绘制会暂停，但是数据仍然在记录。双击该区域，会放大。按 Esc 会退出当前的放大模式，图形会继续绘制，我们看放大图期间的数据也会被绘制。

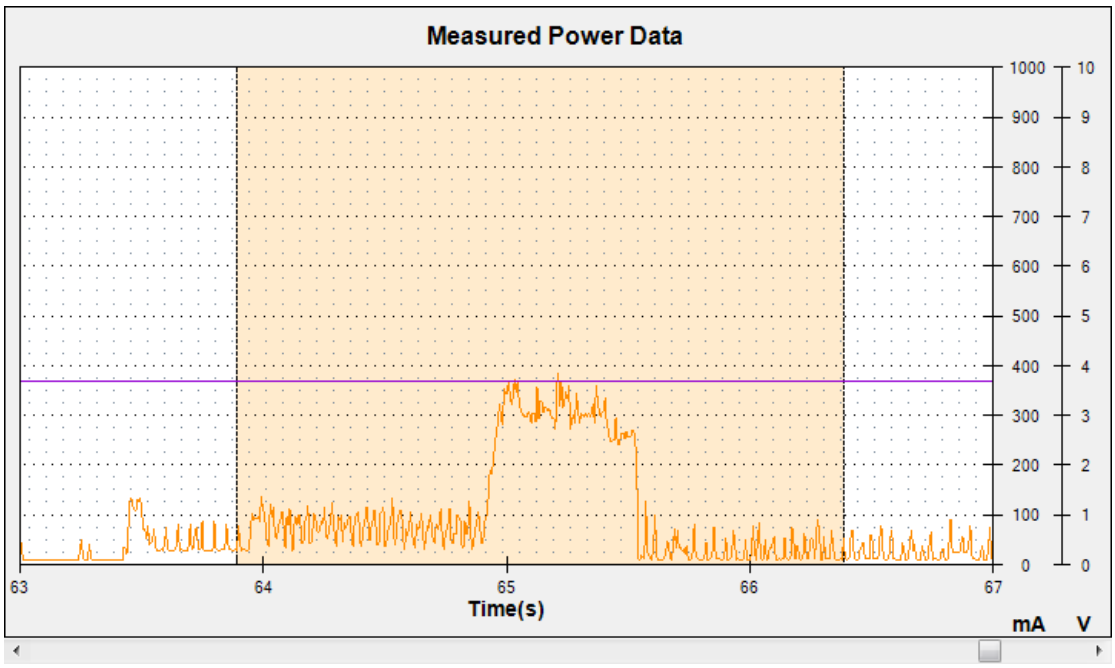

图 5-6 选中区域

Fig.5-6 Select the Areas

4. 数据的保存和导出

点击 STOP 按键后停止采样数据，有 Open、Save 和 Export 三个按键可以对之前的测量数据进行保存和导出。Open 键将数据用 pt3 或 pt4 文件格式打开；Save 键将数据保存在电脑上；Export 键将数据导出为 csv 或 pt4 格式的文件。

5.2 测试数据与分析

5.2.1 场景功耗测试

场景功耗测试是指测试手机在看视频、上网、阅读、通话、听音乐和待机等应用场景下的平均电流，通过平均电流大小反映手机在各个场景下的耗电情况。

如表 5-1 所示常用场景平均电流测试结果，可以看出：待机状态下，手机功耗最低，平均电流约为 4mA；在听音乐和亮屏待机状态场景下，整体功耗较小，平均电流约为 40-80mA；在阅读电子书、上网和看视频场景下，手机功耗居中，平均电流为 120-180mA 之间；手机在游戏应用场景下，功耗较大，约为 400 mA,；在拍照应用场景下，整体功耗最大，平均电流为 500-700mA；手机在不同应用场景下，电流功耗相差很大，主要是由于在不同的场景下，使用到的手机硬件设备不同。视屏与上网都会使用的网络、屏幕、背光等，且 CPU 运行在较高的频率，而阅读没有用到网络，通话音乐没有使用到屏幕和背光等，故功耗较低。

表 5-1 平均电流测试

Table5-1 Average Electricity Testing

测试项	测试方法	优化后	优化前	功耗降低率
睡眠待机	手机进入睡眠待机底电流/(60s)	3.63	3.93	7.63%
MP3/耳机	自带播放器/《父亲.mp3》/耳机模式/黑屏/无声/(60s)	38	41	7.32%
MP3/外放	自带播放器/《父亲.mp3》/外放模式/黑屏/无声/(60s)	58	61	4.92%
亮屏待机	离线/待机界面/(60s)	72	78	7.69%
电子书	看电子书/5 秒划一次/(60s)	115	123	6.50%
上网	wifi/网易新闻/打开一个子新闻/5 秒划动一次界面/(60s)	152	157	3.18%
MP4	莫斯科 mp4/(120s)	176	186	5.38%
游戏《雷霆战机》	wifi/在创库里选项找到 智慧曙光操作/无声/(300s)	415	420	1.19%
前置摄像头	强光下拍照（5 秒一张）/(60s)	522	557	6.28%
后置摄像头	强光下拍照（5 秒一张）/(60s)	700	744	5.91%
总体优化效果	功耗降低率=（优化前-优化后）/优化前			5.60%

如图 5-7 所示手机使用场景平均电流，根据平均电流测试结果生成，比较优化前和优化后的平均电流数据，优化后的各使用场景下的测试平均电流均得到一定程度的减小，功耗降低幅度达到 5.6%，达到了系统低功耗设计的优化目标。

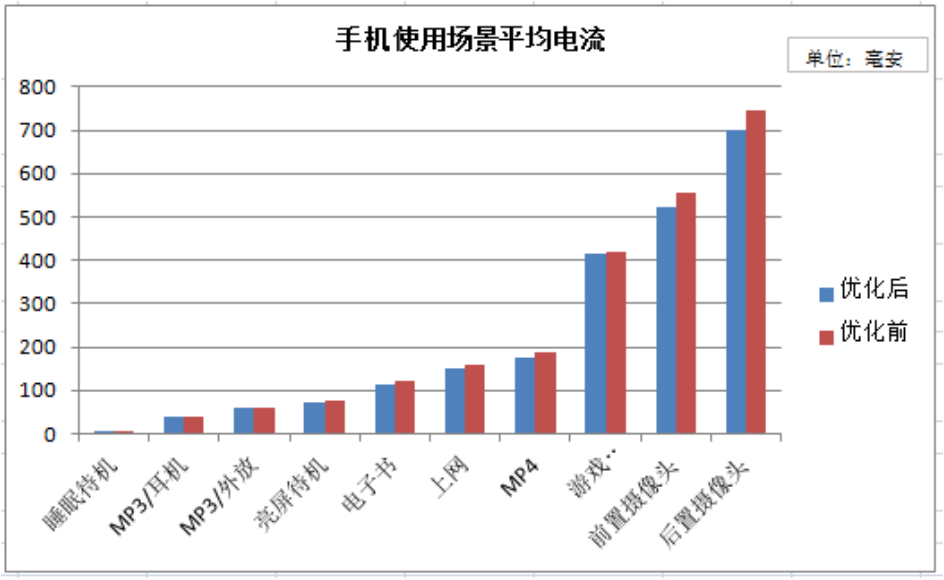

图 5-7 场景功耗测试

Fig.5-7 Applications Scenarios Power Testing

5.2.2 后台耗电测试

后台耗电测试主要是将一些应用程序挂在后台，测试手机的待机功耗情况。

测试步骤如下：1.选取 4 台相同的样机，电池容量为 3000mAH，充满电量到 100%，对其中两台进行低功耗设计优化，另 2 台不做处理。2.分别选取优化前和优化后的 1 台连接 wifi，另 2 台关闭 wifi，并插入手机卡连接 4G 网络；3.选取应用商店中排名 Top20 的应用程序，将它们分别下载到 4 台样机上并全部打开挂在系统后台运行；4.将手机保持该状态待机 15 个小时，分别测出它们的平均电量和耗电量如表 5-2 所示。

表 5-2 系统耗电测试结果

Table5-2 Power Consumption Test Results of System

优化前					
网络状态	**开始电量**	平均电流/mA	耗电量/mAH	结束电量	掉电量
wifi	100%	12.26	184	94%	6%
4G	100%	14.23	212	93%	7%
优化后					
网络状态	**开始电量**	平均电流/mA	耗电量/mAH	结束电量	掉电量
wifi	100%	10.06	151	95%	5%
4G	100%	11.6	174	94%	6%

优化前，wifi 状态下平均待机电流为 12.26 mA，4G 状态下平均待机电流为 14.23 mA；优化后，wifi 状态下平均待机电流为 10.06 mA，4G 状态下平均待机电流为 11.6 mA;优化后 wifi 与 4G 状态下平均待机电流分别降低了 11.4%与 13.1%。

5.3 本章小结

本章首先介绍了用功耗测试仪 Power Monitor 测试手机电流功耗的方法，并用功耗测试仪测试手机在各应用场景下的平均电流大小，计算出手机在个应用场景下的待机续航时间，然后测试优化前后在 wifi 与 4G 状态下的待机平均电流，用测试数据验证了优化后待机电流数据较优化前降低了 10%左右，达到了低功耗设计的预期效果。

第六章 总结与展望

6.1 论文工作总结

1、论文针对智能手机续航能力不足的现状进行分析，总结了智能手机耗电方面存在的问题，结合硬件低功耗设计和软件低功耗设计的实际应用，说明低功耗设计在智能手机中的重要意义。

2、通过对标准的 Linux 电源管理机制和 Android 原生的电源管理机制进行研究，对比 Linux 与 Android 电源管理的差异，并详细研究了 Android 电源管理中的 Early Suspend、Late Resume 和 wake lock 锁机制。

3、基于高通 MSM8953 芯片硬件平台，Android 软件操作系统平台，提出了一种低功耗的设计方案,将唤醒锁管理、后台进程清理、应用联网管理和省电模式多种省电机制相结合，对手机进行综合的省电管理与监控，从而达到降低手机功耗，提升续航能力的目的。

4、介绍了 Power Monitor 的使用方法，搭建手机功耗测试环境，使用 Power Monitor 对手机进行功耗和电流测试，并给出测试数据与分析结果。

5、根据实际应用经验总结了低功耗设计方面目前的工作成果,提出下一步的研究方向。

6.2 未来展望

在撰写论文的过程中，结合了软件项目开发的实际情况，也查阅了大量的文献资料，研究了硬件低功耗设计与软件低功耗设计研究现状与相关方案，论文基于 Android 系统进行的低功耗设计与实现，对手机在各使用场景下的功率有一定程度的降低，故对系统省电也有一定的帮助和效果，但是未来还有可以改进和完善的地方，主要有是以下三个方面。

(1)论文的低功耗设计主要是从操作系统与应用软件层面进行设计与实现的，因为所采用的 MSM8953 硬件芯片平台已有较为成熟的硬件低功耗解决方案，能够很好的处理硬件的电源管理，如果采用其他硬件平台进行设计，需要另外去考虑到相关硬件方面的低功耗设计。

(2)论文中的低功耗设计方案是基于系统框架层去实现的，如果仅通过操作系统的公共接口在应用层有不少关键功能难以实现，但是在系统框架层的实现的功能需要操作系统的 root 权限，导致该方案与软硬件平台的耦合性较高，不易于在其他手机产品上进行移植与兼容。

(3)Android 软件版本更新速度很快，每年会发布一个大版本推出新特性，并发布多个小版本修复与解决 bug。谷歌 10 月份发布了新版本 Android N 对于电源管理方面有较大的改动，论文中的低功耗设计方案也需要基于 Android 版本的变化进行优化与完善。

参考文献

[1] 金大佑，朴宰永，文炳元，Android 系统服务开发[M]，北京，人民邮电出版社，2015，277-279.

[2] 中国互联网协会,2015 中国互联网产业综述与 2016 发展趋势报告[J],2016.01.06.

[3] 王昌林，张勇，李东生，CMOS 集成电路功耗分析及其优化方法，解放军电子工程学院 2006

[4] 陈春鸿，CMOS 集成电路的功耗分析及低功耗设计技术[J].浙江工业大学学报，1998，(9)

[5] Qualcomm，MSM8953 Chipset Introduction Design Guidelines/Training Slides [J]，November，2015.

[6] Qualcomm，Introduction to Qualcomm Quick Charge2.0[J]，September，2014.

[7] Android M 新特性 Doze and App Standby 模式详解[E/OL]，https://zhuanlan.zhihu.com/p/20323263，2015.11.06.

[8] Android 的 Alarm 对齐唤醒机制[E/OL]，http://blog.csdn.net/jakioneplus/article/details/49512607，2015.10.30.

[9] 电源管理方案 APM 和 ACPI 比较[E/OL]，www.voidcn.com/blog/bailyzheng/article/p-5772562.html，2016.4.12.

[10] 高级配置与电源接口[E/OL]，https://zh.wikipedia.org/zh-cn/高级配置与电源接口，2015.6.02.

[11] The Linux Kernel Archives [S/OL]，https://www.kernel.org/doc/Documentation/power/devices.txt，2007.

[12] Linux Kernel suspend/resume 过程[E/OL]，https://blog.csdn.net/xinyuwuxian/article/details/9187641，2013.6.27.

[13] 杨丰盛，Android 技术内幕·系统卷[M]，北京，机械工业出版社，2011，152-155.

[14] 唤醒锁：检测 Android 应用中的 No-Sleep[E/OL]，https://software.intel.com/zh-cn/android/articles/wakelocks-detect-no-sleep-issues-in-android-applications，2014.2.16.

[15] Android early suspend/late resume[E/OL]，www.voidcn.com/blog/mcgrady_tracy/article/p-458724.html，2015.7.13.

[16] Qualcomm，MSM8953 Software Architecture and Migration Overview [J]，November，2015.

[17] Tick-Tock [E/OL],www.baike.baidu.com/,2006.1.5

[18] 什么是 big.LITTLE[E/OL]，http://news.mydrivers.com/1/429/429012.htm，2015.5.15.

[19] 韩超等，Android 核心原理与系统级应用程序高效开发[M]，北京，电子工业出版社，2012，07-12.

[20] Qualcomm，MSM8953 Linux Android Graphics Overview[J]，November，2015.

[21] Qualcomm，Cookie for Android Build[J]，Oct，2015.

[22] Qualcomm，Power Management IC Design Guidelines Training Slides [J]，November，2015.

[23] Qualcomm，Power Management And Heterogeneous Cluster Architecture Overview [J]，Oct，2014.

[24] Qualcomm，MSM8953 Linux Android Software Debug Manual[J]，November，2015.

[25] Qualcomm，MSM8953 Digital Baseband Design Guidelines/Training Slides [J]，November，2015.

致 谢

首先感谢我的导师邓倩妮副教授，感谢邓老师在我撰写论文期间给予的帮助与指导，不论工作有多忙都能及时的审阅论文，并帮助解答疑惑和指明方向。

还要特别感谢张忠能副教授与姚天昉副教授，多次不辞辛苦的从上海飞到深圳来与同学们讨论论文事宜，帮助我们选题，为我们解答各种疑惑，并提供了丰富的资料与经验总结，帮助我顺利的完成了学位论文。在此，表达我最诚挚的感谢。

另外还要感谢杨安老师、张凤英老师和蔡幼玲老师在论文答辩与小论文发表阶段提供的帮助，感谢郑椿萍老师、宋坤老师细心的安排和耐心的提醒，不厌其烦的解答我们各种问题，在遇到困难的时候总能给予帮助。

转眼间在职研究生阶段的学习也即将结束，回首研究生学习期间的经历虽然辛苦，在工作和生活的过程中还要兼顾学习。认识了一群同样努力奋斗的同学们，能够在学习过程中互相指导，互相帮助，还能够时刻的感受到老师们的关心和关怀，合理的课程安排也为我们减轻了不少负担，过程殊为不易但结果美好。研究生的学习经历也会成为我今后的人生财富，也让我铭记“饮水思源，爱国荣校”校训，在未来的工作和学习过程中，我会加倍努力。

最后，我要感谢我的家人，为我创造了良好的环境并给我信心与动力。并再次感谢帮助过我的各位教授、专家、老师、同学和同事，一切都离不开你们的帮助与支持，谢谢！

攻读学位期间发表的学术论文

[1]陈实凡，基于 Android 智能手机的 Push 系统设计 上海交通大学计算机系工程硕士网站公示，2017 年 01 月