To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

RENESAS

应用笔记

电池充电应用

文档编号：U17173CA1V0AN00
发布日期： 2007 年 8 月

德国印制

CMOS设备的注释

① 输入引脚处的电压波形

输入噪音或一个反射波引起的波形失真可能导致错误发生。如果由于噪音等的影响使CMOS设备的输入电压范围保持在VIL(MAX)和VIH(MIN)之间，设备可能发生错误。在输入电平固定时以及输入电平从VIL (MAX)过渡到VIH (MIN)时的传输期间，要防止散射噪声影响设备。

② 未使用的输入引脚的处理

CMOS设备的输入端保持开路可能导致误操作。如果一个输入引脚未被连接，则由于噪音等原因可能会产生内部输入电平，从而导致误操作。CMOS设备的操作特性与Bipolar或NMOS设备不同。CMOS设备的输入电平必须借助上拉或下拉电路固定在高电平或低电平。每一个未使用引脚都应该通过附加电阻连接到VDD或GND。如果有可能尽量定义为输出引脚。对未使用引脚的处理因设备而异，必须遵循与设备相关的规定和说明。

③ ESD防护措施

如果MOS设备周围有强电场，将会击穿氧化栅极，从而影响设备的运行。因此必须采取措施，尽可能防止静电产生。一旦有静电，必须立即释放。对于环境必须有适当的控制。如果空气干燥，应当使用增湿器。建议避免使用容易产生静电的绝缘体。半导体设备的存放和运输必须使用抗静电容器、抗静电屏蔽袋或导电材料容器。所有的测试和测量工具包括工作台和工作面必须良好接地。操作员应当佩戴静电消除手带以保证良好接地。不能用手直接接触半导体设备。对于装配有半导体设备的PW板也应采取类似的静电防范措施。

④ 初始化之前的状态

在上电时MOS设备的初始状态是不确定的。在刚刚上电之后，具有复位功能的MOS设备并没有被初始化。因此上电不能保证输出引脚的电平，I/O设置和寄存器的内容。设备在收到复位信号后才进行初始化。具有复位功能的设备在上电后必须立即进行复位操作。

⑤ 电源开关顺序

在一个设备的内部操作和外部接口使用不同的电源的情况下，按照规定，应先在接通内部电源之后再接通外部电源。当关闭电源时，按照规定，先关闭外部电源再关闭内部电源。如果电源开关顺序颠倒，可能会导致设备的内部组件过电压，产生异常电流，从而引起内部组件的误操作和性能的退化。

对于每个设备电源的正确开关顺序必须依据设备的规范说明分别进行判断。

⑥ 电源关闭状态下的输入信号

不要向没有加电的设备输入信号或提供I/O上拉电源。因为输入信号或提供I/O上拉电源将引起电流注入，从而引起设备的误操作，并产生异常电流，从而使内部组件退化。

每个设备电源关闭时的信号输入必须依据设备的规范说明分别进行判断

详细信息请联系：

（中国区）

网址：
http://www.cn.necel.com/
http://www.necel.com/

[北京]

日电电子（中国）有限公司
中国北京市海淀区知春路 27 号
量子芯座 7，8，9，15 层
电话：（+86）10-8235-1155
传真：（+86）10-8235-7679

[上海]

日电电子（中国）有限公司上海分公司
中国上海市浦东新区银城中路 200 号
中银大厦 2409-2412 和 2509-2510 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

上海恩益禧电子国际贸易有限公司
中国上海市浦东新区银城中路 200 号
中银大厦 2511-2512 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

[深圳]

日电电子（中国）有限公司深圳分公司
深圳市福田区益田路卓越时代广场大厦 39 楼
3901，3902，3909 室
电话：（+86）755-8282-9800
传真：（+86）755-8282-9899

[香港]

香港日电电子有限公司
香港九龙旺角太子道西 193 号新世纪广场
第 2 座 16 楼 1601-1613 室
电话：（+852）2886-9318
传真：（+852）2886-9022
2886-9044

- 本文档信息于 2006 年 5 月开始使用。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。
- 并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。本文件所登载内容的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

  “特殊等级”仅适用于为特定用途而根据用户指定的质量保证程序所开发的日电电子产品。另外，各种日电电子产品的推荐用途取决于其质量等级，详见如下。用户在选用本公司的产品时，请事先确认产品的质量等级。

  “标准等级”： 计算机，办公自动化设备，通信设备，测试和测量设备，音频・视频设备，家电，加工机械以及产业用机器人。

  “专业等级”： 运输设备（汽车、火车、船舶等），交通用信号控制设备，防灾装置，防止犯罪装置，各种安全装置以及医疗设备（不包括专门为维持生命而设计的设备）。

  “特殊等级： 航空器械，宇航设备，海底中继设备，原子能控制系统，为了维持生命的医疗设备、用于维持生命的装置或系统等。

  除在本公司半导体产品的数据表或数据手册等资料中另有特别规定以外，本公司半导体产品的质量等级均为“标准等级”。如果用户希望在本公司设计意图以外使用本公司半导体产品，务必事先与本公司销售代表联系以确认本公司是否同意为该项应用提供支持。

（注）

（1） 本声明中的“本公司”是指日本电气电子株式会社（NEC Electronics Corporation）及其控股公司。

（2） 本声明中的“本公司产品”是指所有由日本电气电子株式会社或为日本电气电子株式会社（定义如上）开发或制造的产品。

目录

第一章　引言 6
第二章　电池充电特性 7
2.1　镍镉（Ni-Cd） 7
2.2　锂离子（Li-lon） 8
2.3　铅酸电池（LEAD-ACID） 9
第三章　充电器基础 10
第四章　降压（BUCK）转换 11
第五章　充电器电路 14
第六章　充电器固件 16
第七章　其它充电方法 25
第八章　结论 26
第九章　固件列表 27

第一章 引言

现在在许多产品和系统中都可以看到充电电池（可重复充电的），从小的手持设备比如可携式录像机和轻便游戏机，到中等尺寸的应用象电源工具，到大型系统例如紧急灯、建筑消防和安全系统。在实际应用中，这些电池在消费产品，工业系统，建筑管理系统和更多的地方用来替换锌-碳电池（zinc-carbon）或者碱性电池（alkaline cells），或在正常（主）电源掉电后提供后背电源的设备。

NEC 的微控制器系列适合于电池充电应用，从低管脚的 78K0S 内核到 32 位的 V850 系列。这些微控制器具有丰富的外围硬件，包括电池充电必须的 A/D 转换器和脉冲宽度调制（PWM）。

本应用笔记描述了几个使用 NEC 微控制器对锂电池（Li-Ion），密封的铅酸电池（SLA），镍镉电池（Ni-Cd）进行充电的方法。通过描述几个方法，希望工程师可以对其产品的成本、性能和复杂性作出优化。

一个途径是使用微控制器本身作为降压式开关（Buck）转换器的基础，这样可将外部器件减到最小。通过微控制器检测电池电压和电流，并且为了保持充电参数的稳定，充电器输出是可变的。这种复用提供了一种解释，微控制器这样才能作为控制专用电池充电器的 IC 。这里通过充电器 IC 执行电池电流&电压的封闭循环控制，并且微控制器执行更多的监控角色，以便电池装入的检测，充电的状态和充电的终止。

SLA、Ni-Cd 和 Li-Ion 电池充电具有不同的要求，以下将做说明。

此外，电池容量以毫安时（mAh）来测量其容量，并且缩写字母‘C’来表示，举例来说，一个电池具有 C = 500 mAh 的容量，表示理论上可以提供 1 个小时 500 mA 的电量，2 个小时 250 mA 的电量，等。C 也作为充/放电比率，所以 C/2 对于一个 500 mAh 的电池意味着一个 250 mAh 的充/放电流，C/10 意味着一个 50 mA 的电流，等等。

第二章 电池充电特性

2.1 镍镉（Ni-Cd）

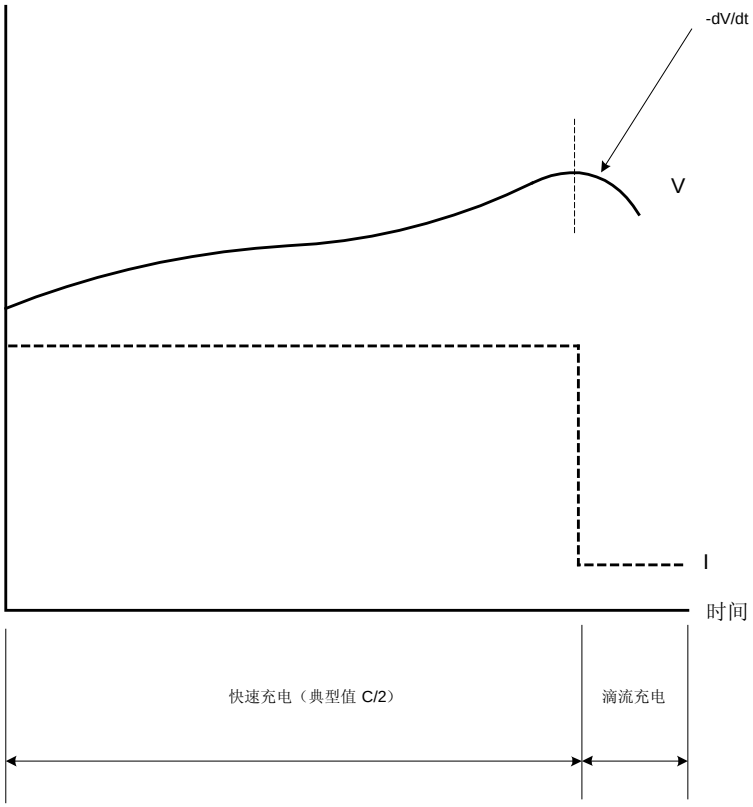


图. 1 镍镉电池充电特性

镍镉电池可以使用和 C 一样高的恒定电流进行充电，但是为了延长电池的寿命和预防过热，经常把 C/2 作为最大的充电电流。电池电压（典型 1.45V/节对于“AA”规格） 在充电时会上升；当电池充满后电池的电压将会下降。这可以通过–dV/dt 条件得到，并且可以被充电器作为一个充满的条件使用。电池的温度会在充满时会突然的上升，这也可以作为一个附加的条件来检测充电器是否充满。

当电池充满时，充电器输出能够被完全关闭或者或者一个低电流负荷用来维持充电。一个范围 C/50 到 C/10 适合这种条件；作为一个选择快速充电阶段可以被完全忽略并且电池只用小电流充电。快速充电 （C/2） 有时候叫做“1 小时充电”（尽管它可能不到一个小时） 而小电流充电叫做“整夜充电”。

2.2 锂离子（Li-Ion）

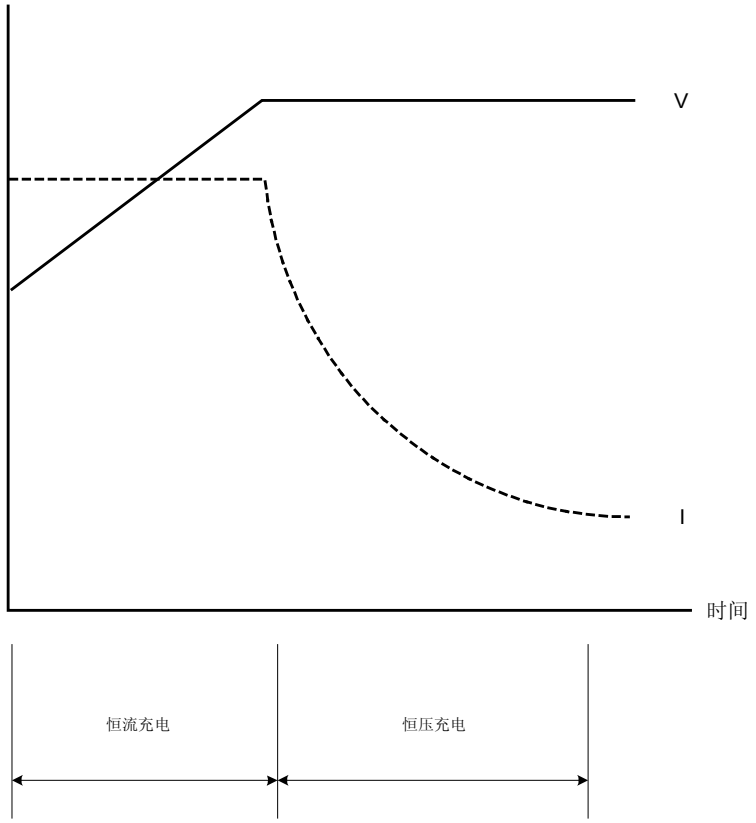


图. 2 锂离子充电特性

锂离子电池（Li-Ion）和镍镉电池（Ni-Cd）相比有不同的充电需求这是由于充电电压的限度是苛刻的，在初始时用一个恒定的电流进行快速充电然后使用一个恒定的电压进行充电。一些充电器不能够提供恒定的电压只是在想得到电压达到之后停止充电，但是这样将只充到了电池最大限度的 70% 。输出电压的公差将减小 0.75%，虽然此图是依据充电中的电池。当电池到达充满电压时充电器的输出电流将逐渐减小（参看图.2）；当这个电流小于一个确定的值（典型值 100-200 mA） 充电器将认为电池充满并且结束充电处理。对于锂电池并不推荐滴流充电，因为支持过充。锂电池（Li-Ion）典型值 3.6V/节。

2.3 铅酸电池（LEAD-ACID）

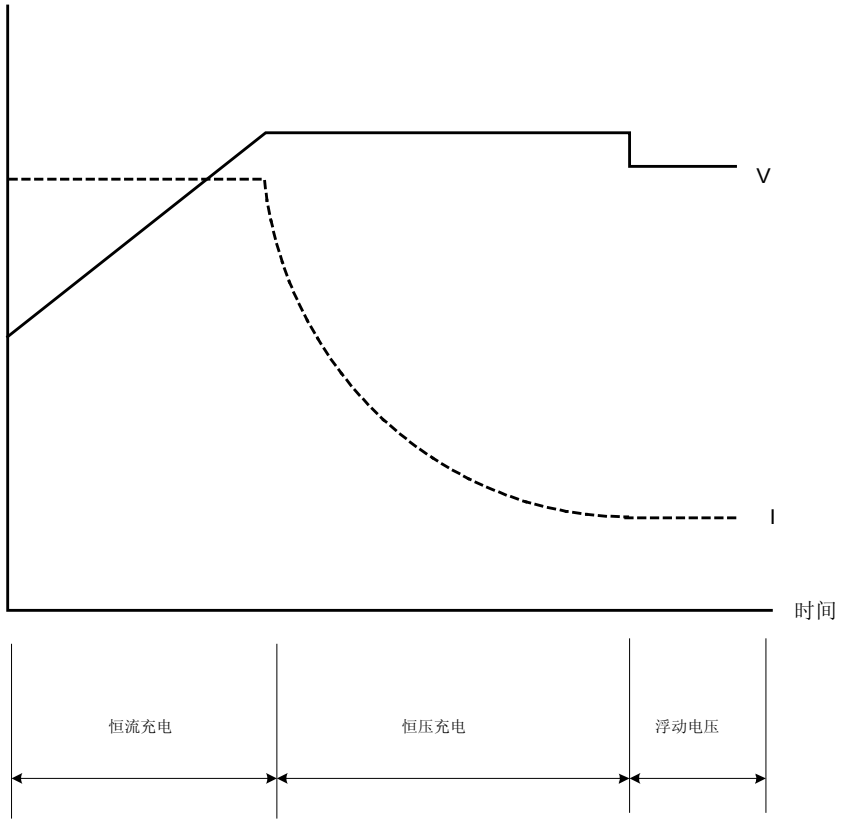


图. 3 铅酸（Lead acid）充电特性

铅酸电池（Lead acid）和锂离子电池（Li-Ion）有一个类似的充电需求由于它们都有恒定电流和恒定电压充电阶段，虽然充电电压应用有一个比锂离子低的严格公差。另外，当电流下降到 3%的最大电流时充电电压可以被从正常的 2.4V/节减小到被称为浮动电压的电压值（典型 2.25V/节）并且这个充电过程可以留下不确定性以避免电池的自放电。

关于上面所有的电池类型，有一些共同点。安全一直是最主要的， 机械装置能够被构建到充电器中去提高它。在其他充电终止方法失败后自动安全定时器可以在一定的时间内结束充电。电池的温度可以被监控（如果 包装上有电热控制器）剧烈的包装温度改变可以触发一个失败模式。 此外，电调节器有时候也用来检测电池是否插入充电器。如果带有温度传感器的充电器用于一个温度变化频繁的环境中，并且其正在读取比较电池温度传感器的值。关闭电池（不是充电器）电压和电流的监视则可以强制其进入安全充电系统。

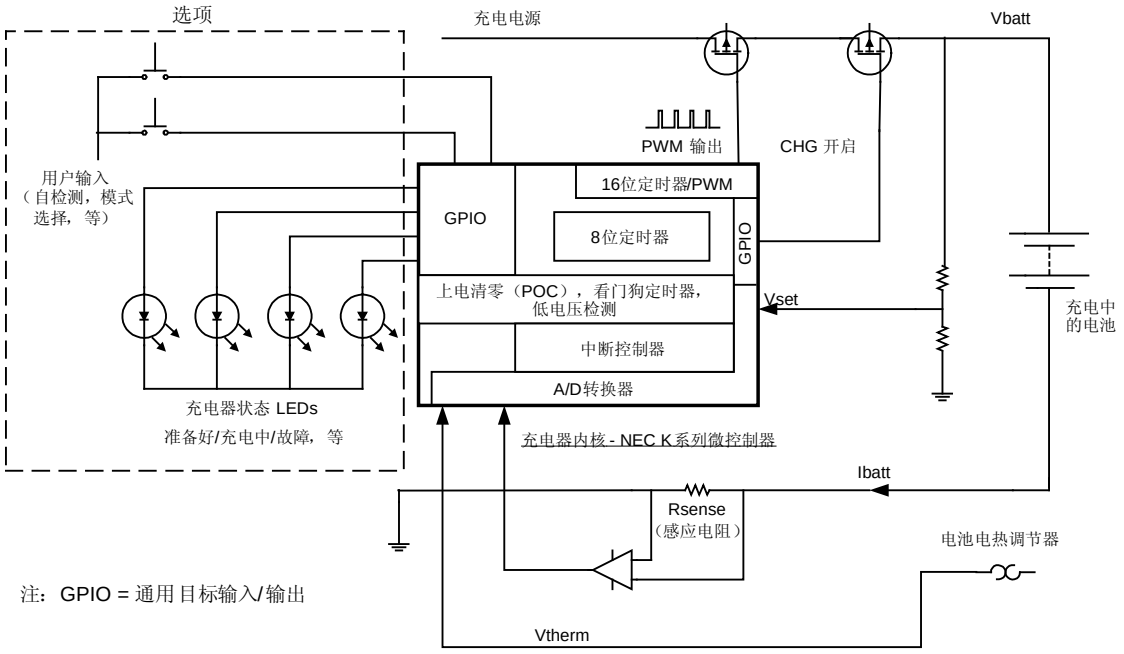


图. 4 使用 NEC K 系列外围硬件的充电器典型模块图

图. 4 展示了一个电池充电系统的例子模块图。有标记的部分可以作为桌面充电器的可选部分，而且如果充电器嵌入到设备中的话，这部分不需要。

系统中主要的器件是一个调节器它用来把 AC 或者 DC 电源转化为电池允许范围内的稳定电压。限流部分也是必须的它用来保证前面方法中所说的恒定电流。调节器可以是一个简单的线性类型，可是为了电源分散和效率的原因通常使用一个开关类型，所以图里面有 PWM 输出驱动。需要一个电流传感放大器它把电池流出的电流转换为电压这个电压被充电器内核用来保持电流恒定。电池的电压必须被反馈到内核用来提供调节。电池温度调节器必须被监控并且在太热的时候动作；有时候也需要一个测试周围环境温度的传感器。附加功能包括设置模式电池类型等的开关和指示属性信息的 LED 或者提供电池充电属性的图形类型显示。

上图中的“充电器内核”可以是一个 微控制器控制的单一模式或者控制一个电池充电器 IC 。

NEC 家族的微控制器非常适合电池充电应用它们具有这个应用需要的 PWM 通道，定时器和模拟数字转。这个充电功能即可以作为微控制器的单独功能也可以作为微控制器其他任务的一个次要功能。另外 NEC 也生产一系列的低电压 MOSFET 它能够由来执行下一节描述的 S 开关功能。

第四章 降压（BUCK）转换

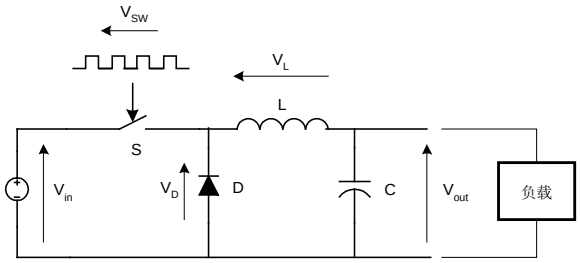
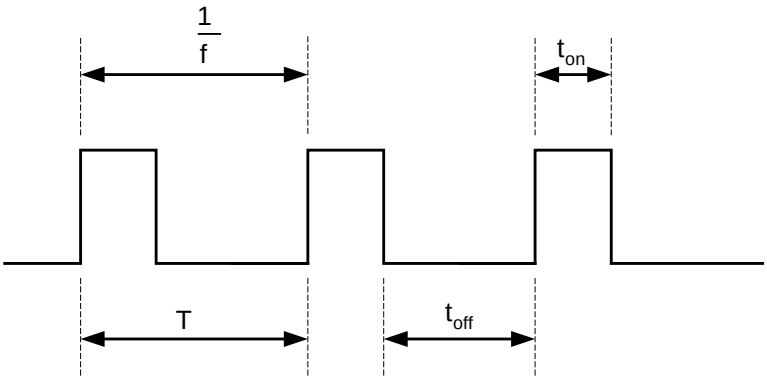


图. 5 Buck 转换器

图. 5 显示了一个 Buck （step-down）转换器。开关‘S’在实际上是一个晶体管（通常上是 MOSFET）它受从微控制器或者电池充电 IC 输出的占空比变化的 PWM 信号控制。当开关 ‘S’被关闭，电流流过它，电感 L、电容 C 和负载的能量被保存在电感器上。当开关打开时，保存在电感器 L 上的电能被释放并且电流通过二极管 D，电容 C 和负载。迅速的重复开关开关产生一个 DC 电平 Vout（包含一些纹波）；这个电压与开关的占空比成比例。



$$\text{占空比} = \frac{t_{on}}{T + t_{on}} \times 100\%$$

图. 6 开关‘S’上的波形

在实际应用中，需要一些形式的调节去补偿电压和负载的变动。图. 4 中反馈给充电器内核的电流和电压被用来修改开关的占空比用来保持输出恒定。

对于一个 Buck 转换器，电感值通过下面的方式给出：

$$L = \frac{V_{in} - V_{out} - V_{SW}}{I_{PK}} t_{on} \qquad \text{公式（1）}$$

V_{in} = 输入电压
V_{out} = 必须的输出电压
V_{SW} = 开关的饱和电压（晶体管）
I_{PK} = 从转换器流出的峰值电流
t_{on} = 开关‘打开’时间

对一个单步向下转换器，

$$I_{PK} = 2I_{max} \qquad \text{公式（2）}$$

这里 I_{max} 是最大的输出电流。

决定电容的电容值使用下面公式：

$$C \geq \frac{I_{PK}(t_{on} + t_{off})}{8V_{ripple}} \qquad \text{公式（3）}$$

下面图. X 显示了电感器上的电流和电压波形。因为 t_{on} 和 t_{off} 的时间长度与 L 所代表的几乎直线的常数时间比较，是小的。

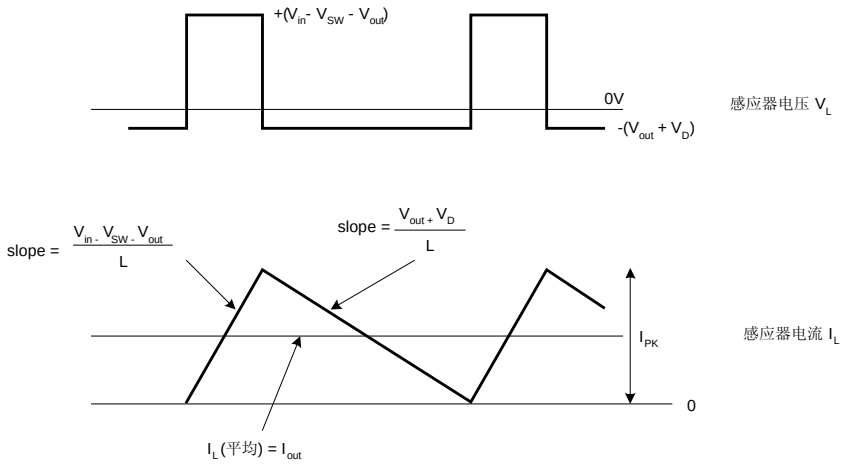


图. 7 电感器波形

例子：

V_{in} = 24V
V_{out} = 12V
V_{SW} = 0.2V
I_{max} = 2A
f = 25KHz

首先，$T = \frac{1}{f} = 40\mu s$

假定 50% 占空比，t_{on} = 20μs

从公式（2），$I_{PK} = 2I_{max} = 2 \times 2 = 4A$

现在，从公式（1），$L = \frac{24-12-0.2}{4} \times 20 \times 10^{-6} = 59\mu H$

现在 C 的值可以使用公式（3）决定：

假定 50mV 纹波可以接受：

$C = \frac{4 \times 40 \times 10^{-6}}{8 \times 50 \times 10^{-3}} = 400\mu F$，所以使用 470μF。

第五章 充电器电路

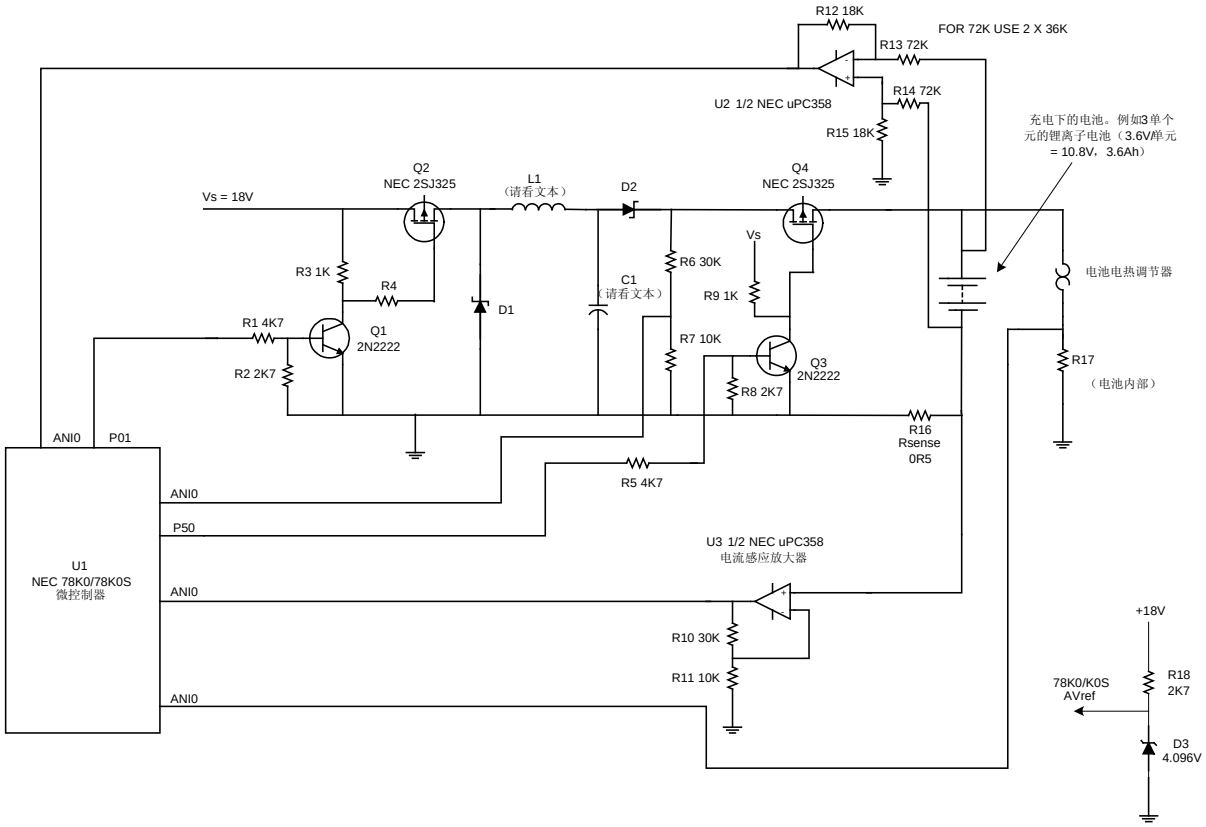


图. 8 充电器示意图

图. 8 显示了电池充电电路的整个示意图。Buck转换器包括 D1、L1、Q2 和 C1。在电路布线的时候，保持这些连线尽量的短。Q4用来作开关控制充电器输出的打开和关断，在测量电池电压的时候如果充电器连接在电池上测量结果将不准确。NPN 晶体管Q1和Q4给MOSFET管Q2和Q4提供门驱动。R6和R7对充电器输出电压的测量进行分压后反馈到ADC。U2组成一个微分放大器用来测量电池电压，U3组成一个电流感应放大器，测量通过传感电阻R16的电压。D3和R18组成一个参考电压提供给微控制器的A/D转换器。在这里没有指定组成公差；它们主要取决于所要充电的电池类型。例如，对于锂离子电池电压传感放大器周边的电阻器可能需要0.1%的精度，参考二极管D3需要0.1%的公差电压， 对于其他的电池化学标准1%构成可能被用到。类似的，L1和C1的精确值取决于电池充电电压，并且可以通过以上公式进行判定。D1和D2是具有充分的电流比率的肖特基二级管。如果充电电流小于要求的1A时，NEC提供uPA507TE，具有肖特基二级管P沟道MOSFET，并保持了SC-95的封装形式，可以用于电路有变动的充电器。R4可以防止MOSFET周边的RF振荡，它的阻值根据MOSFET（C_{iss}）的输入电容，通常为大约100R即可。

以 12V 充电电压下最大电流 2A 为锂离子电池充电为例：

F_{cpu} = 10MHz。V_{in} = 18V，V_{out} = 12.3，I_{max} = 2A

对于 8 位 PWM，PWM 取值：$\frac{12.3}{18}$ X 256 = 205

@ 10MHz，$t_{on} = \frac{1}{10X10^{6}}$ X 255 X $\frac{205}{255}$ = $2.05X10^{-5}$ s

如果 I_{max} = 2A，I_{pk} = 4A 。

因此从公式 (1), L = $\frac{18-12.3-0.2}{4}$ X $2.05X10^{-5}$ = 28μH, 取最接近首选值的数值。

提供足够精确的充电电压是重要的。在这里使用 8 位 PWM 波， 所以产生的误差， 对于上面的例子 12.3V 如下：

1 LSB = $\frac{18V}{256}$ = 0.070V

$\frac{0.070}{12.3}$ X 100% = 0.57%

通过增加 PWM 的分辨率（增加到 10 位）可以提高精度但是 PWM 周期（和 t_{on} 将增加，需要一个更大（和成本更高）的电感器。

第六章 充电器固件

这些代码使用 C 语言在 IAR 的嵌入式工作台编译。参看固件列表获取使用的编译器，汇编器和链接器的版本。

本应用笔记是一个以使用 uPD78F0148 为基础的，从 NEC K 系列产品线在 NEC“Play-It”开发包中。承认 NEC 微控制器家族的可预测性，这个应用比使用 78K0S 家族更具有性价比，或者使用更强有力的 32 位 V850 产品。K 系列产品的外围器件具有很好的兼容性。

代码是一个状态机格式，这样的目的是使电池充电应用不会阻塞其他用户想要在微控制器上运行的应用。例如充电器使用一个 2 秒钟的的延时；等待一个 For 循环将会停止其他功能这是其他应用不接受的。作为替代，一个软件基础定时器开启在主循环中每一次都检测它。因此其他部分的应用不会被阻塞。

这个程序可以提供给下面三种电池类型中的一种，通过设置:

```
charger_mode = CHARGER_MODE_LION;    // 或 CHARGER_MODE_NICD 或
CHARGER_MODE_SLA
```

在不同的电池类型中，型号可以被自动的检测，通常电池电热调节器的值/排布使用图.9 的连线，图 9 所示电热调节器可能的排布；通过调节上拉/下拉电阻和使用 ADC 测量，程序可以判断电池类型。因为这里可能变化，电热调节器值的可能范围，留下读者去判断上拉/下拉电阻适当的值，和一个适当的门限使程序比较电热调节器 ADC 读取的值。

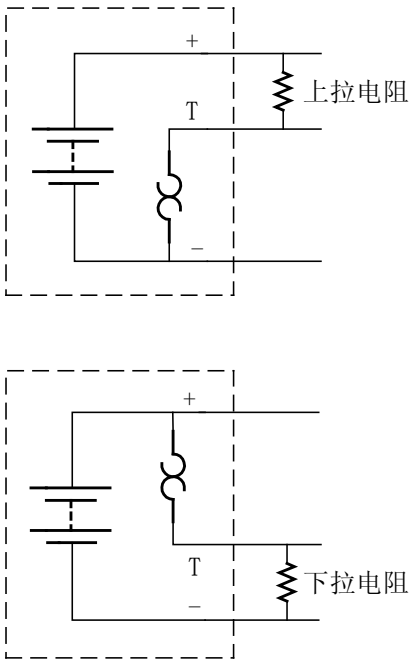


图. 9 可能的电池门限配置

下图. 10 显示一个程序流程的简单框图。在初始化后，循环执行一个普通的 housekeeping 任务，以判断充电状态。前边提到的充电器工作并不阻断，所以在显示的部分可以增加其它代码，提供在运行中不阻断充电器的代码。

下图 11、12 和 13 显示状态框图对于 3 种不同充电步骤的电池类型（i.e. SLA 具有 CC，CV 和浮动，NICD 就是 CC 和滴流）。这个框图是自说明的，并且与其它的非常相近，当时在这里有几个要注意：

1. 系统等待电池的检测，在电池终止时测量电压，并与门限电压比较。

2. 检测后，系统处理将进行处理，对于锂离子电池通过 CC-CV 状态，对于镍镉电池通过 CC-滴流状态，对于 SLA 则通过 CC-CV-浮动状态。

3. 在这些状态期间通用“housekeeping”模块监视过热、过压条件和电池移除。过热和过压条件将使系统进入“fault”状态。读者可以编写代码实现产生此条件后的动作。

4. 当电池检测后，一个长时间定时器（2 个小时）被开始，并且如果其他终止方法失败则作为一个自动防止故障。它也终止滴流和浮动充电（分别为 NiCd 和 SLA）。

5. 当电池充电循环完成时，系统在终止状态等待，直到电池被移除，然后它返回到“检测”状态。


开始
初始化CPU时钟
初始化码
初始化端口
初始化定时器
初始化 ADC
初始化 PWM
检测/充电状态结构
（详细信息请参看状态图）
用户应用码在这里可以运行
看门狗定时器清零
更新软件定时器
运行A/D转换&平均
长时期定时器监视器
电池过压监视器
电池电热调节器监视器
电池电压&电流监视器

图. 10 程序流程

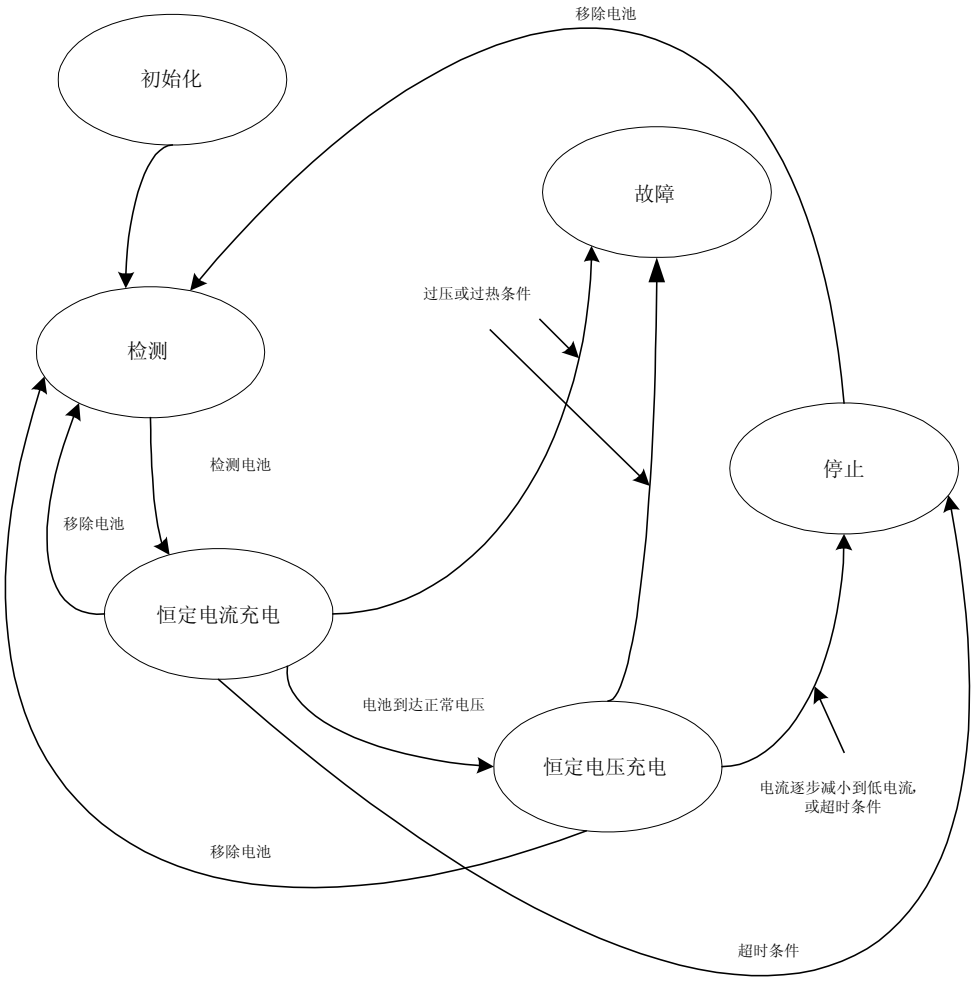


图. 11 锂离子电池充电状态图

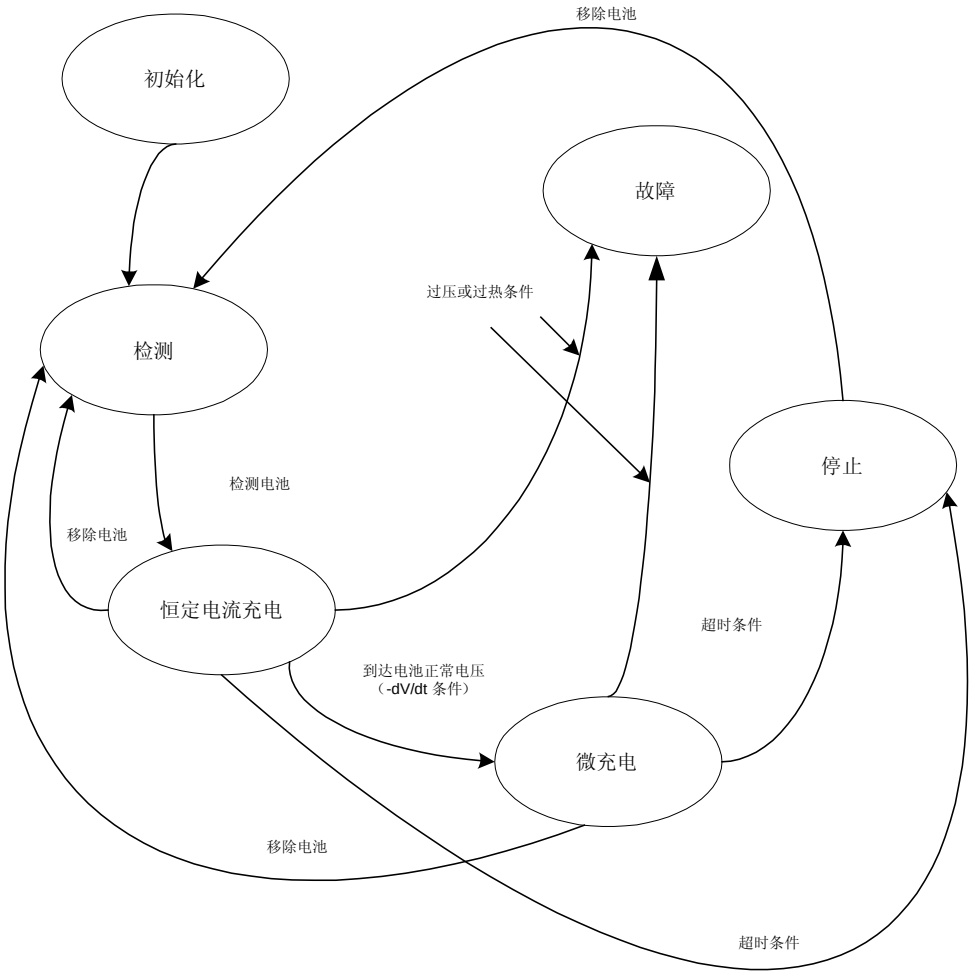


图. 12 镍镉电池充电状态图

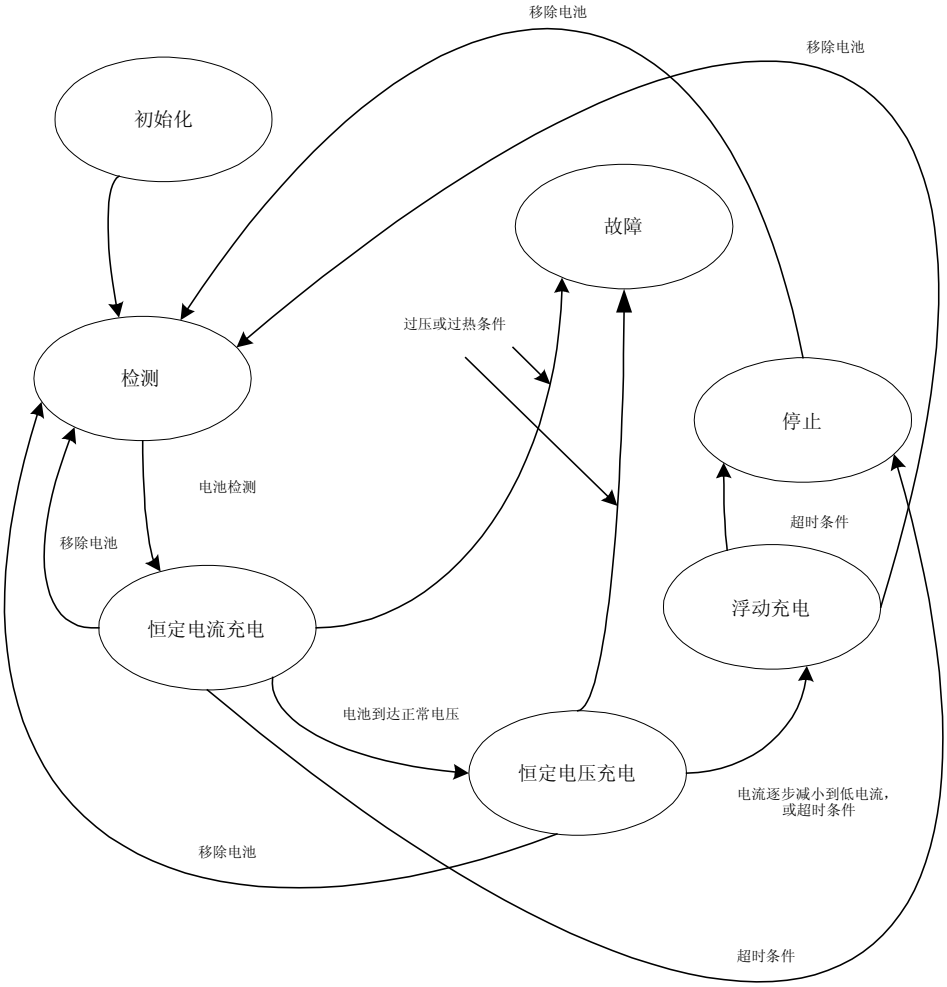


图. 13 铅酸电池充电状态图

电池监控

在电池充电时，电池电压和电流必须都被时时监视。将电池从充电器到测量电压和连接测量电流之间断开是非常有必要的，charge_enable 被用于按要求断开电池。在充电输出被关闭与使电池的终止电压稳定之间有一个短的延迟时间是非常重要的。图. 14 显示电池监视器的操作。图中显示在程序的“housekeeping”段和在充电过程中的不断持续，并且 do_adc_conversions() 功能（也连续运行）将会依靠 charge_enable 状态读取电池电压或电流。

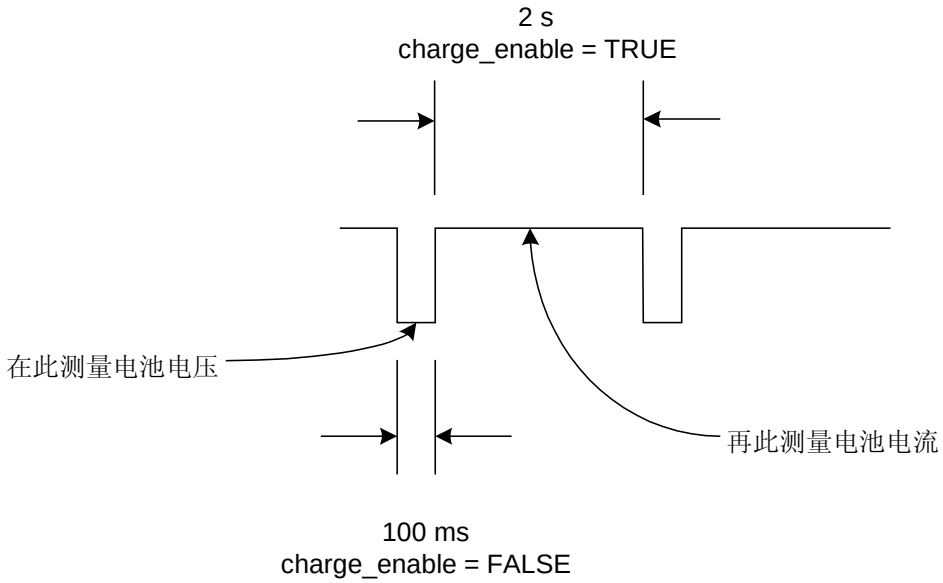


图. 14　电池监视器

比例积分控制

使用的PI调整器在电池电压和电流的封闭环路控制中是经典的比例积分（PI）控制方法。
调整器是基于递归的PI算法，以下形式：

$$G(s) = K_P + K_I \times \frac{1}{S}$$

转换为一种离散的形式：

$$K_P \cdot X_D + K_I \cdot (\Sigma X_D)$$

$$X_D = X(n) - X(n - 1)$$

这里：　K_P 表示增加的比例
　　K_I 表示增加的整数
　　X_D 表示错误
　　ΣX_D 表示累计错误

常数 K_P 和 K_I 是用经验产生的，在实践中，满意的结果包含 $K_I = 0$ ，也就是说，系统仅能有一个好的比率效率。

功能描述

void update_timers(void);	每秒100次调用8位硬件定时器TM50。减小软件定时器，并且设置其为读取的必要等待时间。
void load_timer (char timer_id, int timer_val);	在10us内调用软件定时器（最大65535）。
void start_timer (char timer_id);	启动指定的定时器。
void stop_timer (char timer_id);	停止指定的定时器。（不要复位）。
char check_timer (char timer_id);	返回：TIMER_STOPPED，TIMER_RUNNING，TIMER_TIMEOUT 或者 TIMER_IDLE 。
void reset_timer (char timer_id);	清除指定的定时器，并且设置它的状态为 TIMER_IDLE 。
void initialize_pwm (int initial_pw);	初始化微控制器PPG（PWM）模式为已通过的值。
int get_adc_value (char channel);	对一个指定的A/D转换器通道（这里使用通道0 ~ 3）进行单一的读取操作。
void write_pwm (int pwm_value);	写 pwm_value 值到PWM模式。pwm_value 最大值通过封闭的PWM分辨率（8位是255）来判断。
void initialize_adc (void);	设置 A/D 转换器。
int compute_pi (signed int setpoint, signed int actual);	基于断点（要求值）和实际值返回PI 。
void cv_cc_control (signed int setpoint, signed int parameter);	计算PWM输出。使用 compute_pi 功能。 用 write_pwm 功能限制结果和驱动PWM模式。
void do_adc_conversions (void);	对于电池电压、电流、门限值和充电器电压从主“housekeeping”循环调用、读取ADC通道。通过程序间歇对于使用的变数取平均值并保存结果。

定义

如下显示，可以单独设置电池的各个参数，使用 #difines ：

```
#define   LION_CC_CURRENT_NOMINAL              1800          // 毫安
#define   LION_CV_VOLTAGE_NOMINAL              12300         // 毫伏
#define   LION_TERMINAL_VOLTAGE_NOMINAL        10800         // 毫伏

#define   NICD_CC_CURRENT_NOMINAL              1000          // 毫安
#define   NICD_TRICKLE_CURRENT_NOMINAL200          // 毫安

#define   SLA_CC_CURRENT_NOMINAL               1500    // 毫安
#define   SLA_CV_VOLTAGE_NOMINAL               14700         // 毫伏
#define   SLA_TERMINAL_VOLTAGE_NOMINAL         12000         // 毫伏
#define   SLA_FLOAT_VOLTAGE_NOMINAL            13500         // 毫伏
```

参数使用 mV 或 mA 为单位。

以下用于检测电池门限：

```
#define   LION_DETECT_VOLTAGE_MIN              8000          // 毫伏
#define   LION_DETECT_VOLTAGE_MAX              14000         // 毫伏

#define   NICD_DETECT_VOLTAGE_MIN              8000          // 毫伏
#define   NICD_DETECT_VOLTAGE_MAX              14000         // 毫伏

#define   SLA_DETECT_VOLTAGE_MIN               8000          // 毫伏
#define   SLA_DETECT_VOLTAGE_MAX               14000         // 毫伏
```

限制很宽，并且可以缩紧，如对于一个 10.8V 的锂电池 10000mV ~ 11000mV 。充电器固件能够被更改，以对测量的不同限制的终止电压进行比较，从而对不同电压进行判断，如果这是与以前提及的电热调节器检测的组合，将会产生一个灵活的充电器。

```
#define    AVERAGING_POINTS                4
```

设置对 ADC 读取的值取平均值的点的数值。实际上，这是执行时间与读取电池参数的“smoothness”的折中。

第七章 其它充电方法

其中一个方法是单独使用微控制器为电池充电，这里简要提及一下。在多种源和简单开关模式控制器/充电电源到更多的成熟的设备特性系统管理总线、同步校正器等范围内。

使用外部 IC 的优越性包括微控制器 CPU 时间的减小；循环测量和纠正电池电压和电流将占用 CPU 很小的资源。充电器 IC 要求电压和电流通过微控制器来初始设定（通常情况下用 PWM），但是然后将会使用自己的内部模拟电路来维持封闭的循环控制。此外，充电器 IC 的 PWM 模式将会以 10 倍于微控制器的频率运行，从而可以使用小型的感应器，取得更高的效率。主要的劣势是价格；充电器 IC 明显比微控制器价格高。

这里使用的微控制器更多的是作为监控的角色，同时由于安全原因设置充电电压和电流，能够独立的监视充电器 IC 的电池电压，或者通过 LED 显示或其它一些 LCD 信息提供充电状态信息给用户。充电器 IC 并不经常通过电热调节器读取温度，所以必须通过微控制器提供此项服务，在过热时关闭电器。请参看图 15 。

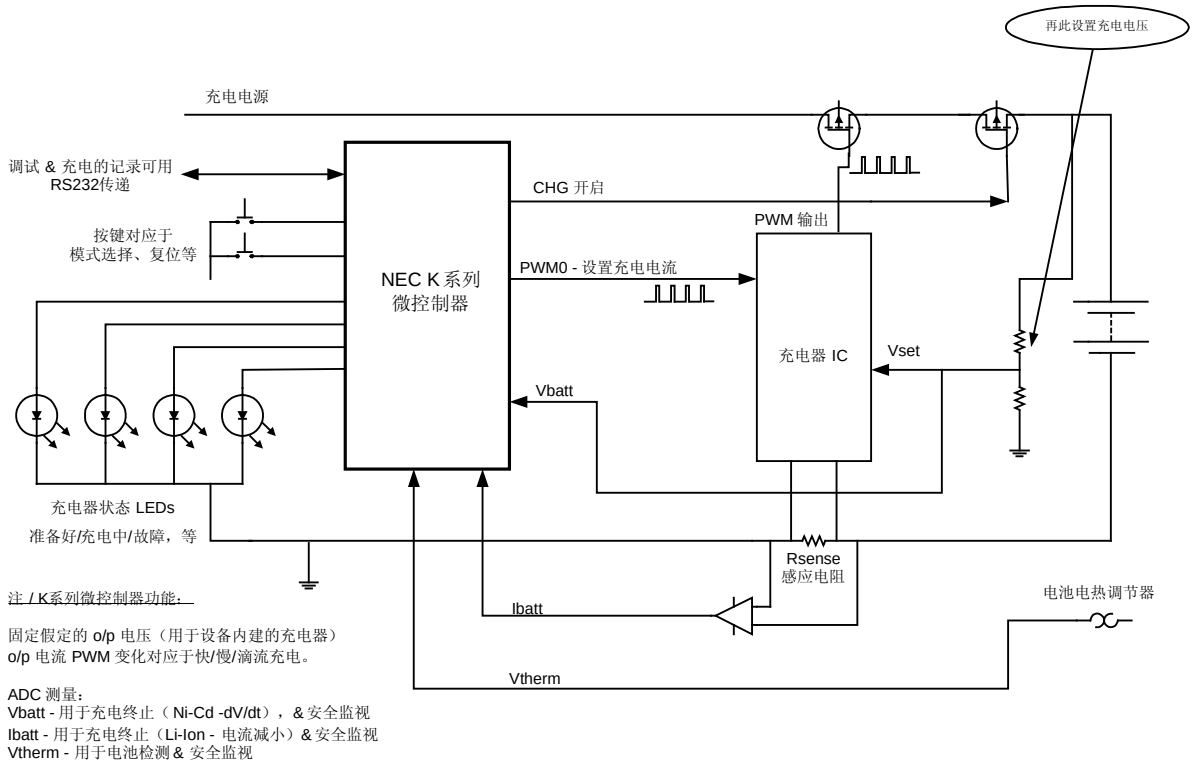


图. 15 外部充电器 IC 的使用

第八章 结论

对于这个应用使用资源如下：

2227 字节的代码存储器
209 字节的数据存储器
6 位的位存储器

所以它还剩余大量的程序空间可以为其它的应用使用。

本应用手册列举了一些在使用中的常用的可充电电池的类型和它们对于充电的要求，并且出示了 NEC 的 K 系列微控制器家族对应于这些应用的几个方法。展示了如何做或者是如何匹配用户的系统。

第九章 固件列表

```
/*============================================================================
** PROJECT = BATTCHG1
** MODULE = battchg1.c
** VERSION = V1.0
** DATE = 11.03.2004
** LAST CHANGE = 12.05.2004
**
** ============================================ ==================================
** 描述：多重化学电池充电器固件
**
** ============================================================================
** Environment:    Device: uPD78F0148
** Assembler:      A78000 Version 3.34.2.4
** C-Compiler:     ICC78000 Version 3.34.2.4
** Linker:         XLINK Version 4.55.9.0
** ============================================================================
** By: NEC Electronics (Europe) GmbH
** Cygnus House
** Sunrise Parkway
** Milton Keynes
**
** ================ ===========================================================
Changes:
** ============================================================================
*/
/* ============================================================================
** pragma
** ============================================================================
*/
#pragma language = extended
/* ============================================================================
** include
** ============================================================================
*/


#include <in78000.h>
#include <Df0148.h>

#define    TRUE                1
#define    FALSE               0
#define    INPUT               1
#define    OUTPUT                0
#define    DISABLED               1
#define    ENABLED               0


#define    MAX_TIMERS              2

#define    TIMER_STOPPED            0
#define    TIMER_RUNNING            1
#define    TIMER_TIMEOUT            2
#define    TIMER_IDLE              3

#define    AVERAGING_POINTS         4

#define    BATTERY_VOLTAGE          0   // ADS 值以选择 ADC 通道
#define    CHARGER_VOLTAGE          1   //      "
#define    BATTERY_THERMISTOR       2   //      "
#define    BATTERY_CURRENT          3   //      "

#define    BATT_V_SCL_FACTOR        16
#define    CHG_V_SCL_FACTOR         16
#define    BATT_I_SCL_FACTOR        2

#define    PWM_RESOLUTION           8
#define    XTAL_FREQ               10000000
#define    PWM_PERIOD              (1 << PWM_RESOLUTION)-1
#define    PWM_OUT_PIN              P0.1
```

```
#define    PWM_OUT_DIR              PM0.1
#define    TOC004                   TOC00.4       // 不在头文件中
#define    PWM_VAL_MIN              50            // 50 OK with 仿真器 @ 10MHz 8 bit PWM

#define    CHARGE_ENABLE            P5.0

#define    CHARGER_MODE_LION        0
#define    CHARGER_MODE_NICD        1
#define    CHARGER_MODE_SLA         2

#define    LION_DET_V_MIN           8000      // 毫伏
#define    LION_DET_V_MAX           14000     //毫伏
#define    NICD_DET_V_MIN           8000      //毫伏
#define    NICD_DET_V_MAX           14000     //毫伏
#define    SLA_DET_V_MIN            8000      //毫伏
#define    SLA_DET_V_MAX            14000     //毫伏

#define    LION_MAX_V               16000     //毫伏
#define    NICD_MAX_V               16000     //毫伏
#define    SLA_MAX_V                16000     //毫伏

#define    BATT_REMOVED_LIMIT       6000      //毫伏

#define    LION_CC_CURRENT_NOM      1800      //毫伏
#define    LION_CV_VOLTAGE_NOM      12300     //毫伏
#define    LION_TERM_VOLTAGE_NOM    10800     //毫伏

#define    NICD_CC_CURRENT_NOM      1000      // 毫安
#define    NICD_TRICKLE_CURRENT_NOM 200       // 毫安

#define    SLA_CC_CURRENT_NOM       1500      // 毫安
#define    SLA_CV_VOLTAGE_NOM       12000     // 毫伏
#define    SLA_TERMINAL_VOLTAGE_NOM 12000     // 毫伏
#define    SLA_FLOAT_VOLTAGE_NOM    11000     // 毫伏

#define    NICD_CHG_TERM_V          250       // 毫伏

// 电热调节器
#define    BATTERY_THERMISTOR_LIMIT 500

// SYSTEM STATES
#define    STARTUP_STATE            0
#define    DETECT_STATE             STARTUP_STATE + 1
#define    CC_LION_STATE            DETECT_STATE + 1
#define    CV_LION_STATE            CC_LION_STATE + 1
#define    CC_NICD_STATE            CV_LION_STATE + 1
#define    TRICKLE_NICD_STATE       CC_NICD_STATE + 1
#define    CC_SLA_STATE             TRICKLE_NICD_STATE + 1
#define    CV_SLA_STATE             CC_SLA_STATE + 1
#define    FLOAT_SLA_STATE          CV_SLA_STATE + 1
#define    FAILURE_STATE            FLOAT_SLA_STATE + 1
#define    TERMINATE_STATE          FAILURE_STATE + 1

#define    ADC_CHANNELS_USED        4

#define    BATT_MON_STATE_0         0
#define    BATT_MON_STATE_1         BATT_MON_STATE_0 + 1
#define    BATT_MON_STATE_2         BATT_MON_STATE_1 + 1


int adc_result = 0;
int data_samples[AVERAGING_POINTS + 1][ADC_CHANNELS_USED];
char sample_pointers[ADC_CHANNELS_USED];

char system_state;

char batt_mon_state;

char sub_state_1;
bit wait_for_interrupt;
int global_pwm_value = PWM_VAL_MIN;
char charger_mode;
char minute_counter;
```

```
bit charge_in_progress;
bit battery_removed;
bit safety_timer_expired;
bit charger_error;
bit over_temperature;

// PI 算法
signed int XD;          // 瞬间错误
signed int Yp;          // 比例错误
signed int Yi;          // 积分错误
signed int integrator;

const int Kp = 1;       // 比例增益常数
const int Ki = 0;       // 积分增益常数(通常非零 – 参见正文)

// 电池 & 充电器参数
int battery_voltage;
int charger_voltage;
int battery_thermistor_value;
int battery_current;
int highest_battery_voltage;


// 原型
void update_timers(void);
void load_timer (char timer_id, int timer_val);
void start_timer (char timer_id);
void stop_timer (char timer_id);
char check_timer (char timer_id);
void reset_timer (char timer_id);
void initialize_pwm (int initial_pw);
int get_adc_value (char channel);
void write_pwm (int pwm_value);
void initialize_adc (void);
int compute_pi (signed int setpoint, signed int actual);
void cv_cc_control (signed int setpoint, signed int parameter);
void do_adc_conversions (void);


typedef struct timer_struct{
    int timer_value;            // counts centiseconds
    char timer_state;
}timers;

timers timer_block[MAX_TIMERS];



void main (void){

    _DI();                      // 关闭中断

    PCC = 0x00;                 // 处理器时钟控制寄存器
                                // 振荡可能
                                // 使用片上反馈寄存器
                                // X1 输入时钟或 Ring-osc 时钟
                                // CPU 时钟（fCPU）选择 = 1:1

    OSTS = 0x05;                // 振荡稳定时间 = 2^16/Fxp = 6.55 ms @10MHz
    MOC = 0x00;                 // 启动主时钟
    while (OSTC <= 0x18){       // 等待主时钟稳定
        _NOP();
    }
    MCM0 = 1;                   // 设置 CPU 时钟 = 主时钟

    PM5.0 = 0;                  // 设置 CHARGE_ENABLE 的数据指向


                                // 初始化 8 位定时器 TM50
    TCL50 = 0x07;               // 时钟选择 fx/(2^13)
    CR50 = 12;
    TMC50 = 0x80;               // 开启定时器 50
    TMIF50 = FALSE;
```

```
    load_timer (0, 25);
    start_timer (0);

    // 确保位变量清零
    charge_in_progress = FALSE;
    battery_removed = FALSE;
    safety_timer_expired = FALSE;
    charger_error = FALSE;
    over_temperature = FALSE;

    CHARGE_ENABLE = FALSE;

    initialize_adc();

    wait_for_interrupt = FALSE;
    initialize_pwm (50);          // 初始化，将 0 给高 DC（> 99%）
    TMIF000 = FALSE;
    _EI();                        // 打开全部中断

    charger_mode = CHARGER_MODE_LION;      // 在此设置电池类型
//    charger_mode = CHARGER_MODE_NICD;
//    charger_mode = CHARGER_MODE_SLA;

    for(;;){

        WDTE = 0xAC;

        if (TMIF50){
            TMIF50 = FALSE;
            update_timers();
        }

        do_adc_conversions();

        if (charge_in_progress == TRUE){
            if (check_timer(1) == TIMER_TIMEOUT){
                load_timer (1, 6000);
                minute_counter--;
                if (minute_counter == 0)
                    safety_timer_expired = TRUE;
            }

        }

        switch (charger_mode){

            case (CHARGER_MODE_LION):{

                if (battery_voltage >= LION_MAX_V){
                    system_state = FAILURE_STATE;
                    charger_error = TRUE;
                }

                break;

            }

            case (CHARGER_MODE_NICD):{

                if (battery_voltage >= NICD_MAX_V){
                    system_state = FAILURE_STATE;
                    charger_error = TRUE;
                }

                break;

            }

            case (CHARGER_MODE_SLA):{

                if (battery_voltage >= SLA_MAX_V){
                    system_state = FAILURE_STATE;
                    charger_error = TRUE;
                }
```

```
        break;

    }

}

if (battery_thermistor_value >= BATTERY_THERMISTOR_LIMIT){
    over_temperature = TRUE;
    system_state = FAILURE_STATE;

}

switch (batt_mon_state){
    case (BATT_MON_STATE_0):{
        CHARGE_ENABLE = FALSE;
        load_timer (0, 10);
        start_timer (0);
        batt_mon_state = BATT_MON_STATE_1;
        break;
    }

    case (BATT_MON_STATE_1):{
        if (check_timer (0) == TIMER_TIMEOUT){
            if ((charge_in_progress == TRUE) && (system_state != TERMINATE_STATE) && (charger_error == FALSE))
                CHARGE_ENABLE = TRUE;       // 不要开启充电输出
                                            // 当检测电池时，
                                            // 在结束状态或错误状态

            load_timer (0, 200);
            start_timer (0);
            batt_mon_state = BATT_MON_STATE_2;
        }
        break;
    }

    case (BATT_MON_STATE_2):{
        if (check_timer (0) == TIMER_TIMEOUT){
            batt_mon_state = BATT_MON_STATE_0;
        }
        break;
    }

}

switch (system_state){

    case (STARTUP_STATE):{

            system_state = DETECT_STATE;

        break;

    }

    case (DETECT_STATE):{

        switch (charger_mode){

            case (CHARGER_MODE_LION):{
                if ((battery_voltage >= LION_DET_V_MIN) && (battery_voltage <= LION_DET_V_MAX)){
                    CHARGE_ENABLE = TRUE;
                    charge_in_progress = TRUE;
                    minute_counter = 120;           // 2 个小时
                    load_timer (1, 6000);           // 1 分钟时期
                    start_timer (1);
                    system_state = CC_LION_STATE;

                }

                break;

            }

            case (CHARGER_MODE_NICD):{
```

```
            if ((battery_voltage >= NICD_DET_V_MIN) && (battery_voltage <= NICD_DET_V_MAX)){
                highest_battery_voltage = 2000; // initial (low) value
                CHARGE_ENABLE = TRUE;
                charge_in_progress = TRUE;
                minute_counter = 120;          //  2 个小时
                load_timer (1, 6000);          // 1 分钟时期
                start_timer (1);
                system_state = CC_NICD_STATE;

            }

            break;

        }

        case (CHARGER_MODE_SLA):{
            if ((battery_voltage >= SLA_DET_V_MIN) && (battery_voltage <= SLA_DET_V_MAX)){
                CHARGE_ENABLE = TRUE;
                charge_in_progress = TRUE;
                minute_counter = 120;          // 2 个小时
                load_timer (1, 6000);          // 1 分钟时期
                start_timer (1);
                system_state = CC_SLA_STATE;

            }

            break;

        }

    }

    break;

}

case (CC_LION_STATE):{

    if (charge_in_progress == FALSE)           // 移除电池
        system_state = DETECT_STATE;
    else if (safety_timer_expired == TRUE){
        safety_timer_expired = FALSE;
        system_state = TERMINATE_STATE;
        reset_timer (1);
    }
    else if (battery_voltage >= LION_TERM_VOLTAGE_NOM)
            system_state = CV_LION_STATE;      // 准备好恒定电压充电
    else
        cv_cc_control (LION_CC_CURRENT_NOM, battery_current);


    break;

}

case (CV_LION_STATE):{

    if (charge_in_progress == FALSE)           // 移除电池
        system_state = DETECT_STATE;
    else if (safety_timer_expired == TRUE){
        safety_timer_expired = FALSE;
        system_state = TERMINATE_STATE;
        reset_timer (1);
    }
    else if (battery_current <= (LION_CC_CURRENT_NOM / 10)){
            system_state = TERMINATE_STATE;    // 电流减小，电池充电

    }
    else
        cv_cc_control (LION_CV_VOLTAGE_NOM, battery_voltage);   // 应用恒定电压


    break;
```

```
        }

        case (CC_NICD_STATE):{

            if (charge_in_progress == FALSE)              // 移除电池
                system_state = DETECT_STATE;
            else if (safety_timer_expired == TRUE){
                safety_timer_expired = FALSE;
                system_state = TERMINATE_STATE;
                reset_timer (1);
            }
            else if ((battery_voltage < (highest_battery_voltage - NICD_CHG_TERM_V)) && (battery_voltage >
(highest_battery_voltage - (2 * NICD_CHG_TERM_V))))
                    system_state = TRICKLE_NICD_STATE;     // 准备好滴流充电
            else{
                cv_cc_control (NICD_CC_CURRENT_NOM, battery_current);
                if (battery_voltage > highest_battery_voltage)
                    highest_battery_voltage = battery_voltage;
            }

            break;

        }

        case (TRICKLE_NICD_STATE):{

            if (charge_in_progress == FALSE)              // 移除电池
                system_state = DETECT_STATE;
            else if (safety_timer_expired == TRUE){
                safety_timer_expired = FALSE;
                system_state = TERMINATE_STATE;
                reset_timer (1);
            }
            else
                cv_cc_control (NICD_TRICKLE_CURRENT_NOM, battery_current);

            break;

        }

        case (CC_SLA_STATE):{

            if (charge_in_progress == FALSE)              // 移除电池
                system_state = DETECT_STATE;
            else if (safety_timer_expired == TRUE){
                safety_timer_expired = FALSE;
                system_state = TERMINATE_STATE;
                reset_timer (1);
            }
            else if (battery_voltage >= SLA_TERMINAL_VOLTAGE_NOM)
                    system_state = CV_SLA_STATE;          // 准备好恒定电压充电
            else
                cv_cc_control (SLA_CC_CURRENT_NOM, battery_current);

            break;

        }

        case (CV_SLA_STATE):{

            if (charge_in_progress == FALSE)              // 移除电池
                system_state = DETECT_STATE;
            else if (safety_timer_expired == TRUE){
                safety_timer_expired = FALSE;
                system_state = TERMINATE_STATE;
                reset_timer (1);
            }
            else if (battery_current <= (SLA_CC_CURRENT_NOM / 50)){
                    system_state = FLOAT_SLA_STATE;       // 电流减小，电池充电，准备好浮动电压

            }
            else
                cv_cc_control (SLA_CV_VOLTAGE_NOM, battery_voltage);   // 应用恒定电压

            break;
```

```
        }

        case (FLOAT_SLA_STATE):{

            if (charge_in_progress == FALSE)            // 移除电池
                system_state = DETECT_STATE;
            else if (safety_timer_expired == TRUE){
                safety_timer_expired = FALSE;
                system_state = TERMINATE_STATE;
                reset_timer (1);
            }
            else
                cv_cc_control (SLA_FLOAT_VOLTAGE_NOM, battery_voltage); // 应用恒定电压

            break;

        }

        case (TERMINATE_STATE):{

            CHARGE_ENABLE = FALSE;                  // 关闭充电器输出

            if (charge_in_progress == FALSE){           // 等待移除电池
                system_state = DETECT_STATE;

            }
            break;

        }

        case (FAILURE_STATE):{

            CHARGE_ENABLE = FALSE;

            break;

        }

      }

   }

}

// ===========================================================================

void update_timers (void){

    char loop;

    for (loop = 0; loop < MAX_TIMERS; loop++){
        if (timer_block[loop].timer_value){     // 如果定时器的值不是 0
            if (!--timer_block[loop].timer_value)
                timer_block[loop].timer_state = TIMER_TIMEOUT;
        }
    }
}

// ===========================================================================

void load_timer (char timer_id, int timer_val){

    timer_block[timer_id].timer_value = timer_val;
    timer_block[timer_id].timer_state = TIMER_RUNNING;

}

// ===========================================================================

char check_timer (char timer_id){

        return (timer_block[timer_id].timer_state);

}
```

```
// ================================================================================

void start_timer (char timer_id){

    timer_block[timer_id].timer_state = TIMER_RUNNING;

}

// ================================================================================

void reset_timer (char timer_id){

    timer_block[timer_id].timer_state = TIMER_IDLE;
    timer_block[timer_id].timer_value = 0;

}

// ================================================================================

void initialize_pwm (int initial_pw){

    PWM_OUT_PIN = FALSE;        // 设置 PWM 输出
    PWM_OUT_DIR = OUTPUT;

    TMC00 = 0x00;               // 停止定时器 TM00
    PRM00 = 0x00;               // 1:1 预分频 TM00
                        // *** 不能使 DCs < 50 作为 PWM 工作的太快 ***
    CRC00 = 0x00;               // CR000 作为比较寄存器工作
    CR000 = PWM_PERIOD;
    CR010 = initial_pw;         // 设置 PWM 占空比
    TOC00.0 = 1;
    TOC00 |= 0x1B;
    TOC00 &= 0x13;               // 当 CR000 和 TM00 相等时，开启反向
                        // 定时器输出 F/F 设置（1）
                        // 当 CR010 和 TM00 相等时，开启反向

    TMC00 = 0x0c;               // 16 位定时器模式控制寄存器
                        // 当 TMC0n2 和 TMC0n3 被设置时，16 位定时器计数器开始工作
                        // 当 TM00 和 CR000 相等时，产生清零和启动
                        // 当 TM00 和 CR000 相等时，或 TM00 和 CR010 相等时产生反向


}

// ========================================================================================

interrupt [INTTM000_vect] void timer00 (void){

    TMC00 = 0x00;
    wait_for_interrupt = FALSE;
    CR010 = global_pwm_value;
    TMC00 = 0x0C;

}

// ============================================================================

int get_adc_value (char channel){

    int adc_result;

    ADS = channel;                  // 选择 ADC 通道

    ADCS = TRUE;                     // 开始转换
    while (!ADIF)                   // 等待 EOC
        ;
    adc_result = (ADCR >> 6);
    ADCS = FALSE;                    // 停止继续转换
    ADIF = FALSE;                   // 清除中断标志

    return (adc_result);
```

```
}

// ================================================================================

void write_pwm (int pwm_value){

    global_pwm_value = pwm_value;
    TMIF000 = FALSE;
    TMMK000 = ENABLED;              // 在此时期结束时，载入 PWM
    wait_for_interrupt = TRUE;
    while (wait_for_interrupt == TRUE)
        WDTE = 0xAC;
    TMMK000 = DISABLED;

}

// ================================================================================

void initialize_adc (void){

    ADM = 0x38;          // ADC 模式寄存器
    ADS = 0x02;          // 模拟输入通道指定寄存器
    PFM = 0x00;          // 掉电比较模式寄存器 – POR 的值
    PFT = 0x00;          // 掉电比较门限寄存器 - POR 的值
    ADIF = FALSE;

}

// ================================================================================

int compute_pi (int setpoint, int actual){

    XD = setpoint - actual;

    Yp = Kp * XD;

    integrator += XD;

    Yi = Ki * integrator;

    return (Yp + Yi);

}

// ================================================================================

void cv_cc_control (signed int setpoint, signed int parameter){

    signed int temp;

    temp = compute_pi(setpoint, parameter);

    if (temp > 0)
        global_pwm_value++;
    else if (temp < 0)
        global_pwm_value--;

    if (global_pwm_value <= PWM_VAL_MIN)
        global_pwm_value = PWM_VAL_MIN;
    else if (global_pwm_value >= 254)
        global_pwm_value = 254;

    write_pwm (global_pwm_value);

}

// ================================================================================

void do_adc_conversions (void){

    char loop_1;
    char loop_2;
    int total;
    static char pointer;
```

```
    if (CHARGE_ENABLE == TRUE)  // 如果开启充电器输出，则测量电池电流...
       data_samples[pointer][BATTERY_CURRENT] = get_adc_value(BATTERY_CURRENT);
    else                        // ...否则测量电池电压
       data_samples[pointer][BATTERY_VOLTAGE] = get_adc_value(BATTERY_VOLTAGE);

    data_samples[pointer][CHARGER_VOLTAGE] = get_adc_value(CHARGER_VOLTAGE);
    data_samples[pointer][BATTERY_THERMISTOR] = get_adc_value(BATTERY_THERMISTOR);

    if (++pointer >= AVERAGING_POINTS)
       pointer = 0;

    for (loop_1 = 0; loop_1 < ADC_CHANNELS_USED; loop_1++){
       total = 0;
       for (loop_2 = 0; loop_2 < AVERAGING_POINTS; loop_2++){
          total += data_samples[loop_2][loop_1];
       }
       total /= AVERAGING_POINTS;
       data_samples [AVERAGING_POINTS][loop_1] = total;
    }

    battery_voltage = data_samples [AVERAGING_POINTS][BATTERY_VOLTAGE] * BATT_V_SCL_FACTOR;
    charger_voltage = data_samples [AVERAGING_POINTS][CHARGER_VOLTAGE] * CHG_V_SCL_FACTOR;
    battery_thermistor_value = data_samples [AVERAGING_POINTS][BATTERY_THERMISTOR];
    battery_current = data_samples [AVERAGING_POINTS][BATTERY_CURRENT] * BATT_I_SCL_FACTOR;

    if ((charge_in_progress == TRUE) && (battery_voltage < BATT_REMOVED_LIMIT)){
       charge_in_progress = FALSE;        // 然后移除电池
       CHARGE_ENABLE = FALSE;             // 关闭充电器
       reset_timer (1);
    }

}

// ================================================================================
// ================================================================================
```

［备忘录］

NEC

传真信息

来自：

姓名：

公司：

电话：　　传真：

地址：

虽然NEC已经采取各种可行措施以确保供给客户完整的，无误的以及最新的文档，但我们也愿意接受可能出现的错误。尽管我们已经采取谨慎态度和预防措施，但你还有可能遇到文档问题，如果要向我们举报错误或提出建议，请完成这个表格。

感谢你的支持。

[北京]
日电电子（中国）有限公司
中国北京市海淀区知春路 27 号
量子芯座 7，8，9，15 层
电话：（+86）10-8235-1155
传真：（+86）10-8235-7679

[深圳]
日电电子（中国）有限公司深圳分公司
深圳市福田区益田路卓越时代广场大厦 39 楼 3901，3902，3909 室
电话：（+86）755-8282-9800
传 真 ：（ +86 ） 755-8282-9899

上海恩益禧电子国际贸易有限公司
中国上海市浦东新区银城中路 200 号
中银大厦 2511-2512 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

[上海]
日电电子（中国）有限公司上 海分公司
中国上海市浦东新区银城中路 200 号
中银大厦 2409-2412 和 2509- 2510 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

[香港]
香港日电电子有限公司
香港九龙旺角太子道西 193 号新世纪广场第 2 座 16 楼 1601-1613 室
电话：（+852）2886-9318
传真：（+852）2886-9022
2886-9044

我想举报错误/提出如下意见：

文档标题：

文档号：　　页码：

如果有可能，请传真引用页及图片。

文档等级	很好	较好	可接受	不好
清晰度	□	□	□	□
技术准确度	□	□	□	□
结构	□	□	□	□

CS 99．1

［备忘录］