针对传感节点的基于 signal-slot 电源管理方案设计与实现

李洪刚 李 烨 任国稳

（中国科学院深圳先进技术研究院生物医学信息技术研究中心 深圳 518055）

摘 要 随着传感网络和 3G 网络的融合，物联网已经成为新世纪最重要的技术之一，如何延长传感节点的工作时间已成为物联网研究的一个重要课题。传统的电源管理规范如 APM（Advanced Power Management）和 ACPI（Advanced Configuration and Power Interface）主要针对 PC 设计，因其复杂性和对 BIOS 层要求等因素，在无线传感节点中并不适用。为了解决此问题，针对传感节点计算和存储能力有限的特点，我们首先开发了精简的 signal-slot 框架，基于 signal-slot 框架，并设计了简单有效的电源管理方案 SPM（Simple Power Management），并将 SPM 在流行的传感节点操作系统 Contiki 中实现。

关键词 signal-slot；电源管理；传感节点；Contiki

中图分类号 TP 316 **文献标志码** A

The Design and Implementation of Power Management Scheme Based on Signal-Slot for Sensor Node

LI Honggang LI Ye REN Guowen

[3](*Research Center for Biomedical Information Technology*, *Shenzhen Institutes of Advanced Technology*, *Chinese Academy of Sciences*, *Shenzhen* 518055, *China*)

Abstract With the integration of 3G Network and Wireless Sensor Network, the Internet of Things is considered to be one of the most important technologies of the new century. For the Internet of Things, how to extend the working time of sensor nodes is the core problem. At present, there are two specifications of power management: APM (Advanced Power Management) and ACPI (Advanced Configuration and Power Interface) which are designed for PC. Due to their complexity and the requirements for BIOS layer, these two methods are not applicable to wireless sensor node. In this paper, the signal-slot framework was developed firstly. Secondly, based on signal-slot framework, a simple and effective power management scheme-SPM (simple power management) was designed and applied in the popular sensor node operating system Contiki.

Keywords signal-slot; power management; sensor node; Contiki

收稿日期：2014-3-17

基金项目：国家自然科学基金资助项目（81101127）。

作者简介：李洪刚，博士研究生，研究方向为传感网络、操作系统和移动计算；李烨（通讯作者），博士生导师，研究方向为移动医疗、健康物联网和绿色无线通信，E-mail：ye.li@siat.ac.cn.；任国稳，硕士，研究方向为网络通信和无线传感器网络。

1 引 言

物联网近年来发展迅猛。作为数据提供源的传感节点，在物联网中扮演着重要角色，但其受限于有限的电池电量和电池不易更换的特点。如何在满足诸如 deadline 约束的情况下延长电池工作时间，成为一个重要的研究课题[1]。为了延长传感节点的电池寿命，软硬件厂商们不断尝试开发新的节电技术。简单地说，这些节电技术可以分为两类：静态技术和动态技术。

1.1 静态技术

静态技术就是根据系统负载进行性能调节，这种调节不是针对运行任务的，主要针对外设和 CPU，通过停止某些模块的时钟和电源供应将能耗降至最低。嵌入式系统处理器一般支持睡眠（Sleep）、空闲（Idle）和运行（Run）等模式，每种模式的功耗不同。系统可以根据实际情况进行转换，在运行状态下，设备全部正常工作，而在睡眠与空闲模式下，处理器则可以按照特定的模式进行相应的节能。

1.2 动态技术

动态技术是根据芯片所运行的应用程序对计算能力的不同需要，动态调节芯片的运行频率和电压（对于同一芯片，频率越高，需要的电压也越高），从而达到节能的目的。从 1996 年第一篇有关电压动态论文的发表，到 1998 年实验性平台的建立[2]，动态电压调节日趋成熟，目前大多数 CPU 提供了动态电压调节功能 DVFS（Dynamic Voltage and Frequency Scaling）来支持动态节能技术。DVFS 即动态电压频率调整，目前许多芯片支持 DVFS，比如 Intel 公司的芯片支持 SpeedStep，ARM 的支持 IEM（Intelligent Energy Manager）和 AVS（Adaptive Voltage Scaling）等。由于 DVFS 技术本身的复杂性，在实现方式上主要分为硬件实现和软件实现[3]。

1.2.1 硬件实现

在硬件实现中，CPU 负载跟踪与性能预测由硬件来实现。这样一方面增强了负载计算的准确性，另一方面也减轻了软件用于负载跟踪与性能预测的复杂性和实现难度。当然，这样做也有一个弊端，就是无法灵活选择预测算法，Intel imx31 的调频可由硬件自动处理，CPU 通过检测温度和预测系统的负载进而调压调频。

1.2.2 软件实现

在这种方式下，内核会将极端时间内的 15 次负载采样返回到用户空间，此时用户空间的程序需要通过一定的算法为 CPU 选择一个合适的频率。值得注意的是，这 15 次负载采样完全基于硬件寄存器（load track）并通过 dma 上传到用户空间的 buf 中。这说明这是一个极端时间内的负载采样值，用户层的程序需要关注的是这一段时间内的采样规律，进而动态调整 CPU 的频率[4, 5]。

为了将上述节电技术应用到实际当中，APM（Advanced Power Management）[6]、ACPI（Advanced Configuration and Power Interface）[7] 等电源管理规范被提出。

APM：APM 模型将电源管理几乎完全分配给 BIOS 控制，这样大大限制了操作系统在控制电能消耗方面的功能，仅提供机器悬挂（Suspend）或备用（Standby）状态以及检查电池容量等功能。也就是说操作系统层仅有这些功能，它是通过 APM 接口实现对硬件层的功耗控制的。

ACPI：1997 年因特尔、微软和东芝三家公司共同制定了一个电源管理标准，即 ACPI。

无线传感节点具有如下特点：

硬件多样性：传感网络硬件平台各不相同，有的 CPU 支持动态电源管理，有的则仅支持有限的 CPU 工作模式（Sleep、Wake 等状态）。

软件多样性：采用的操作系统有 Tinyos 系列、Contiki、SOS 等，并且有不少无操作系统。

APM、ACPI 这些传统的电源管理方案是针

对一般通用的操作系统设计的，无法直接应用于无线传感节点当中。

综上所述，设计一种能兼顾无线传感网络各种操作系统和硬件差别的电源管理解决方案将能给相关领域研究带来一定研究意义。

1.3　SPM 方案

我们设计的方案主要包括以下：

(1) 设计针对无线传感网络节点特点的信号槽 signal-slot 机制；

(2) 基于 signal-slot 设计精简电源管理方案 SPM(Simple Power Management)；

(3) 将 SPM 在无线传感网络操作系统 Contiki 中进行实现。

1.3.1 signal-slot

Signal-slot 信号槽这一术语最初来自 Trolltech 公司 Qt 库[8](现在已经被 Nokia 收购)。1994 年，Qt 的第一个版本发布，为我们带来了信号槽的概念。这一概念立刻引起计算机科学界的注意，提出了多种不同的实现。如今信号槽依然是 Qt 库的核心之一，其他许多库也提供了类似的实现，甚至出现了一些专门提供这一机制的工具库。

信号槽机制实际上是设计模式中观察者模式的一种变形，它是面向组件编程的一种很强大的工具。

Signal-slot 能够降低组件之间的耦合程度，这对电源管理方案有十分重要的意义。比如，当 CPU 要进入 sleep 状态时，仅需要触发一个 sleep 事件信号就可以完成现场保存，相应设备休眠等操作。相对于传统的调用不同函数来完成状态切换，信号槽方式要稳定方便许多。但其最大的缺点在于要稍微牺牲一点性能。根据 Trolltech 公司的自测，在 CPU 为 500 MHz 的设备上，对于一个信号对应一个反应槽的连接来说，一秒钟可以调用两百万次；对于一个信号对应两个反应槽的连接来说，一秒钟可以调用一百二十万次。在实现方式上，因为需要解决的问题不同，目前 signal-slot 实现有如下三种：

(1) QT；

(2) gnome 桌面环境的 gobject ；

(3) dbus 中的 signals。

上述实现方式为了应对 GUI 的各种操作，实现时定义了各种字段，而这些字段对于传感网络的电源管理来说太过于冗长。并且为了编程的健状性，通常都加入了类型检测等手段，这些在传感网络节点中是不需要的。所以，我们需要根据实际情况，自己设计与实现针对传感网络特点的 signal-slot。

论文的其他部分组织如下：第 2 部分详细描述了 signal-slot 实现，第 3 部分提出了基于 signal-slot 针对无线传感网络的电源管理方案 SPM，第 4 部分给出了 SPM 在无线传感操作系统 Contiki 中的实现，第 5 部分给出了实验结果，论文的结论部分在第 6 部分描述。

2　Signal-slot 方案设计

Signal-slot 机制顾名思义就是一个发送和接收信号的系统。通常描述为 signal 是一个发布者，slot 可以是一个或多个订阅者，这些 slots 作为回调者连接到 signal 上，在 signal 发出(emit)的时候会自动被调用。一般情况下，一个 signal 何时会发生是不可预知的，至少在建立 slots 连接时是如此，这非常适合用于在逻辑上独立性很强的代码间建立通讯。

我们设计的 signal-slot 提供如下接口：

(1) 注册事件：

Connect(char *signal, void *handler, int priority)

(2) 断开事件：

Disconnect(char *signal,void *handler)

(3) 触发事件：

Emit(char *signal, struct list_head **pParamHead)

在 signle-slot 的设计上，我们采用全局的 hash hash_list[HASH_LEN]，每一个 signal 在 hash 占一项，其 value 代表处理函数形成的链表，通过 priority 字段支持函数按优先级调用。

$$K=\sum_{i=0}^{N} Ascii(i), H(K)=K\%L \quad (1)$$

公式(1)给出了 signal-slot 所用的 hash 函数实现过程，N 代表 signal 的字节总数，K 代表 ASCII 码之和，L 是 hash 数组长度。通过 $H(K)$ 计算相应的 signal 的存储位置。无线传感网络节点电源事件有限，通常的时间复杂度为 O(1)。

3 SPM 方案设计

在前几部分，我们给出了 signal-slot 的接口及其实现，在下面部分，我们将针对无线传感网络中十分流行的操作系统 Contiki 设计一种精简的电源管理方案 SPM。SPM 在设计上参考 APM，但对其进行了很大改动。SPM 采用 signal-slot 作为整个体系结构的基石，并把调度策略放到应用层实现。在接口规范上，提供事件注册、断开、触发接口，这使开发工作变得异常简单。在实现方式上，SPM 结合 Contiki 调度的特点，把 Time-Out 的实现放在 IDLE 状态中，这样能使和输入输出无关的后台进程得到有效执行。作为 Time-Out 的补充，在系统初始化阶段，建立监测电源电量的后台进程。SPM 在整个系统结构上，采用类 APM 的三层体系结构：Bootloader 层、kernel 及 driver 层和应用层，其体系结构如图 1 所示。

3.1 SPM 调度策略

电源管理策略的研究一直是电源管理规范

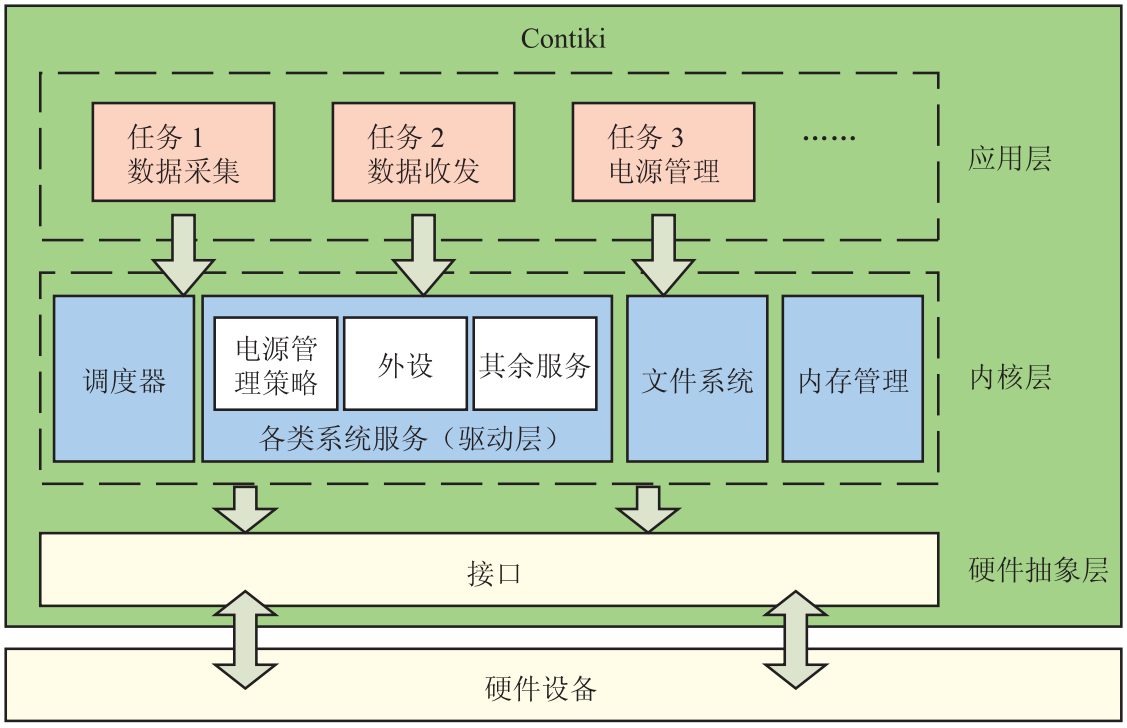

图 1 SPM 框架图

Fig. 1. The SPM architecture

的重点[9-12]，在我们的 SPM 电源管理方案中，针对无线传感节点有限的计算能力和资源，采用 Time-Out 超时机制，主要实现电源管理中的静态管理方法；针对 CPU 工作状态采用 Time-Out 机制进行管理；基于开发的 signal-slot 机制易于在不同的软硬件环境中实现，在稳定性和可控性上都能得到保证。对于 Time-Out 的超时时间，分为自适应和非自适应两种。非自适应是对超时时间 t 设定的固定值，而自适应的 Time-Out 中的超时 t 是由系统自行调节的。在 SPM 中，采用非自适应算法，超时时间 t 由用户程序自行设定，如果用户没有设定，则采用默认值(默认值为 60 秒)。在 Time-Out 调度策略中，有两点是关键：一是超时值的设定，二是超时计时起点的设定。在一些系统实现中，只是简单根据外部按键或触摸屏事件来计时，只要在一定时间内没有按键，系统就进入到另一个功耗状态。这样做有明显的不足之处：有程序(进程或后台进程)运行时，如果没有按键或触摸屏事件，也要进入另一个状态，这会使程序不能得到有效执行任务(比如在嵌入式系统中常用的 Flash 文件系统 JFFS，需要有后台线程进行碎片收集整理工作)。所以在 SPM 中，将定计时的起点设在系统进入 IDLE 空闲状态中，当系统进入此进程时，代表已经没有进程运行了，这样就能使电源管理和进程的运行有效分开，互不影响。超时时间分为 3 个超时值 T1、T2 和 T3。超时策略的运行情况如图 2 所示。

当系统进入 IDLE 状态，代表系统已经没有其他进程在就绪态，此时 T1 开始计时，如果此时有新的进程执行或事件发生，就把计时值清零，让其重新开始。当 T1 计时达到设定值时，系统进入空闲状态，此时 T2 开始计时，如果在 T2 达到设定值前有进程执行或事件发生，那么系统返回到正常态运行。当 T2 达到设定值时，系统进入低速态运行，T3 开始计时，如果在 T3

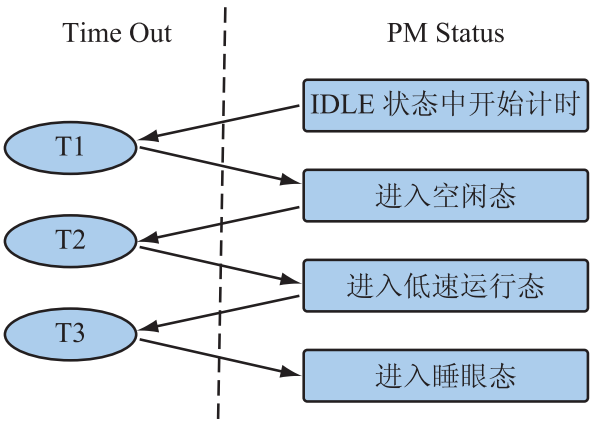

图 2　Time-Out 策略运行示意图

Fig. 2.　The time-out policy specification using sequence diagram

达到设定值前有进程就绪或事件发生，此时系统返回到正常态。而当 T3 达到设定值时，系统进入睡眠态。

这样的调度策略通过设定计时的开始点，有效避免了仅靠判断一定时间内外部事件是否发生来决定是否进入下一个状态所带来的问题，但也有一个不足之处，就是要使系统进入 IDLE 状态，才能使整个 Time-Out 机制发生作用。如果系统没有进入 IDLE 状态运行，那么整个 Time-Out 就不能生效，为了弥补这一不足，在 SPM 中设定了后台进程来实现电源的管理，我们将在下一章节介绍电源管理的后台进程实现。

3.2　后台进程实现电源管理

此功能是对 Time-Out 机制的一个扩充，目的是实时监视电源的电量情况，因为在 Time-Out 的实现要依靠 IDLE 状态，如果用户程序进不了 IDLE 状态，而这时电源电量又比较低，就会使系统的电量很快耗尽。如果有一个内核进程实时监控电源电量，就可避免这种情况的发生。在 SPM 体系中，在系统初始化阶段，会建立后台进程，此进程每隔一定时间查询电量状态，当系统电量小于设定值时，动态地调整 CPU 的工作频率，以延长系统的工作时间。

3.3 电源状态的设计

针对无线传感计算能力和资源受限等特点，SPM 将电源管理默认分为四种状态，同时得益于 signal-slot 灵活的体系结构，电源状态可以根据不同的软硬件进行相应扩展。

(1)运行态：运行态是传感的正常操作状态，此时所有的电源供给处于工作状态，所有的时钟处于运行中，每一种片上资源都是可用的，可以通过设置使相应的外设省电。

(2)空闲态：整个系统处于无事可做状态，这时将 CPU 断电，相应外设正常工作。当有中断请示时 CPU 恢复供电，正常工作。

(3)低频态：此时系统的供电电量不足，CPU 进入一个频率较低的状态下工作。

(4)睡眠态：整个系统睡眠，DRAM 进入自刷新状态。

4 SPM 方案实现

在上一章中，我们设计了整个电源管理方案，而在本章中，将根据上一章的设计来将电源管理 SPM 在流行的传感节点操作系统 Contiki 中实现。在本章中，还将电源状态的改变从空闲态－Susepnd 睡眠－Bootloader 操作 DRAM 自刷新恢复－Resume 唤醒的线索进行展示描述，我们也将以此为主线来组织本章内容。

4.1 Time-Out 调度策略实现

在简单电源管理策略 SPM 中，我们采用 Time-out 机制来实现电源管理的智能化。首先我们分析 Contiki 内核的调度机制，然后根据 Contiki 内核调度的特点实现 Time-Out 机制。

Contiki 内核调度分析：本文的目的不是专门分析 Contiki 进程的管理，所以本节仅分析与 SPM 相关的部分。Contiki 系统在初始化完成之后，会进入死循环执行 process_run 函数，在 process_run 中处理系统所有 needspoll 标记为 1 的进程及处理事件队列的下一个事件，程序代码如下：

```
int main()
{
dbg_setup_uart();
usart_puts("Initialising\n");
clock_init();
process_init();
process_start(&etimer_process,NULL);
autostart_start(autostart_processes);
do{
    If(spm_count++>设定值){
//执行此处代表系统进入 IDLE 状态，在此处执行电源管理
    SPM_IDLE()
}
}while(process_run()>0);
return0;
}
```

我们增加全局变量 spm_count，在 process_run 中，如果有进程或事件需要处理，表明系统处于激活状态，将 spm_count 清零。当 spm_count 大于设定值时，表明系统没有进程和事件需要处理时，调用 SPM_IDLE 函数，我们在 SPM_IDLE 实现 Time-out 策略。Time-Out 调度策略的目的，就是当系统无事可做时，让其进入我们设定的各种省电模式。实现程序伪码表示如下：

```
static inline void SPM_IDLE(void)
{
    if (达到用户设定的值进入低功能)
            进入低功耗函数 IDLE_Mode;
}
void IDLE_Mode(void)
{
    使 CPU 断电;
    Emit(“PM”,”0”);
    触发电源管理事件，将空闲外设进入省
```

电模式
当有中断触发时，恢复CPU供电;
}

至此，整个系统的 Time-Out 全部实现了。SPM 的调度策略发生点是在系统的 IDLE 状态当中，要达到这一目的就要求开发人员在写相关外设驱动的时候一定要写成支持阻塞调用的驱动程序，应用程序在调用驱动程序的时候也要用阻塞方式调用。如果写成轮询方式的驱动程序，那么相关进程一直在轮询相关的状态，这时进程还一直处于运行态，IDLE 状态就无法得到调用，SPM 也就无从谈起。同样的道理，即使驱动阻塞方式，如果在应用程序层轮询，IDLE 状态也没办法得到调用，SPM 机制也无法实现。

4.2　Suspend 实现

Suspend 使整个系统进入睡眠状态，各个外设都停止工作，此时功耗相对于其他模式最低。进入 Suspend 过程大致如下：内核收到要求睡眠的请求，内核在当前允许的情况下，设置唤醒中断源，调用睡眠函数，寄存器内容保存到内存后，动态内存自动刷新，设备切换到低频或低压，系统进入睡眠状态，各种寄存器内容丢失。当接收到外部中断，且该中断是唤醒源（比如按键），网络中断，系统进入唤醒 RESET，重新执行 Boot Loader 处的程序，恢复现场，跳转恢复程序，把寄存器等内容恢复，继续执行原来的程序。可见进入睡眠要完成如下的主要工作：

(1)通知外设：通知外设模块，关闭相应模块供电，需要保存现场的，进行现场保护，在唤醒再调用唤醒函数将其恢复。

(2)保存现场：由于 CPU 进入睡眠状态以后，各个寄存器内容要丢失，所以要将其保存到一定存储区域。在嵌入式系统中，因为硬盘少有这样的存储设备，所以一般存在 SDARM 中，当 CPU 进入睡眠态时，SDARM 进入自刷新模式，内容不丢失。

(3)设置唤醒源：当进入睡眠态时，要使 CPU 能恢复，必须要设置一个或多个中断唤醒源，比如 RTC、键盘中断。

(4)改变 CPU 频率，进入睡眠。完成上述几步工作以后，就可以进入睡眠态。

挂起函数 SPM_Suspend()的伪码描述如下：

```
void SPM_Suspend()
{
    申请内存，以保护 CPU 中重要寄存器的
    内容;
    调用 pm_send_all 向在 PM 模块中注册了
    的设备发送系统 suspend 的通知
    调用 SPM_CPU_Suspend()，进入睡眠;
}
```

4.3　Bootloader 的支持及 Resume 实现

我们以常用的 ARM 处理器为例进行说明。ARM 处理器分为 7 个运行模式，当处理器开始上电运行时从 0x00000000 地址处运行。而当进入到未定义指令错误、软件中断、预取指令错误、数据存取错误、IRQ 和 FIQ 时，分别到 0x000000018 和 0x0000001c 等地址处执行相应的程序。根据这个特点，要在 0x00000000 处放上跳转指令，在此跳转指令指向的地址中实现相应的 Bootloader 功能。

嵌入式系统中的 Bootloader 相当于 PC 中的 BIOS，主要起到引导系统和传递一些参数信息的作用。在一些功能比较强大的 Bootloader(如 u-boot 等)中，实现了如 ftp 等更加强大的功能，在一定程度上可以说形成了又一个微型内核。只不过这个微型内核的作用不是控制系统，而是为了更好地引导系统。出于方便维护和简洁的特点，我们针对自己的嵌入式系统开发了简单的 Bootloader，其实现的主要功能是搬运、引导内核，通过串口更新内核。在系统引导起来之后，Booloader 要把从外设接到的异常、中断交到操作系统，由操作系统来处理。

4.4 设备驱动程序实现

设备驱动程序并不参与到调度当中，只是被动地接受系统的调用。当系统进入 Suspend 时，调用各个外设的 Suspend 处理函数，外设进行相应的保存现场处理；当 Resume 时，外设唤醒。从这个层次来说，设备驱动程序的电源管理只是整个电源管理对驱动开发的一个接口部分，但由于驱动开发和内核的关系比较紧密，所以在划分电源管理的体系结构时，把它归到内核这一层次。在我们的 SPM 电源管理体系结构中，针对设备驱动程序的结构分为两个部分：一是供内核调用的接口，这些接口可以看做是内核与设备驱动程序之间的桥梁。这些函数有：

int emit（“PM”,void *data）;

通知一个支持电源管理的设备当前状态，要求其采取相应的动作。动作类型有如下几种：

i. PM_SUSPEND 挂起设备；

ii. PM_RESUME 恢复设备。

signal-slot 系统会遍历已注册设备，找出符合条件的设备，并返回设备句柄指针。

下面是驱动程序在注册时要实现的接口，这一部分接口和上面的系统接口是相辅相成的。上面接口函数用到的结构都要在驱动程序中注册，换句话说，如果在驱动程序编写时没有实现相关的接口，那么系统接口也无现实意义。从调用层次上讲，上面的接口函数是对内核调用的，对驱动程序编写人员来说是透明的，而下面的接口是驱动程序必须实现，否则整个系统电源管理就难以实现。

要实现的接口及作用如下：

Connect（char *signal, void *handler,1）

注册一个电源管理设备，其中 hanler 为系统执行电源管理时调用的函数，用以设备的电源管理任务。

DisConnect（char *signal, void *handler）

这两个函数用来监测设备处于忙或者空闲状态。比如，当某一设备有事件发生时，调用 emit（“PM_RESUME”，null）更新设备访问时间或者唤醒设备；当设备处于空闲时，调用 emit（“PM_SUSPEND”，null）更新设备的空闲状态。

在编写设备程序时，一定要使驱动程序可阻塞调用。当驱动程序是轮询方式时，系统将无法进入优先级低的 IDLE 状态。而电源管理的 Time-out 需在 IDLE 状态中实现，否则整个电源管理策略将形同虚设。

4.5 CPU 频率和 LCD 背光管理

CPU 在不同的频率下工作，其功耗是不一样的。我们实验平台采用的 LM3S 系列单片机主要有 3 种工作模式：运行模式（Run-Mode）、睡眠模式（Sleep-Mode）和深度睡眠模式（Deep-Sleep-Mode），某些型号还具有单独的极为省电的冬眠模块（Hibernation Module）。而对各个模式下的外设时钟选通以及系统时钟源的控制主要由配置相应寄存器来完成。根据这一特性，可在系统初始化阶段启动一个后台进程，在进程定期检测系统电量，根据电量值使 CPU 进入不同的频率工作。这有点像笔记本，插上电源时就以较高的频率工作，电量不足时就以较低的频率工作，以达到省电的目的。

后台进程可以根据电源电量设置相应寄存器来改变 CPU 频率，还可以通过设置 CLKSLOW 进入 SLOW 模式下工作。进入 SLOW 的过程如下：

```
void enter_SlowMode()
{
  保存寄存器配置;
  进入 SLOW 模式下工作;
  重新配置存储器控制;
}
```

后台处理进程的创建：

static void rest_init（void）

```
{
   其他函数省去;
   建立系统电量检测后台进程;
}
static void CPU_thread(void)
{
   if ( 电量<SLOW的电量值)
         CPU 进入 SLOW 模式运行;
   else
if (电量<正常运行值)
               改变CPU频率运行;
}
```

LCD 是嵌入式设备中比较耗电的部分，有效地对 LCD 进行电源管理，能节省大量电能，延长嵌入式设备的运行时间。对 LCD 驱动程序的编写，让其他设备可以通过相关调用来打开或关闭背光。比如在用户按键时如果没有关闭背光就应该把背光打开，在一定时间内没有按键就应该把 LCD 的背光关闭。

5　实验结果

采用 signal-slot 体系会带来额外开销，如当调用 signal-slot 的 emit 触发函数时，emit 采用 hash 算法进行函数指针查找，时间复杂度为 O(1)。经过测试，在我们的测试平台中(CPU 主频 300 MHz)，每 100 W 调用耗时 25 ms；如果直接调用，每 100 W 调用耗时 6 ms。因此 single-slot 调用速度相当于直接调用的 1/4。

我们将上面实现的电源管理软件框架应用到 Contiki 中，并将 Contiki 移植到我们的网络融合测试平台，如图 3 所示。由于电源管理的效果和实际应用场景相关性很强，目前我们仅仅验证 SPM 可以根据系统状态切换 CPU 工作状态，由于 Contiki 本身没有自带电源管理，实际的节能效果数据还需要进一步得出。

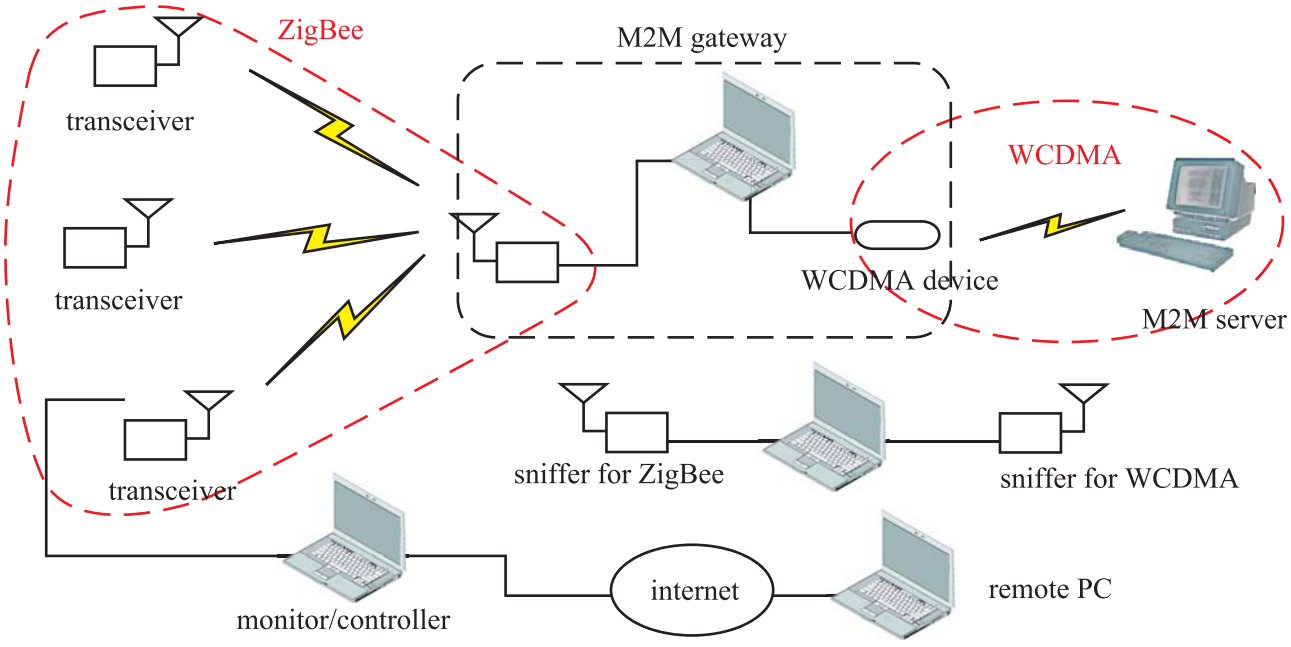

图 3　网络融合测试平台

Fig. 3.　The network integration testing platform

6 结 论

本文提出了一种针对无线传感网络节点的 signal-slot 软件体系，并基于 signal-slot 设计了精简的电源管理方案 SPM，最后将 SPM 应用到传感器操作系统 Contiki 中，弥补了 Contiki 没有电源管理方案的不足。极大地方便了物联网传感节点软件的开发，但是目前基于 signal-slot 的 SPM 还没有得到充分优化，需要设计一些实际的应用场景进一步改善。

参 考 文 献

[1] Yazar D, Dunkels A. Efficient application integration in IP-based sensor networks [C] // Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, 2009: 43-48.

[2] Chandrakasan A, Gutnik V, Xanthopoulos T. Data driven signal processing: an approach for energy efficient computing [C] // International Symposium on Low Power Electronics and Design, 1996: 374-352.

[3] Benini L, Bogliolo A, De Micheli G. A survey of design techniques for system-level dynamic power management [J]. IEEE Transactions on Very Large Scale Integration Systems, 2000, 8(3): 299-316.

[4] Pouwelse J, Langendoen K, Sips H. Application directed voltage scaling [J]. IEEE Transactions on Very Large Scale Integration, 2003, 11(5): 812-826.

[5] Liu XT, Shenoy P, Corner M. Chameleon: application level power management with performance isolation [C] // Proceedings of the 13th annual ACM International Conference on Multimedia, 2005: 839-848.

[6] Intel/Microsoft Corporation. Advanced Power Management (APM) BIOS Interface Specification, revision 1.2 [Z]. [1996-12]

[7] Hewlett-Packard, Intel Corporation, Microsoft, Phoenix Technologies, Toshiba. Advanced Configuration and Power Interface Specification, revision 5.0 [OL]. [2011-12-6] http://www.acpi.info/spec50.htm.

[8] Qt Project. Signals & Slots [EB/OL]. http://developer.qt.nokia.com/doc/qt-4.8/signalsandslots.html.

[9] Wu Q, Juang P, Martonosi M, et al. Formal online methods for voltage/frequency control in multiple clock domain microprocessors [C] // Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, 2004, 38(5): 248-259.

[10] Yuan WH, Nahrstedt K. Practical voltage scaling for mobile multimedia devices [C] // Proceedings of the 12th annual ACM International Conference on Multimedia, 2004: 924-931.

[11] Pouwelse JA, Langendoen K, Sips HJ. Application-directed voltage scaling [J]. IEEE Transactions on Very Large Scale Integration, 2003, 11(5): 812-826.

[12] Lu Z, Hein J, Humphrey M, et al. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads [C] // Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2002: 156-163.