**摘要**

本文首先介绍了对于开发平台的选择，包括蓝牙芯片 DA14580 的简要介绍以及与相似蓝牙开发平台的对比分析,接着简要阐述了在使用 DA14580 蓝牙芯片的嵌入式单片机上移植实时操作系统的优势，包括最终所确定的操作系统 μC/OS-II 的优越性。最后介绍了目前在可穿穿戴设备上降低功耗的必要性，以及目前工业界拥有的成果，简述我们的解决方案的优势所在。

# 基于 ARM 架构的超低功耗 OS 调研报告

崔天一 曹焕琦 邓翔 李泽昌

2016 年 3 月 13 日

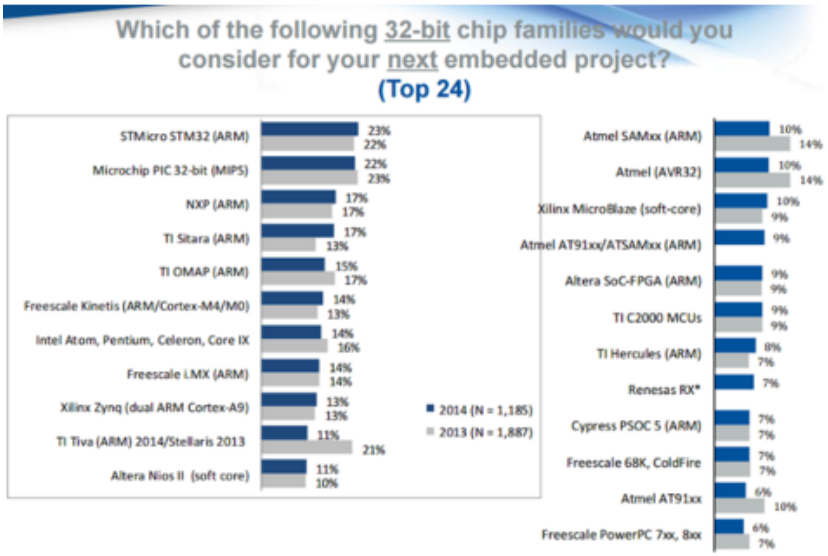
## 目录

# 1 开发平台的选择

对于开发平台的选择，我们需要同时考虑到低功耗特性和足够的功能以面对各类实际问题。下面我们比较一下若干可能选择的情况。

| 平台类型 | 开发方式 | 功耗 | 功能 |
|---|---|---|---|
| FPGA | HDL | 极低 | 可强可弱，实现困难 |
| 高效高通用性平台（如 ARM Cortex-A） | 汇编语言，C/C++ | 较高 | 强 |
| 高可靠性嵌入式平台（如 ARM Cortex-R） | 汇编语言，C/C++ | 中 | 较强 |
| 高能效嵌入式平台（如 ARM Cortex-M） | 汇编语言，C/C++ | 低 | 较强 |

由以上列表可看出，出于实现难度、功耗、功能三方面考虑，高能效的嵌入式平台更加适合超低功耗的可扩展平台的开发。基于此理由，我们倾向于选择高能效的微处理器。



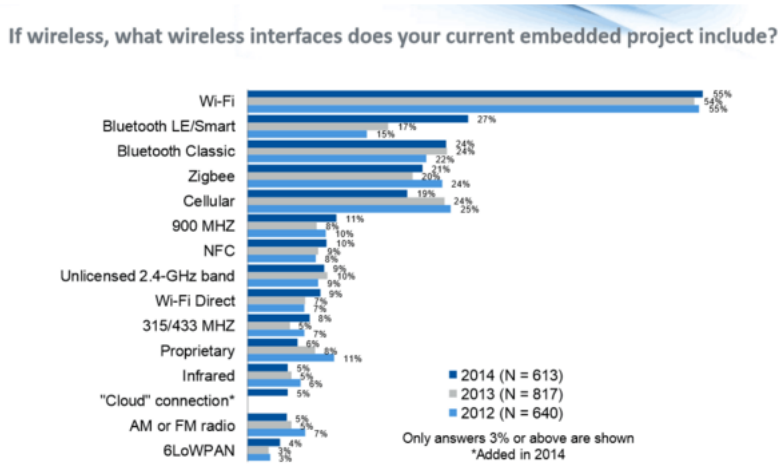

在微处理器的架构选取中，主要有 ARM、MIPS、AVR32、Intel Atom 等架构。为方便开发，我们倾向于选取使用率较高的微处理器架构；因为它们的相关资料往往较多，开发难度也较低，同时使用率高也代表着它有着其独特的优越性。

在上图中，我们可以看到，ARM 架构处理器占据了前五名的三个位置；由此可见 ARM 架构有着更加适合嵌入式开发的特性。我们倾向于选择 ARM Cortex-M 系列微处理器，它在硬件条件令人满意的同时，也有着丰富的开发资源和良好的生态环境，能够适应我们的高可扩展性要求。

而在比较 Cortex-M 系列各芯片中，Cortex-M0 的功耗最低，虽然主频通常较低、功能也较弱，但在嵌入式的环境下已经足够使用。

# 2 实际平台的选取——BLE 芯片 DA14580

Bluetooth Smart，技术称 Bluetooth low energy，在 2006 年由 Nokia 提出，并于 2010 年正式由 SIG 加入 Bluetooth 标准。它的主要目标是保证传输范围与效果的前提下，大幅度降低功耗。近年来，绝大多数移动平台，包括移动电话、笔记本电脑、可穿戴设备，均包含 Bluetooth Smart 支持；见下图。




从图中我们可以看出，除 Wi-Fi 由于其互联网链接特性有着比 Bluetooth 更高的占有率外，Bluetooth Smart 在连续的飞速增长中占居第二，作为其前代产品的 Bluetooth Classic 也始终稳居前几名。蓝牙有着便利的数据传输功能，相比其它无线连接往往更低的功耗，和令人满意的速度；因此，以蓝牙为基底进行低功耗 OS 的开发是合理的，尤其是考虑到 Bluetooth Smart 本身的低功耗特性。一方面 Bluetooth Smart 模块可作为复杂设备（如移动电话）的组件，另一方面也可以作为可穿戴设备等小型设备的中心

组成；对于后一种情况，往往需要一个适用于嵌入式的低功耗 CPU 来完成整个系统。这里我们比对流行的蓝牙智能芯片的数据如下 [10][7][6]，休眠时保留时钟：

| 芯片型号 | RX 工作电流 | TX 工作电流 | 休眠电流 | 处理器 |
|---|---|---|---|---|
| DA14580 | 4.9mA | 4.9mA | 400nA | ARM Cortex-M0 |
| CC2540 | >19.6mA | 24mA | 900nA | ARM Cortex-M0 |
| CSR101x | 20/18mA | 20/18mA | 5uA | 8051 |

三类流行的蓝牙智能芯片之间，功耗对比十分明显：DA14580 有着远低于其它二者的功耗。同时，搭载了 ARM Cortex M0 微处理器的 DA14580 也有着足够的灵活度以面对各种实际问题。综合各方面因素，它是低功耗高灵活度蓝牙智能平台解决方案的最佳选择之一，本项目中也将选择此芯片作为平台。

# 3 单片机移植嵌入式操作系统优越性

传统单片机通常并未安装操作系统，通过各类为特定目的编写专用程序来实现相应功能，在过去各类功能需求较为有限的情况下，这种方式能够提供足够可靠的解决方案。然而随着时代的发展却暴露出了越来越多的问题。但是通过移植操作系统可以较好的解决这些问题，具体如下：

1. 降低开发门槛。

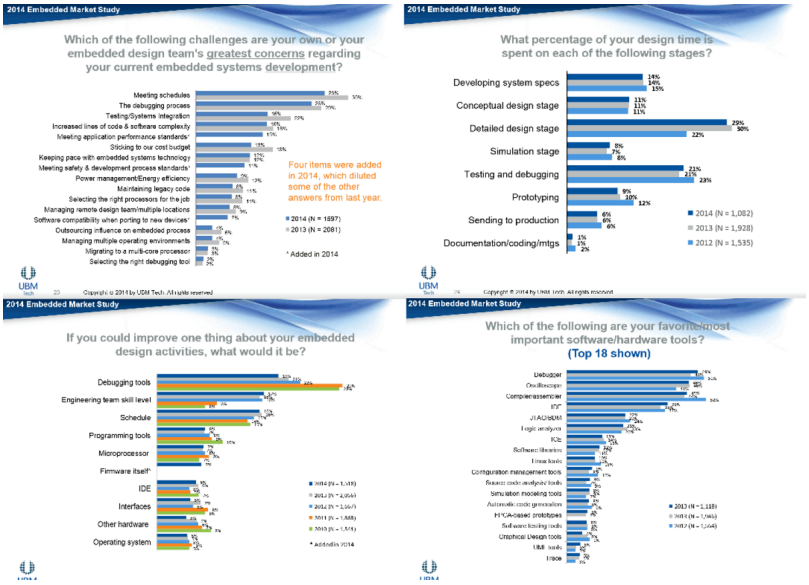
   传统单片机应用的开发，通常与底层硬件紧密相关，这要求开发人员不仅需要掌握程序设计的一般技巧，更要对于底层硬件语言与各类接口有着较好的理解，这大大提高了开发门槛，使得只有极少数人能够为单片机开发应用。将操作系统移植到单片机之后可以使各类开发者专注于自己的本职工作。软件开发者可以将注意力放在如何通过与操作配合开发出更加高效以及功能更加丰富的软件。而系统工程师则可以更多的关注于操作系统与硬件之间的协同，从而在性能和功耗之间达成更好的平衡。[4] 类似于编程语言从机器码到汇编到高级程序设计语言的进步。可以大大提高效率。

2. 便于外设添加以及应用开发移植。

传统方式的单片机开发在往往应用与硬件结合较为紧密，一旦软件编写结束烧入板子之后如果需要加入新的功能往往需要将原有的设计完全推翻重来，即使不必如此也往往会将原有的设计打乱，从而影响整体的工作效率。而移植操作系统之后，操作系统将底层接口等封装起来，从而大大提高了效率并缩减了所需要的时间，这在如今功能需求变化迅速的时代尤为重要。同时操作系统使得同一应用在不同平台之间的移植变得便利，减少了开发人员的负担，能够吸引更多人参与开发，促进产业发展。[8]

3. 便于纠错和分析。

传统单片机开发软件编写好之后需要烧入芯片才能进行调试。调试需要使用专用接口并且需要有相应的开发经验和对于底层硬件很强的理解，非常不便和耗时。同时也很难区分究竟是硬件问题还是软件问题。当应用程序变得复杂之后，这些问题会越发的凸显出来。




由以上调查可知 [9]，真实开发过程中，检错测试所占用的时间与开发耗时近乎相等。同时许多开发者也表示调试工具的便利性是他们非常重视的一个环节。而许多操作系统自带相关的调试和分析工具，能够帮助开发者迅速发现问题从而解决问题。同时，操作系统能够协助开发者检测应用的资源消耗，从而针对性的进行优化。

4. 更出色的多任务调度。

   操作系统提供了更好并且更安全的多任务并行方式，在不含操作系统的小型单片机中，开发者常常利用全局变量来实现不同模块和应用间的通信和同步。但是大量使用全局变量可能导致错误与安全性问题，尤其是当系统规模变大时，这些被共享的信息很容易造成冲突从而影响系统整体的工作效率甚至导致系统崩溃。但是 RTOS 通常都已经提供了较为完善的进程调度方案，移植操作系统后，开发者便可以专注于应用本身的开发而将这些协调问题交给操作系统解决。

5. 对于硬件资源的更加高效的应用。

   传统的单片机应用多采取轮询方式来检测中断是否发生，结果导致了大量的核心闲置时间。得益于多任务实时操作系统所采用的中断驱动方式，我们可以最大限度的减少轮询。这使得核心的资源得到了最大化利用，并且能够在闲置时尽可能多的进入休眠状态从而降低功耗。[8]

# 4 嵌入式操作系统选择——μC/OS-II

嵌入式操作系统是在嵌入式设备上运行的操作系统，这类系统通常具有紧密、高效、可靠、实时等特性。[16] 这类操作系统通常为实时操作系统（RTOS），运行在资源受限的设备上，它们内存通常较少，且为了充分利用 CPU，有一部分代码使用汇编语言编写。特别地，嵌入式操作系统区别于其他的桌面操作系统，通常不从 ROM 中动态加载应用程序，而是将所用应用程序和操作系统自身打包成一个刷写镜像刷写至设备中。

目前，市面上流行的嵌入式操作系统主要有以下几类：[2]

- 嵌入式 Linux[15]

  这类操作系统以 Linux 内核为基础，被设计用来使用嵌入式设备，这类操作系统被广泛的用在手机等消费产品中。

  如上图为一个使用了嵌入式 Linux 的手机。该系统和其他系统相比有以下好处：

  – 开放源代码
  – 不需版权费用
  – 成熟与稳定（经过这些年的发展和使用）

但是，我们不能采用嵌入式 Linux 系统作为我们移植至芯片上的系统，这是因为一个 RTLinux 通产需要 2M 存储空间，但开发板上通产只有若干 KB 的存储空间（对于开发板来说是 84KB），由于存储空间的限制，无法使用这类操作系统。

- Windows CE[13]

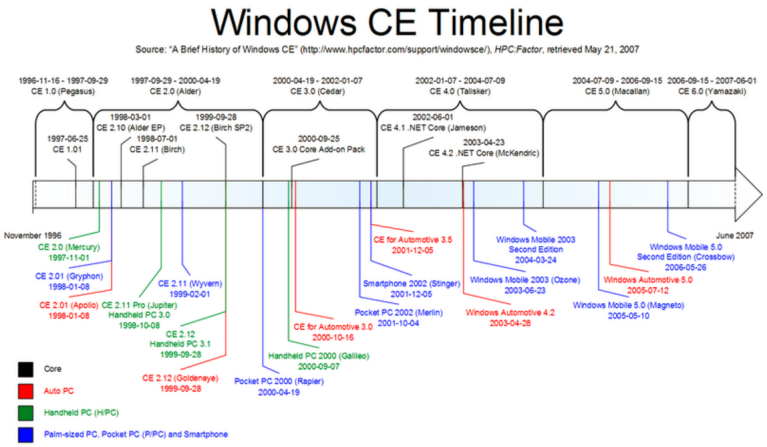
  Windows Embedded Compact（旧称 Microsoft Windows CE）为微软研发的一套嵌入式操作系统，如下为 Windows CE 的发布历史。

  但我们仍然无法使用其作为我们要移植的操作系统，这是因为 Windows CE 未完全开放其源代码，使我们的移植工作受到了很大的限制。另外，Windows CE 的实时性不如 C/OS。

- VxWorks[14]

  VxWorks 是美国 WindRiver 公司设计的一种 RTOS，具有高性能的内核、友善的开发环境，且有较高的市场份额。[11] 如下为各商业嵌入式操作系统的市场份额。且其广泛应用于通信、航空、航天、军事等领域内。特别是国内军用手持设备也大量的使用了 Vxworks 的产品。

  但使用其需要支付昂贵的授权费，故我们无法使用该操作系统。




- µC/OS-II[12]

  µC/OS-II 是由 Jean J. Labrosse 于 1991 年开发的实时操作系统，它基本上是用 C 语言编写完成的，具有可固化、可移植的特性。其进程调度是根据优先级进行的。同时，该系统还提供了内存管理、时序管理（timing）、进程间通讯等功能。

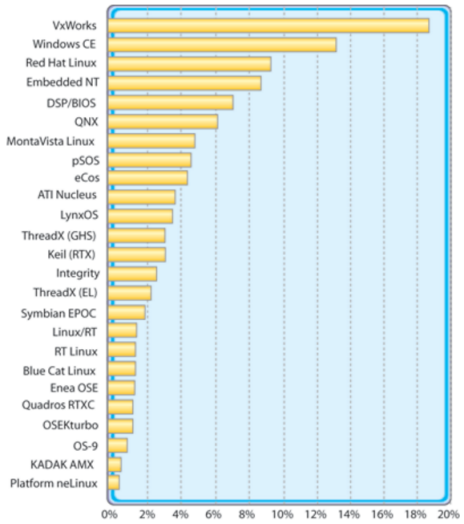
  在本项目中，我们由于其源码可得、代码精简、可移植性强、文档丰富的特性选择其为我们带移植的系统。

- µC/OS-III[1]

  µC/OS-III 系统为 2009 年发布的针对 µC/OS-II 的改进版本，其主要增加了如下的功能

  – Round robin 进程调度算法
  – 将进程数上限由 255 提升至无限
  – 允许多进程共享相同的优先级

  但由于 µC/OS-III 的代码长度和复杂程度比 µC/OS-II 有一定程度的增加，为了保证项目的结构和代码的清晰，同时考虑到在我们的平台上进程数较少，µC/OS-III 更加优秀的进程调度可能没有显著的效果，我们使用 µC/OS-II 作为待移植的系统而非 µC/OS-III。




# 5 µC/OS-II 的优点

1. 可移植

   µC/OS-II 源代码公开且绝大多数用 ANSI C 编写，其余用较少量的汇编语言编写，可分为内核层和移植层，因此移植起来比较方便。只要对应的微处理器具有堆栈指针和 CPU 内部出栈、入栈指令，µC/OS-II 就可以移植，故现已广泛用于 8 位、16 位和 32 位甚至 64 位的单片机或 DSP。

2. 可裁剪

   可以根据用户的需求，选择性地减少 µC/OS-II 的调用，以此减小存储器的占用。而且可裁剪性可以通过条件编译较为简易地进行实现。充分裁剪后的 µC/OS-II 系统可以达到 10K 以内，故认为其满足嵌入式系统的要求。

3. 可固化

   µC/OS-II 专门为了嵌入式系统而设计，只要具备合适的系列软件工具 (C 编译、汇编、链接及下载/固化)，完全可以将 µC/OS-II 嵌入到产品中。

4. 稳定性

2000 年 7 月，µC/OS-II 得到了 FAA 对用于商用飞机、符合 RTCA DO-178B 标准的认证。这个认证充分说明了 µC/OS-II 的安全性与稳定性，能用于像商业飞机这样苛求安全稳定的场合。

5. 系统服务

µC/OS-II 提供了许多系统服务，如信号量、事件标志、消息队列、消息邮箱等等。

# 6 µC/OS-II 的应用

µC/OS-II 由于其可靠性、轻量性，被广泛地应用于各类特殊用途的嵌入式开发。在以下嵌入式系统中，有大量的产品选择使用了 µC/OS-II。

- Avionics
- Medical equipment/devices
- Data communications equipment
- White goods (appliances)
- Mobile phones, PDAs, MIDs
- Industrial controls
- Consumer electronics
- Automotive

这里我们列举一些实际的应用。

- 好奇号火星车

  好奇号火星车上搭载的用来探测火星大气与土壤中化学与同位素成分的 SAM 模块是由 µC/OS-II 操作系统内核控制的。它的高实时性保证了配合采集行为即时分析物质组成的能力，它的高可靠性同时保证了组件在恶劣环境下的正常工作。

- 基于 µC/OS-II 和 MSP430F5438A 的心率监视仪

  Texas Instruments 推出的 MSP430F5438A 可用于制造心率监视仪。根据来自 TI 的应用报告，他们推荐使用 µC/OS-II 作为平台来进行开


发。这说明了 µC/OS-II 的高可靠性，同样也侧面表明了 µC/OS-II 的广泛用途。

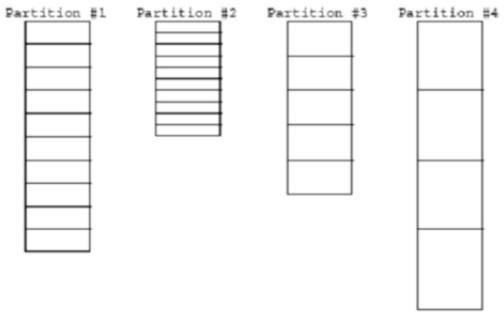
# 7 µC/OS-II 部分特性简介

1. 任务管理

   µC/OS-II 可以管理多达 64 个任务，每个任务的优先级均不同，故 µC/OS-II 有 64 个优先级，级号越低，优先级越高。例如当 µC/OS-II 初始化时，优先级最低的 OS_LOWEST_PRIO 总被赋与空闲任务 idle task。因为优先级是唯一的，故 µC/OS-II 也可通过优先级来识别任务。由于保留 8 个优先级，用户有多达 56 个应用任务。每个任务放在任务就绪表中，里面有两个变量 OSRdyGrp 和 OSRdyTbl[]。可以任务的就绪状态通过这两个变量的置位来表示。

2. 内存管理

malloc 和 free 这两个函数在 ANSI C 中被用来动态控制内存的分配与回收。但在嵌入式操作系统中，这样做却是十分危险的。如果多次地调用这两个函数，原本一块完整的较大的内存区域会被分割成许多不相邻且非常小的内存碎片。最终程序将连一块很小的内存也申请不到。µC/OS-II 通过把内存分区管理，解决了这个问题。每个区包含整数个大小相同的内存块。于是，µC/OS-II 改进了 malloc 和 free 两个函数，使之申请与释放固定大小的内存块。如图所示，不同的分区的内存块大小不同，于是通过申请不同分区的内存块，可获得不同大小的内存，释放时同样回到原内存分区。



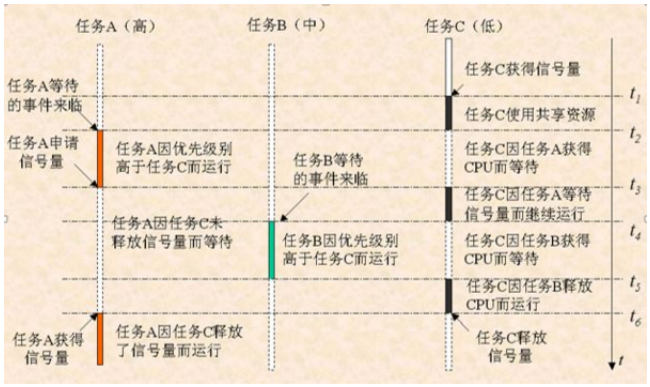

3. 时间管理

   µC/OS-II 要求提供定时中断，以实现延时与超时等功能。用硬件定时器产生一个周期为 ms 级的周期性中断来实现系统时钟，最小的时钟单位就是两次中断之间相间隔的时间，这个最小时钟单位叫做时钟节拍 (Time Tick)，应发生 10-100 次/秒。时钟节拍的频率越高，系统负荷越大。硬件定时器以时钟节拍为周期定时地产生中断，该中断的中断服务程序叫做 OSTickISR()。中断服务程序通过调用函数 OSTimeTick() 在每个时钟节拍了解每个任务的延时状态，使其中已经到了延时时限的非挂起任务进入就绪状态。有时需要让高优先级的任务暂时让出 CPU 的使用权，那么可以用 OSTimeDly() 函数暂停当前任务的运行。此外还有取消延时的 OSTimeDlyResume() 函数，获取系统时间的 OSTimeGet() 函数等等。总而言之，µC/OS-II 对时间的操作是非常灵活的。

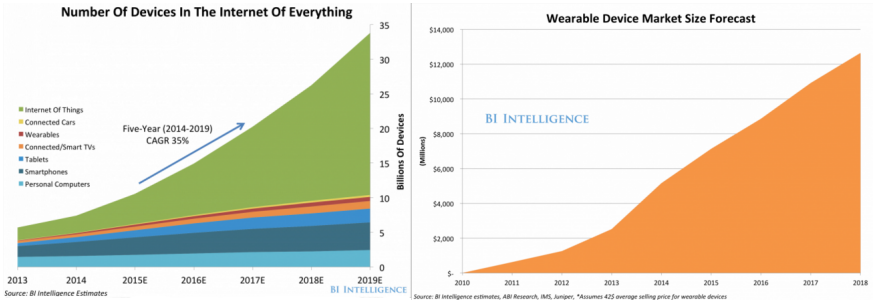
4. 互斥型信号量

   µC/OS-II 是可剥夺性内核，当任务以独占方式使用共享资源时，会出

现任务优先级反转的情况。下图显示的就是从 t1 到 t6 出现的优先级反转情况。为了解决这个问题，应让获得信号量正在使用共享资源的优先级暂时得到提升，要高于所有任务，从而使共享资源尽快被使用完而释放。等释放了信号量，再恢复任务原有的优先级。而互斥型信号量的 ECB 模型就支持临时提升优先级的操作。



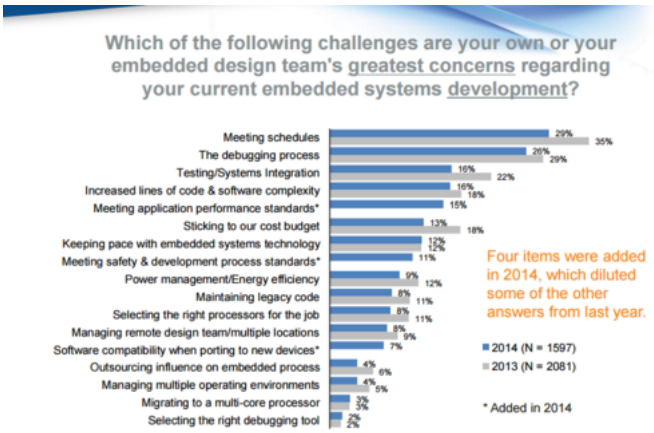

# 8 降低芯片功耗的必要性




从上二图中可以看到，随着互联网技术的发展，物联网、智能家居、可穿戴设备也进入了人们的视野。[5] 而作为新时代中新的人机交互形式和更便利的电子设备，可穿戴设备的市场预期也在不断地增长，各大厂商亦在不断地推出新产品。[3] 但由于可穿戴设备的低重量、小体积要求，它们往往只能配备小容量电池；为了保证待机，必须做到较低的功耗。

在上图中 [9]，我们可以发现，降低芯片功耗的必要性已经纳入了许多开发者最优先考虑的因素；功耗决定待机时间，而待机时间极大地影响着产品在市场上的受欢迎程度。

但同时，我们需要注意到，通常的嵌入式开发由于往往出于种种因素不



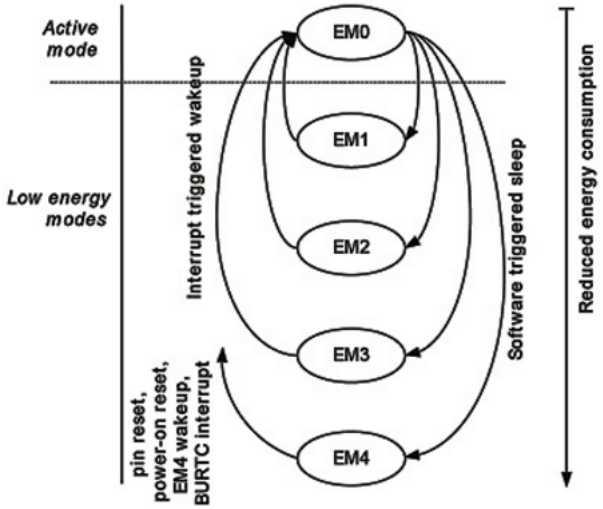

考虑通用性，也即并不存在完整的操作系统，因而能够避免 CPU 调度、完整地调配 CPU 的行为，在适当的时候进行休眠，从而避免功耗的增加；但当我们为了通用性和可扩展性而加入操作系统时，功耗就变得很难以降低。对于通常的操作系统，CPU 始终工作，导致了巨大的能耗；同时，若不适时地关闭蓝牙模块，它的功耗也是不能忽视的。

# 9 休眠——一种降低功耗的实现

ARM Cortex-M0 微处理器允许芯片制造商提供了多种能耗模式（Energy Mode）进行选择，这是指关闭或开启部分组件从而在牺牲一部分功能的情况下降低功耗。如下图为某芯片提供的在各个能耗模式间切换的状态图：

如上图所示，该芯片正常工作在 EM0，当执行相应的休眠指令时，会转移至相应的休眠状态（EM1 4）。对于 EM1 3，可通过中断唤醒；对于 EM4，只能通过外部 Reset 信号唤醒。休眠状态的数量和行为并不由 ARM 定义，而是由芯片制造商定义。

根据上图（来自 Dialog 提供的 DA14580 SDK），对于 DA14580，它将四种休眠状态缩减为 3 种：Idle，Extended Sleep Mode 和 Deep Sleep Mode。对于第一种和第二种，它们通过中断唤醒，类似于先前的芯片的 EM1 3；对于后者，它通过特定信号唤醒，相当于 EM4。在 Idle 状态下，芯片中的 CPU 停止工作，但蓝牙保持工作状态，SysRAM 上电；在 Extended Sleep Mode 中，处理器和蓝牙休眠，但保持 SysRAM 上电；在 Deep Sleep Mode



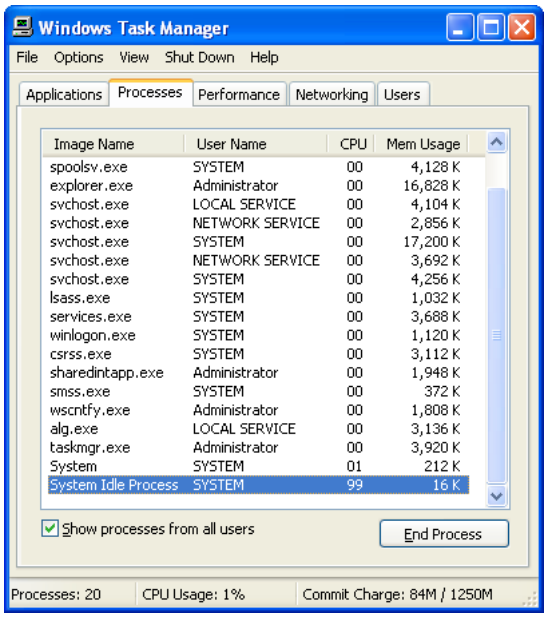

```
typedef enum
{
    mode_active = 0,
    mode_idle,
    mode_ext_sleep,
    mode_deep_sleep,
    mode_sleeping,
}sleep_mode_t;
```

中，处理器、蓝牙和 SysRAM 均停止工作。mode_sleeping 指将进入休眠，具体休眠状态由应用决定。

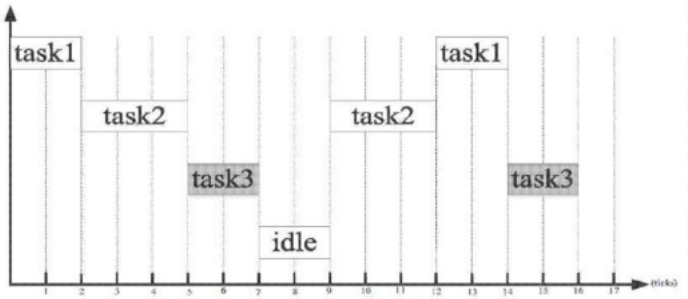
在大多数现有的通用系统中，由于 CPU 性能往往过剩，CPU 处于等待任务状态的时间往往较长。若按照传统方案，Idle 状态 CPU 应处在一个循环中，直到新任务抵达。但在这个循环中，CPU 将进行大量的取指、译码、执行等操作，大幅度提高了功耗。如下图，在 Windows NT 内核操作系统中，System Idle Process 往往占用着超过 90% 的 CPU 时间 [17]。

因此，可以考虑将 Idle 循环替换为休眠，以实现功耗的降低。而对应于 µC/OS-II，在操作系统系统初始化后会建立一个优先级最低的任务——空闲任务。在没有任何任务就绪时会执行这个空闲任务，即进入 idle 任务。

如下图所示，在 task3 执行完毕后，执行 task2 之前，并不需要频繁地唤醒系统。若在 tick8 处唤醒系统，系统仍然会进入 idle 任务。因此可利用 µC/OS-II 的空闲任务扩展接口，进入低功耗模式而不影响其他任务的调度与系统的稳定性，降低系统的功耗。与此同时，利用 RTI 信号周期性地唤




醒 CPU。系统被唤醒后，将执行中断服务程序，做一定的处理后，检测当前是否有任务恰好处于就绪态。若存在任务处于就绪态，则执行该任务，否则重复上述的操作。这也是当下流行的 µC/OS-II 低功耗处理方法。




而关于具体的实现，一种实现方法是，首先在进入 idle 任务时遍历当前任务链表，获取所有等待状态任务的最小延时。再把定时器设为这个时间，由此便可以增加进入低功耗模式的时间，减少唤醒次数，降低系统的功耗。经过这段最小延时后，触发中断服务程序。该中断服务程序将当前系统的基时加上该最小延时，遍历任务链表，把任务的等待时间减去该最小延时，此时应有任务处于就绪状态，最后中断服务程序退出。在 idle 任务时查找最小延时需要临时关闭中断，等待被定时器唤醒时，开中断，执行终端服务程

序。

再回到 DA14580。考虑到蓝牙模块的能耗，在 idle 任务下不仅需进入 mode_idle，还需适当地关闭蓝牙模块，即进入 mode_ext_sleep。通过这样的处理，我们能够大幅度地降低功耗。由于现有的低能耗解决方案往往没有出现在带有蓝牙功能的芯片上，适时关闭蓝牙模块的方式与效果尚需要我们进行研究与完成其实现。

# 10 结论

随着技术发展与市场扩张，低功耗高可用的小型设备的重要性变得不可忽视。而以低功耗、高可用性、高可扩展性为目标，我们选定了基于 ARM 架构的 DA14580 芯片，并将在其上移植 µC/OS-II，进而实现休眠式的 Idle，以完成超低功耗的目标。

# 参考文献

[1] µC/OS-III RTOS kernel | micrium. `https://www.micrium.com/rtos/ucosiii/overview/`. Accessed: 2016-3-12.

[2] 三种嵌入式操作系统的分析与比较 - 随意 ru 风的专栏 - 博客频道 - CSDN.NET. `http://blog.csdn.net/sui_yirufeng/article/details/8997008`. Accessed: 2016-3-12.

[3] Internet of things market statistics - 2015 - IoT stats. `http://www.ironpaper.com/webintel/articles/internet-things-market-statistics-2015/`, 5 March 2015. Accessed: 2016-3-12.

[4] T. N. B. Anh and S. L. Tan. Real-time operating systems for small microcontrollers. *IEEE Micro*, 29(5):30–45, 2009.

[5] Marcelo Ballve. Wearable gadgets are still not getting the attention they deserve — here's why they will create a massive new market. `http://www.businessinsider.com/wearable-devices-create-a-new-market-2013-8`, 29 August 2013. Accessed: 2016-3-12.

[6] CSR. Csr µenergy® csr101x family.

[7] Texas Instruments. Cc2540 | bluetooth / 蓝牙低耗能 | 无线连接 | 描述与参数.

[8] Texas Instruments. Why use a real-time operating system in mcu applications.

[9] Embedded Systems Design magazine. 2014 embedded market survey.

[10] Dialog Semiconductor. Smartbond da14580 bluetooth smart power size and system cost without compromise.

[11] Jim Turley. Embedded systems survey: Operating systems up for grabs. `http://www.embedded.com/electronics-blogs/-include/4025539/Embedded-systems-survey-Operating-systems-up-for-grabs`. Accessed: 2016-3-12.

[12] wikipedia. Micro-controller operating systems.

[13] wikipedia. Vxworks.

[14] wikipedia. windows ce.

[15] wikipedia. 嵌入式 linux.

[16] wikipedia. 嵌入式操作系统.

[17] 维基百科. System idle process.