

# TinyOS: An Operating System for Sensor Networks

P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A.  
Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler

郑霄龙



# Outline

---

1. WSN操作系统
2. TinyOS简介
3. 主要技术特点
4. 技术实现



# 无线传感器网络及其操作系统

- 无线传感器网络（Wireless Sensor Network, WSN）是由部署在监测区域内大量的廉价微型**传感器节点**组成，通过无线通信方式形成的一个多跳的自组织网络系统，其目的是协作地**感知、采集和处理**网络覆盖区域中**感知对象**的信息，并发送给**观察者**。
- 有人认为没有必要设计一个专门的操作系统，可以**直接在硬件上设计应用程序**，但在实际过程中会遇到许多问题：
  - 面向传感器网络的**应用开发难度会加大**，应用开发人员不得不直接面对硬件进行编程，无法得到像操作系统那样提供的丰富的服务。
  - 软件的**重用性差**，程序员无法继承已有的软件成果，降低了开发效率。



# 无线传感器网络的特点

---

- 单个传感器节点对操作系统提出新要求的很突出的两个特点：
  - 并发性密集，即可能存在多个需要同时执行的逻辑控制；
  - 传感器节点模块化程度很高，要求系统能够让应用程序方便的对硬件进行控制。
- 因此，必须针对这些特点来设计WSN操作系统。



# WSN操作系统的设计目标

---

- 要求WSN操作系统能够良好的**模块化**设计，使应用/协议/服务与硬件资源之间可以**随意搭配**
- 节点资源有限，操作系统必须能够**高效地使用各种资源**
- WSN是个网络系统，其操作系统需面向网络化开发，必须为应用提供高效的组网和通信机制



# 内容提要

---

1. WSN操作系统
2. TinyOS简介
3. 主要技术特点
4. 技术实现



# TinyOS简介

---

- TinyOS是一个开源的嵌入式操作系统，由加州大学伯利克分校开发出来的，主要应用于无线传感器网络方面。
- 基于组件的架构，能够快速实现各种应用。
- 因为与同样是他们设计的硬件平台珠联璧合而声名鹊起，目前已经成为无线传感器网络领域最广泛使用的平台
- 支持的硬件平台： eyesIFXv2、intelmote2、mica2、mica2dot、micaZ、telosb、tinynode 。



# TinyOS简介

---

- TinyOS的程序采用的是模块化设计
  - 程序核心往往都很小（一般来说核心代码和数据大概在**400 Bytes**左右）
  - 能够突破传感器存储资源少的限制，让**TinyOS**很有效地运行在无线传感器网络节点上并去执行相应的管理工作等。





# TinyOS的设计理念

- 由于WSN的特殊性，研究人员在设计TinyOS系统时就提出以下几个原则：
  - 1) 能在有限的资源上运行：要求执行模式允许在单一的协议栈上运行；
  - 2) 允许高度的并发性：要求执行模式能对事件作出快速的直接响应；
  - 3) 适应硬件升级：要求组件和执行模式能够应对硬件/软件的替换；
  - 4) 支持多样化的应用程序：要求能够根据实际需要，裁减操作系统的服务；
  - 5) 鲁棒性强：要求通过组件间有限的交互渠道，就能应对各种复杂情况；
  - 6) 支持一系列平台：要求操作系统的服务具有可移植性。



# TinyOS的设计理念

---

- TinyOS关注核心两个设计原则：
  - ***Event Centric:*** Like the applications, the solution must be event centric. The normal operation is the reactive execution of concurrent events.
  - ***Platform for Innovation:*** The space of networked sensors is novel and complex: we therefore focus on flexibility and enabling innovation, rather than the “right” OS from the beginning.



# 内容提要

---

1. WSN操作系统
2. TinyOS简介
3. 主要技术特点
4. 技术实现



# TinyOS的技术特点

## ➤ 组件化编程（Component-Based）

- 提供一系列可重用的组件；每个组件对外提供接口（**interfaces**）
- 提供的接口描述了该组件提供给上一层调用者的功能，而使用的接口则表示了该组件本身工作时需要的功能。
- 一个应用程序可以通过连接配置文件（**A Wiring Specification**）将各种组件连接起来，以完成它所需要的功能。

## ➤ 事件驱动模式（Event-Driven）

- 应用程序都是基于事件驱动模式的，采用事件触发去唤醒传感器工作。
- 事件相当于不同组件之间传递状态信息的信号。当事件对应的硬件中断发生时，系统能够快速调用相关的事件处理程序。



# TinyOS的技术特点

---

## ■ Computational abstractions

- 命令（commands）：typically a request to a component to perform some service, such as initiating a sensor reading.
- 事件（events）：signals the completion of that service.
- 任务（tasks）：function executed by the TinyOS scheduler at a later time.



# TinyOS的技术特点

---

## ■ 两级调度方式

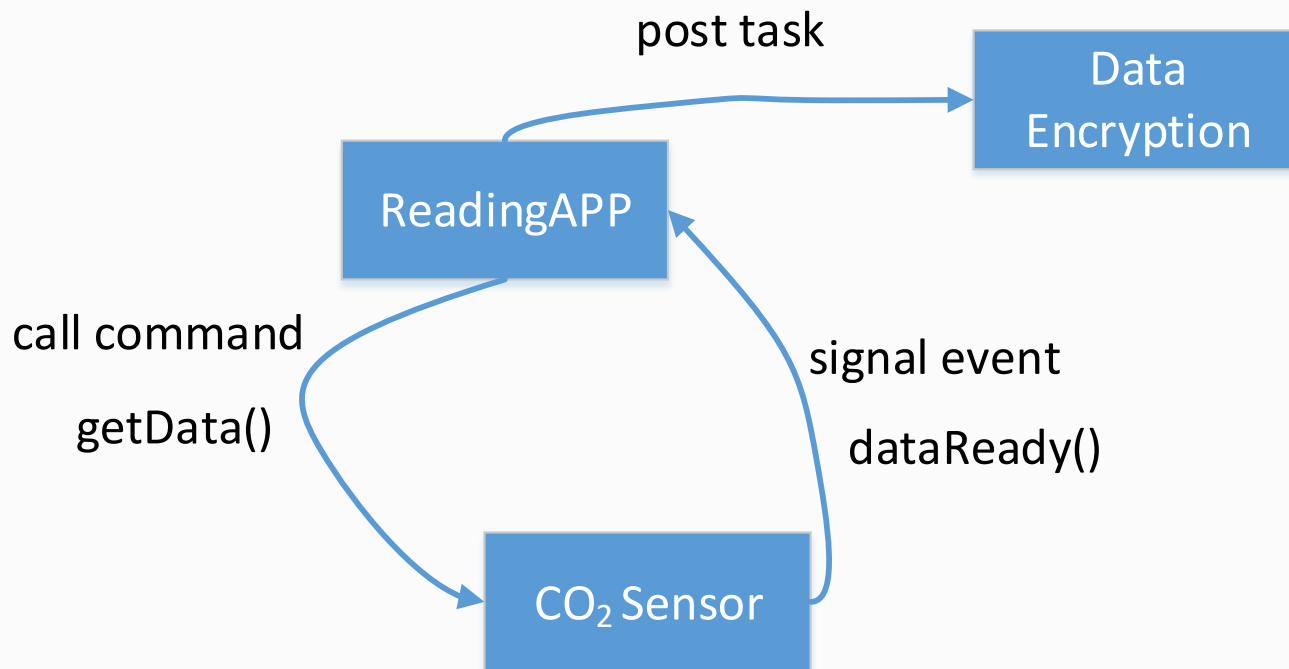
- **任务 (Task)** 一般用在对于时间要求不是很高的应用中
  - 任务都很短小，能够使系统的负担较轻；
- **事件 (Event)** 一般用在对于时间的要求很严格的应用中，且它可以优先于任务执行
  - 由硬件中断处理来驱动事件

## ■ 分阶段作业 (Split-Phase Operations)

- TinyOS没有提供任何阻塞操作，为了让一个耗时较长的操作尽快完成，一般来说都是将对这个操作的请求(**event**、**command**)和这个操作的完成 (**task**) 分开来实现，以便获得较高的执行效率。

# TinyOS的技术特点

## ■ 分阶段作业 ( Split-Phase Operations )



任务队列 FIFO





# 内容提要

---

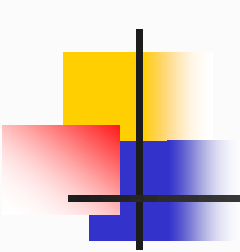
1. WSN操作系统
2. TinyOS简介
3. 主要技术特点
4. 技术实现





# TinyOS的编程语言

- TinyOS最初是用汇编和C语言编写的，后来改用支持组件化编程的nesC语言。
  - nesC语言把组件化/模块化思想和基于事件驱动的执行模型结合起来。
- nesC：使用C作为其基础语言，支持所有的C语言词法和语法，其独有的特色如下：
  - 增加了组件（component）和接口（interface）
  - 定义了接口及如何使用接口表达组件之间关系的方法；
  - 目前只支持组件的静态连接，不能实现动态连接和配置。

- 
- 
- 任何一个使用nesC编写的应用程序都是由一个或多个组件连接而成，从而成为一个可执行的、完整的程序。



# 接口



# 接口

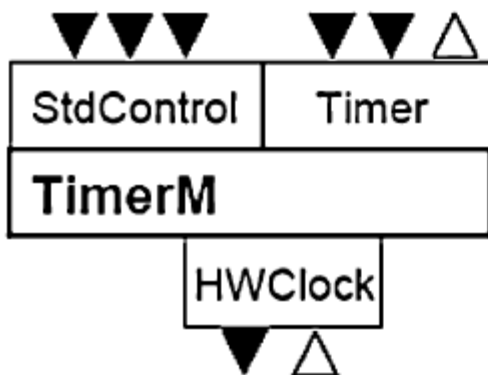
---

- 接口是连接不同组件的纽带
- 接口和接口文件是一一对应的
- 接口名在TinyOS 操作系统中具有唯一性

# 接口

- 两类接口：提供（*provides*）或使用（*uses*）。
  - 组件提供的接口：  
描述本组件提供给上一层调用者的功能
  - 组件使用的接口：  
描述本组件本身工作时需要的功能

## 组件的接口说明举例：



```
module TimerM {  
    provides {  
        interface StdControl;  
        interface Timer[uint8_t id];  
    }  
    uses interface Clock;  
}  
implementation {  
    ... a dialect of C ...  
}
```



# 接口（Interface）

---

- 接口是双向的：提供或使用。
- 接口指定了一组命令（**command**），其职能由接口的提供者实现。还指定了一组事件（**event**），其职能由该接口的使用者实现。
- 也就是说，提供了接口的组件必须实现该接口的命令函数；而使用了某接口的组件必须实现该接口的事件函数。
- 如果一个组件调用了（**call**）一个接口的命令，必须实现该接口的事件。
- 一个组件可以使用（**use**）或提供（**provide**）多个接口或者同一接口的多个实例。

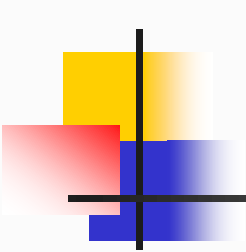


# 接口 (Interface)

---

## ■ 接口的特点：

- **Provides**未必一定有组件使用，但**uses**一定要有人提供，否则编译会提示出错。在动态组件配置语言中**uses**也可以动态配置。
- 一个**module**可以同时提供一组相同的接口，又称参数化接口，表明该**Module**可提供多份同类资源，能够同时给多个组件分享。



## 组件 (Component)

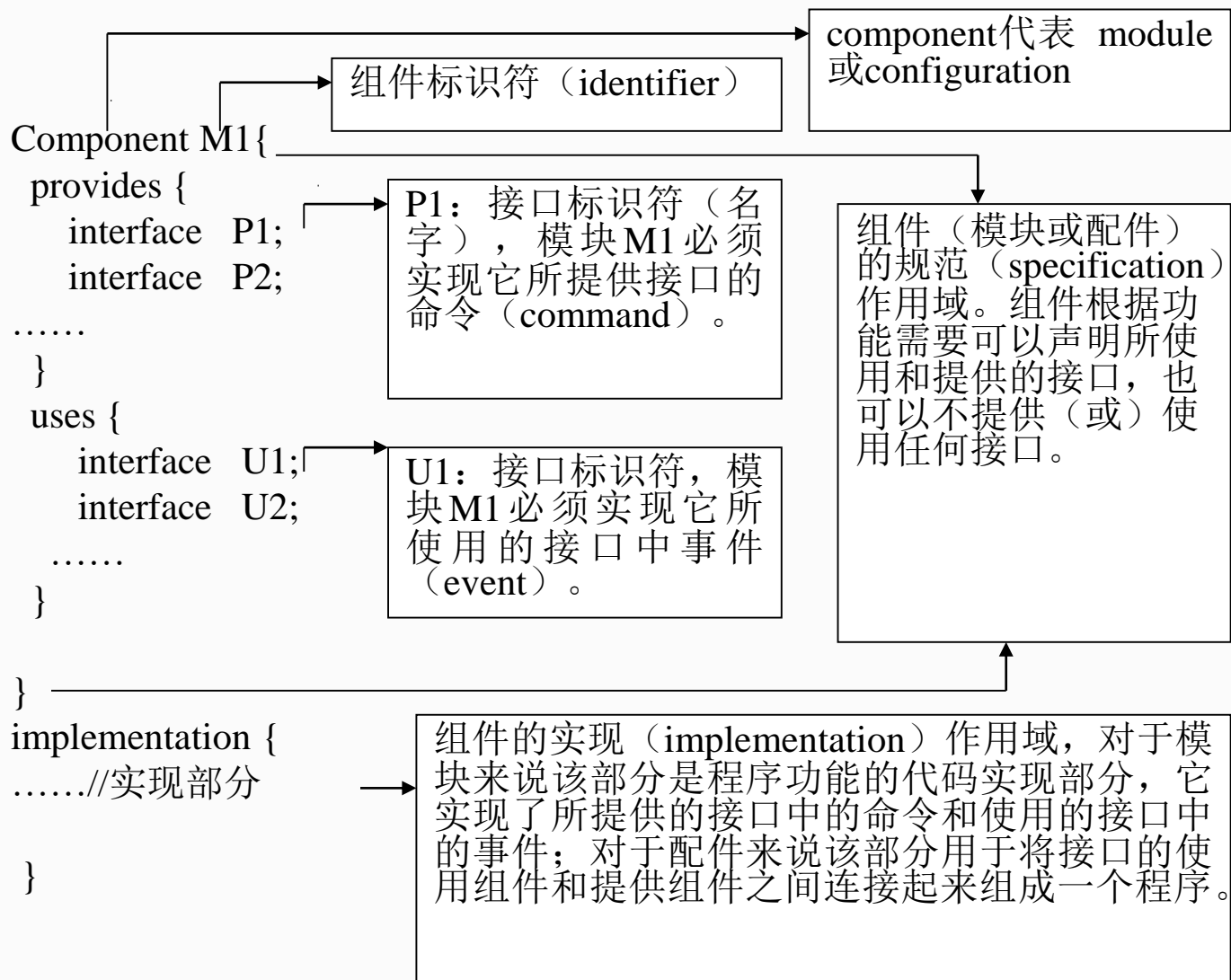




# 组件和接口

- 一个nesc编写的程序由一个或多个组件构成或连接而成。
- 一个组件由两部分组成：一个是规范说明，包含要用接口的名字；另一部分是它们的具体实现。
- 组件分两种：
  - Module组件（模块）：实现某种逻辑功能；
  - Configuration组件（配件）：将各个组件连接起来成为一个整体。
- 组件特征：组件内变量、函数可以自由访问，但组件之间不能访问和调用。

# 组件定义格式示例



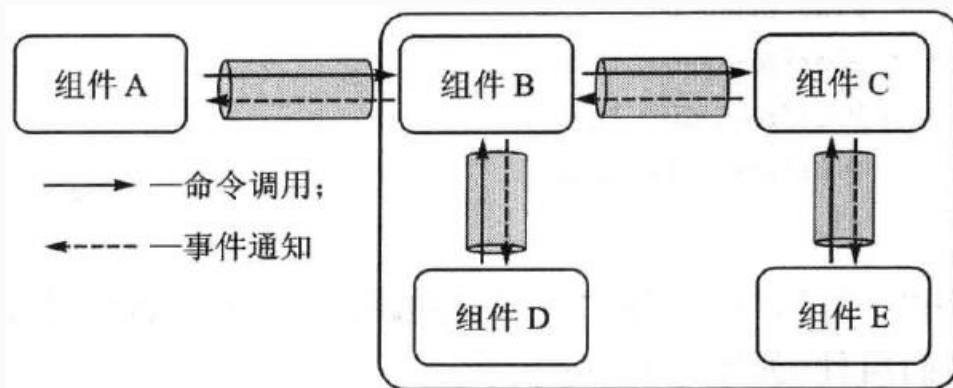


---

## 配件(configurations) 和模块(modules)

# 配件和模块

- 组件有两种：配件和模块
  - 模块（module）：提供一个或多个接口的实现
  - 配件（configuration）：把其他的组件装配起来，即把组件使用的接口连接到其提供者
- 任何一个nesC应用程序都是通过把一个或多个组件进行连接，从而成为一个可执行的、完整的程序



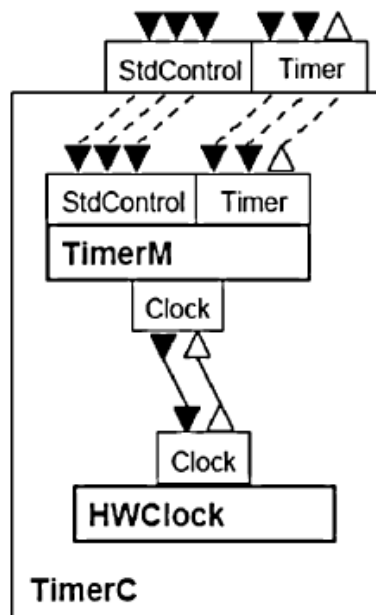
组件装配示意图

# Configuration

- TinyOS TimerC configuration
  - Configurations allow multiple components to be aggregated together into a single “super-component” that exposes a single set of interfaces.

Timer module

Hardware clock component



```
configuration TimerC {  
  provides {  
    interface StdControl;  
    interface Timer[uint8_t id];  
  }  
  implementation {  
    components TimerM, HWClock;  
  
    StdControl = TimerM.StdControl;  
    Timer = TimerM.Timer;  
  
    TimerM.Clk -> HWClock.Clock;  
  }  
}
```



# 配件和模块

---

- 每个nesC应用程序都必须有且只有一个顶层配件连接内部组件
- 组件设计分为 模块与配件 的目的：  
为了让系统设计者在构建应用程序的时候可以脱离组件的具体实现。



# 连接

- 配件将内部的各组件对应的接口连接在一起，这个操作称为连接（wiring），它将接口的提供者与接口的使用者关联起来。
- 要把一个接口连接到另一个接口时，一定要是同一类接口
- 连接操作使用 “->”来表示
- 格式：User组件.接口 -> Provider组件.接口

例如

- A.a -> B.b 意为 组件A的接口a连接到组件B的接口b
  - A 是接口的使用者(user)，而 B是接口的提供者(provider)
- BlinkC.Boot -> MainC.Boot;



# 接口连接

---

## 组件及连接举例：

- 模块组件（BlinkC.nc）
- 配置组件(BlinkAppC.nc)



# 接口连接

- 当一个组件只含有一个接口的时候，可以省略接口的名字。
  - LedsC组件只包含一个接口leds  
BnC.leds -> LedsC.leds  
等同于  
BnC.leds -> LedsC
- 连接的箭头是 可以对称倒反的
  - “->”和 “<-”作用相同  
User组件.接口->Provider组件.接口  
等同于  
Provider组件.接口<- User组件.接口



---

# 任务 (Tasks)



# 任务

---

- 命令和事件配合形成的代码是同步(sync)的。它以单一的前后执行顺序运行，没有任何形式的抢占
  - (1) 当同步代码开始运行后，直到完成前它不会释放CPU给其他的同步代码
  - (2) 如果一段同步代码运行时间很长，它会阻止其他同步代码运行，从而不利于系统的反应



# 任务

---

- 当需要一个组件去做一件事，但此时还有宽裕的时间，最好给TinyOS延迟计算的能力
  - 把任务安排在“处理完之前已在等待的事情”之后再执行
- 任务是一个函数，组件告诉TinyOS稍后再运行而不是立即运行



# 任务

---

- 在模块中声明任务的语法：

**task** void taskname(){.....}

说明：

taskname：任务名称。

void任务必须空返回

不能带有任何参数

- 派遣一个任务去执行，语法：

**post** taskname();

- 一个组件可以在命令、事件或者任务里派遣一个任务
- 一个任务可以安全地调用命令和触发事件

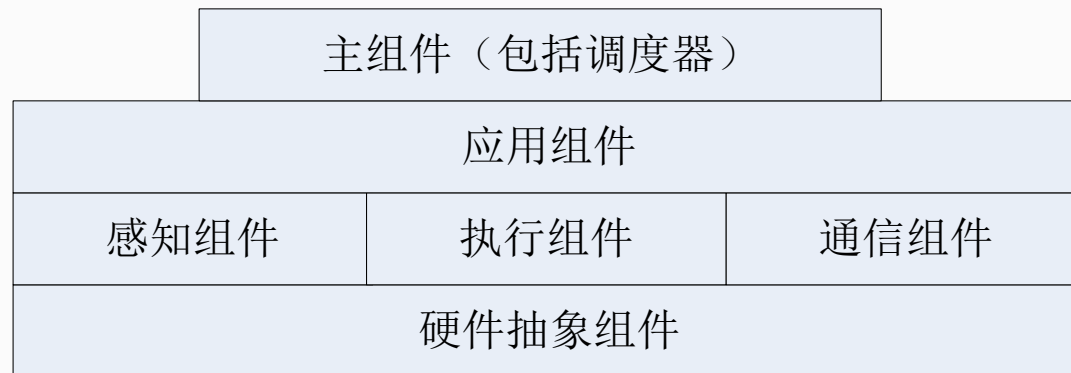


---

# 体系结构

# TinyOS的体系结构

- TinyOS操作系统采用了组件的结构。系统本身提供了一系列的组件供用户调用，其中包括主组件、应用组件、执行组件、传感组件、通信组件和硬件抽象组件，如下图所示：



- 组件由下到上可分为3类：硬件抽象组件、综合硬件组件和高层软件组件。
  - 硬件抽象组件将物理硬件映射到TinyOS的组件模型；
  - 综合硬件组件模拟高级的硬件行为，如感知组件、通信组件等；
  - 高层次的软件组件实现控制、路由以及数据传输等应用层的功能。



# TinyOS的应用程序

- 每个TinyOS程序应当具有至少一个应用组件，即用户组件。该应用组件通过接口调用下层组件提供的服务，实现针对特定应用的具体逻辑功能，如数据采集、数据处理、数据收发等。
- 一个完整的应用系统由一个内核调度器（简称调度器）和许多功能独立且相互联系的组件构成，可以把TinyOS系统和在其上运行的应用程序看成是一个大的“执行程序”。
- 现有的TinyOS系统提供了大多数传感网硬件平台和应用领域里都可用到的组件，例如定时器组件、传感器组件、消息收发组件、电源管理组件等，从而把用户和底层硬件隔离开来。在此基础上，用户只需开发针对特殊硬件和特殊应用需求的少量组件，大大提高了应用的开发效率。





# 启动顺序

---

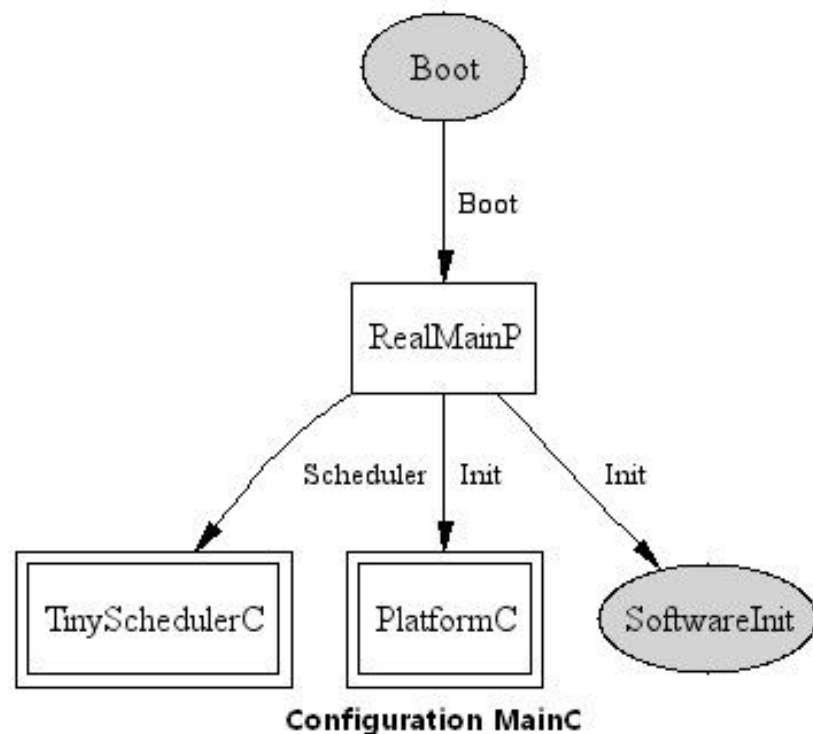
- `main()`函数在哪里？
- 应用程序处理Boot.booted事件，然后从此处开始运行。下面将介绍这个事件的前后过程，如何适宜地初始化组件。

# TinyOS 2.x的启动顺序

- TinyOS的启动顺序有以下4步：
- Step1：调度程序初始化；
- Step2：组件初始化；
- Step3：发送启动boot过程完成的信号；
- Step4：运行调度程序。

MainC组件（位于tos/system）是一个应用层级别的启动组件。

MainC组件提供了Boot接口，使用一个Init接口（SoftwareInit）。调用SoftwareInit.init()作为step 2的一部分，并在step 3触发Boot.booted()事件。





# Boot.booted()事件

---

- 一旦所有的初始化完成了，MainC的Boot.booted()事件就触发了。组件可以自由地调用start()命令以及其他组件使用的其他命令。
- 在应用程序里，定时器就是在booted()事件里启动的。这个booted事件就是TinyOS的main函数。

```
event void Boot.booted() {  
    call Timer0.startPeriodic(TIMER_PERIOD_MILLI);  
}
```



# 调度循环

---

- TinyOS 就会进入核心的调度循环（core scheduling loop）。只要有任务在排队，调度者就会继续运行。
- 一发现任务队伍为空，调度就会把微处理器调节到硬件资源允许的低能耗状态。
- 处理器进入休眠状态直到它碰到中断。当一个中断到达时，MCU退出休眠模式，运行中断程序。



# 总结

---

- TinyOS是一个开源的嵌入式操作系统，由加州大学伯利克分校开发出来，是基于一种组件（Component-Based）的架构方式，能够快速实现各种应用。
  - Limited resources: 核心系统代码和数据大概在400Bytes，还有编译优化。
  - Reactive Concurrency: split phase operation
  - Flexibility: fine-grained components; hardware/software transparency
  - Low-power: application-specific nature of TinyOS ensures no unnecessary functions consume energy; application-transparent CPU management; power management interfaces.



---

# Thank you!

Xiaolong Zheng

Email: [zhengxiaolong@bupt.edu.cn](mailto:zhengxiaolong@bupt.edu.cn)

Homepage: <https://xiaolongbupt.github.io/>