

2011

ZigBee 视频教程

应用开发指导

基于德州仪器 TI-CC2530 射频单片机芯片

锋硕电子科技有限公司

www.fuccesso.com.cn

2011/2/10



目录

第一章	绪论	4
1.1	ZigBee 定义	4
1.2	IEEE 802.15.4 标准概述	7
1.3	ZigBee 协议体系结构	8
第二章	ZigBee 基本概念	13
2.1	设备类型(Device Types)	13
2.1.1	Coordinator (协调器)	14
2.1.2	Router (路由器)	15
2.1.3	End-Device (终端设备)	15
2.2	协议栈规范 (Stack Profile)	15
2.3	拓扑结构	16
2.4	信标与非信标模式	18
2.5	地址	18
2.5.1	地址定义	18
2.5.2	网络地址分配	19
2.5.3	寻址	22
2.5.4	重要设备地址 (Important Device Addresses)	24
2.6	ZigBee 术语	25
2.6.1	属性	25
2.6.2	群集	25
2.6.3	设备描述	25
2.6.4	端点	26
2.6.5	节点	26
2.7	绑定 (binding)	27
2.8	路由(Routing)	28
2.8.1	概述 (Overview)	28
2.8.2	路由协议 (Routing Protocol)	28
2.8.3	路径的发现和选择 (Route Discovery and Selection)	29
2.8.4	路径保持维护 (Route maintenance)	30
2.8.5	路径期满 (Route expiry)	30
2.9	ZigBee 原语	31
第三章	Z-Stack 协议栈总体设计	33
3.1	任务初始化	34
3.2	任务调度	35
3.3	时间管理	38
3.4	原语通信	39
第四章	开发工具的安装及使用	43
4.1	IAR 安装	43
4.2	ZigBee 2007 协议栈安装	48
4.3	下载器硬件连接和驱动程序安装	51
第五章	ZigBee 开发套件	56
5.1	协调器节点	57

5.2 路由器节点	60
5.3 终端设备	63
5.4 外部接口	66
第六章 CC2530 概述	68
6.1 特性描述	68
6.2 应用范围	70
6.3 引脚描述	70
第七章 IAR 使用介绍	73

第一章 绪论

1.1 ZigBee 定义

物联网的定义是：通过射频识别（RFID）、红外感应器、全球定位系统、激光扫描器等信息传感设备，按约定的协议，把任何物体与互联网相连接，进行信息交换和通信，以实现物体的智能化识别、定位、跟踪、监控和管理的一种网络。

无线传感网络的定义是：大规模、无线、自组织、多跳、无分区、无基础设施支持的网络。其中的节点是同构的、成本较低、体积较小，大部分节点不移动，被随意撒布在工作区域，要求网络系统有尽可能长的工作时间。在通信方式上，虽然可以采用有线、无线、红外和光等多种形式，但一般认为短距离的无线低功率通信技术最适合传感器网络使用，为明确起见，一般称无线传感器网络(WSN, Wireless Sensor Network)。

Zigbee 是 IEEE 802.15.4 协议的代名词。根据这个协议规定的技术是一种短距离、低功耗的无线通信技术。这一名称来源于蜜蜂的八字舞，由于蜜蜂(bee)是靠飞翔和“嗡嗡”(zig)地抖动翅膀的“舞蹈”来与同伴传递花粉所在方位信息，也就是说蜜蜂依靠这样的方式构成了群体中的通信网络。其特点是近距离、低复杂度、自组织、低功耗、低数据速率、低成本。主要适用于自动控制和远程控制领域，可以嵌入各种设备。简而言之，ZigBee 就是一种便宜的，低功耗的近距离无线组网通讯技术。

无线传感网络的无线通信技术可以采用 ZigBee 技术、蓝牙、Wi-Fi 和红外等技术。ZigBee 技术是一种短距离、低复杂度、低功耗、低数据速率、低成本的双向无线通信技术或无线网络技术，是一组基于 IEEE802.15.4 无线标准研制开发的组网、安全和应用软件方面的通信技术。

协议栈是指网络中各层协议的总和，其形象的反映了一个网络中文件传输的过程：由上层协议到底层协议，再由底层协议到上层协议。使用最广泛的是英特网协议栈，由上到下的协议

分别是：应用层（HTTP，TELNET，DNS，EMAIL 等），运输层（TCP，UDP），网络层（IP），链路层（WI-FI，以太网，令牌环，FDDI 等），物理层。

ZigBee 联盟于 2005 年公布了第一份 ZigBee 规范“ZigBee Specification V1.0”。ZigBee 协议规范使用了 IEEE 802.15.4 定义的物理层（PHY）和媒体介质访问层（MAC），并在此基础上定义了网络层（NWK）和应用层（APL）架构。

ZigBee2007/PRO 无线传感器网络与 ZigBee2006 无线传感器网络相比最大区别在于其支持最新 ZigBee2007/PRO 网络，提供更多更精确传感器(如增加高精度温湿度数字传感器等)，提供更多可扩展接口，提供更大网络支持，速度更快/处理能力更强低功耗微控制器等。

ZigBee 的技术特性决定它将是无线传感器网络的最好选择，广泛用于物联网，自动控制和监视等诸多领域。以美国德州仪器 TI 公司 CC2430/CC2530 芯片为代表的 Zigbee SOC 解决方案在国内高校企业掀起了一股 Zigbee 技术应用的热潮。CC2430/CC2530 集成了 51 单片机内核，相比于众多的 Zigbee 芯片，CC2430/CC2530 颇受青睐。

ZigBee 新一代 SOC 芯片 CC2530 是真正的片上系统解决方案，支持 IEEE 802.15.4 标准 /ZigBee/ZigBee RF4CE 和能源的应用。拥有庞大的快闪记忆体多达 256 个字节，CC2530 是理想 ZigBee 专业应用。CC2530 结合了一个完全集成的，高性能的 RF 收发器与一个 8051 微处理器，8 kB 的 RAM，32/64/128/256 KB 闪存，以及其他强大的支持功能和外设。

CC2530 提供了 101dB 的链路质量，优秀的接收器灵敏度和健壮的抗干扰性，四种供电模式，多种闪存尺寸，以及一套广泛的外设集——包括 2 个 USART、12 位 ADC 和 21 个通用 GPIO，以及更多。除了通过优秀的 RF 性能、选择性和业界标准增强 8051MCU 内核，支持一般的低功耗无线通信，CC2530 还可以配备 TI 的一个标准兼容或专有的网络协议栈（RemoTI，Z-Stack，或 SimpliciTI）来简化开发，使你更快的获得市场。CC2530 可以用于的应用包括远程控制、消费型电子、家庭控制、计量和智能能源、楼宇自动化、医疗以及更多领域。

德州仪器 (TI) 宣布推出领先的 [ZigBee](#) 认证 [Z-Stack](#) 软件的最新版 (可通过

<http://focus.ti.com.cn/cn/docs/toolsw/folders/print/z-stack.html>

免费下载)。ZStack-CC2530-2.3.1-1.4.0 软件全面支持 ZigBee 与 ZigBee PRO 特性集并符合最新智能能源规范, 非常适用于高级电表架构 ([AMI](#))。

ZStack-CC2530-2.3.1-1.4.0 软件可与 TI 的 SmartRF 05 平台协同工作, 该平台包括 MSP430 超低功耗[微控制器](#) (MCU)、CC2520 RF 收发器以及 CC2591 距离扩展器, 通信连接距离可达数公里。该软件提供了其所支持的应用范例库, 其中包括智能能源、家庭自动化以及无线下载 (OAD) 等功能。

TI 推出了最丰富、最完整的 ZigBee 系列产品, 包括收发器、片上系统以及新近推出的 2.4GHz 距离扩展器 (CC2591)。该系列产品具有无与伦比的高性能、高灵活性与定制功能, 从而有助于客户提供特色化设计方案。TI 还为其不断丰富的低功耗 RF 系列产品提供一流的软件、工具、应用知识及全球技术支持, 帮助 ZigBee 设计人员在市场中取得成功。

Z-Stack 软件因其出色的 ZigBee 与 ZigBee PRO 特性集被 ZigBee 测试机构国家技术服务公司 (NTS) 评为 ZigBee 联盟最高业内水平, 目前该软件已为全球数以千计的 ZigBee 开发人员广泛采用。Z-Stack 还可为 CC2430 片上系统以及带硬件定位检测引擎的 CC2431 提供支持, 从而使 ZigBee 应用能根据节点所处的当前位置改变行为。

Z-Stack 是在 2007 年 4 月, 德州仪器推出业界领先的 ZigBee 协议栈, Z-Stack 符合 ZigBee 2006 规范, 支持多种平台, Z-Stack 包含了网状网络拓扑的几近于全功能的协议栈, 在竞争激烈的 ZigBee 领域占有重要地位。配合 OSAL 完成整个协议栈的运行。

Z-Stack 只是 ZigBee 协议的一种具体的实现, 我们要澄清的是 ZigBee 不仅仅有 Z-Stack 这一种, 也不能把 Z-Stack 等同于 ZigBee 协议, 现在也有好几个真正开源的 ZigBee 协议栈, 例如: msstatePAN 协议栈, freakz 协议栈, 这些都是 ZigBee 协议的具体实现, 而且是全部真正的开源

的，它们的所有源代码我们都可以看到，而 Z-Stack 中的很多关键的代码是以库文件的形式给出来，也就是我们只能用它们，而看不到它们的具体的实现。其中核心部分的代码都是编译好的，以库文件的形式给出的，比如安全模块，路由模块，和 Mesh 自组网模块。那些真正开源的 ZigBee 协议栈没有大的商业公司的支持，开发升级方面，性能方面和 TI 公司的还是有很大差距。

1.2 IEEE 802.15.4 标准概述

IEEE 802.15.4 是一个低速率无线个人局域网(Low Rate Wireless PersonalArea Networks, LR-WPAN)标准。该标准定义了物理层(PHY)和介质访问控制层(MAC)。这种低速率无线个人局域网的网络结构简单、成本低廉、具有有限的功率和灵活的吞吐量。低速率无线个人局域网的主要目标是实现安装容易、数据传输可靠、短距离通信、极低的成本、合理的电池寿命，并且拥有一个简单而且灵活的通信网络协议。

LR-WPAN 网络具有如下特点：

- ◆ 实现 250kb/s, 40kb/s, 20kb/s 三种传输速率。
- ◆ 支持星型或者点对点两种网络拓扑结构。
- ◆ 具有 16 位短地址或者 64 位扩展地址。
- ◆ 支持冲突避免载波多路侦听技术(carrier sense multiple access with collision avoidance, CSMA-CA)。
- ◆ 用于可靠传输的全应答协议。
- ◆ 低功耗。
- ◆ 能量检测(Energy Detection, ED)。
- ◆ 链路质量指示(Link Quality Indication, LQI)。

◆ 在 2450MHz 频带内定义了 16 个通道；在 915MHz 频带内定义了 10 个通道；在 868MHz 频带内定义了 1 个通道。

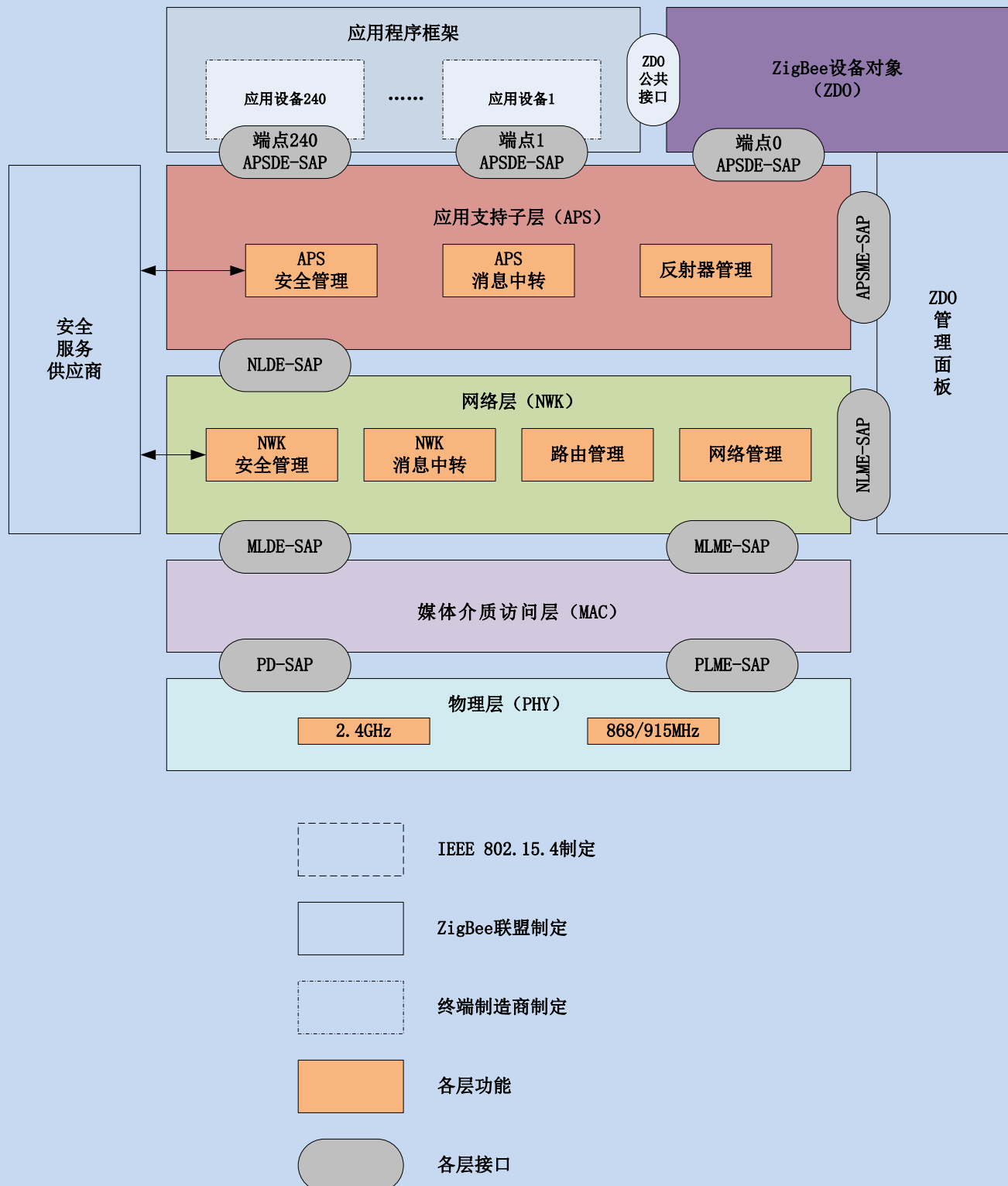
为了使供应商能够提供最低可能功耗的设备，IEEE(Institute ofElectrical and Electronics Engineers，电气及电子工程师学会)定义了两种不同类型的设备：一种是完整功能设备(full. functional device, FFD)，另一种是简化功能设备(reduced. functional device, RFD)。

1.3 ZigBee 协议体系结构

ZigBee 协议栈建立在 IEEE 802. 15. 4 的 PHY 层和 MAC 子层规范之上。它实现了网络层(networklayer, NWK)和应用层(applicationlayer, APL)。在应用层内提供了应用支持子层(application support sub—layer, APS)和 ZigBee 设备对象(ZigBee Device Object, ZDO)。应用框架中则加入了用户自定义的应用对象

ZigBee 的体系结构由称为层的各模块组成。每一层为其上层提供特定的服务：即由数据服务实体提供数据传输服务；管理实体提供所有的其他管理服务。每个服务实体通过相应的服务接入点(SAP)为其上层提供一个接口，每个服务接入点通过服务原语来完成所对应的功能。ZigBee 协议的体系结构如下图所示：

ZigBee协议体系结构



➤ 物理层 (PHY)

物理层定义了物理无线信道和 MAC 子层之间的接口，提供物理层数据服务和

物理层管理服务。

物理层内容：

- 1) ZigBee 的激活；
- 2) 当前信道的能量检测；
- 3) 接收链路服务质量信息；
- 4) ZigBee 信道接入方式；
- 5) 信道频率选择；
- 6) 数据传输和接收。

➤ 介质接入控制子层 (MAC)

MAC 层负责处理所有的物理无线信道访问，并产生网络信号、同步信号；支持 PAN 连接和分离，提供两个对等 MAC 实体之间可靠的链路。

MAC 层功能：

- 1) 网络协调器产生信标；
- 2) 与信标同步；
- 3) 支持 PAN(个域网)链路的建立和断开；
- 4) 为设备的安全性提供支持；
- 5) 信道接入方式采用免冲突载波检测多址接入(CSMA-CA)机制；
- 6) 处理和维持保护时隙(GTS)机制；
- 7) 在两个对等的 MAC 实体之间提供一个可靠的通信链路。

➤ 网络层 (NWK)

ZigBee 协议栈的核心部分在网络层。网络层主要实现节点加入或离开网络、接收或抛弃其他节点、路由查找及传送数据等功能。

网络层功能：

- 1) 网络发现；
- 2) 网络形成；
- 3) 允许设备连接；
- 4) 路由器初始化；
- 5) 设备同网络连接；
- 6) 直接将设备同网络连接；
- 7) 断开网络连接；
- 8) 重新复位设备；
- 9) 接收机同步；
- 10) 信息库维护。

➤ 应用层 (APL)

ZigBee 应用层框架包括应用支持层(APS)、ZigBee 设备对象(ZDO)和制造商所定义的应用对象。

应用支持层的功能包括：维持绑定表、在绑定的设备之间传送消息。

ZigBee 设备对象的功能包括：定义设备在网络中的角色(如 ZigBee 协调器和终端设备)，发起和响应绑定请求，在网络设备之间建立安全机制。ZigBee 设备对象还负责发现网络中的设备，并且决定向他们提供何种应用服务。

ZigBee 应用层除了提供一些必要函数以及为网络层提供合适的服务接口外，一个重要的功能是应用者可在这层定义自己的应用对象。

➤ 应用程序框架 (AF)：

运行在 ZigBee 协议栈上的应用程序实际上就是厂商自定义的应用对象，并且

遵循规范（profile）运行在端点 1~240 上。在 ZigBee 应用中，提供 2 种标准服务类型：键值对（KVP）或报文（MSG）

➤ **ZigBee 设备对象（ZDO）：**

ZigBee 设备对象(ZDO)的功能包括负责定义网络中设备的角色,如:协调器或者终端设备。还包括对绑定请求的初始化或者响应,在网络设备之间建立安全联系等。实现这些功能,ZDO 使用 APS 层的 APSDE-SAP 和网络层的 NLME-SAP。ZDO 是特殊的应用对象,它在端点(entire)0 上实现。远程设备通过 ZDO 请求描述符信息,接收到这些请求时,ZDO 会调用配置对象获取相应描述符值。

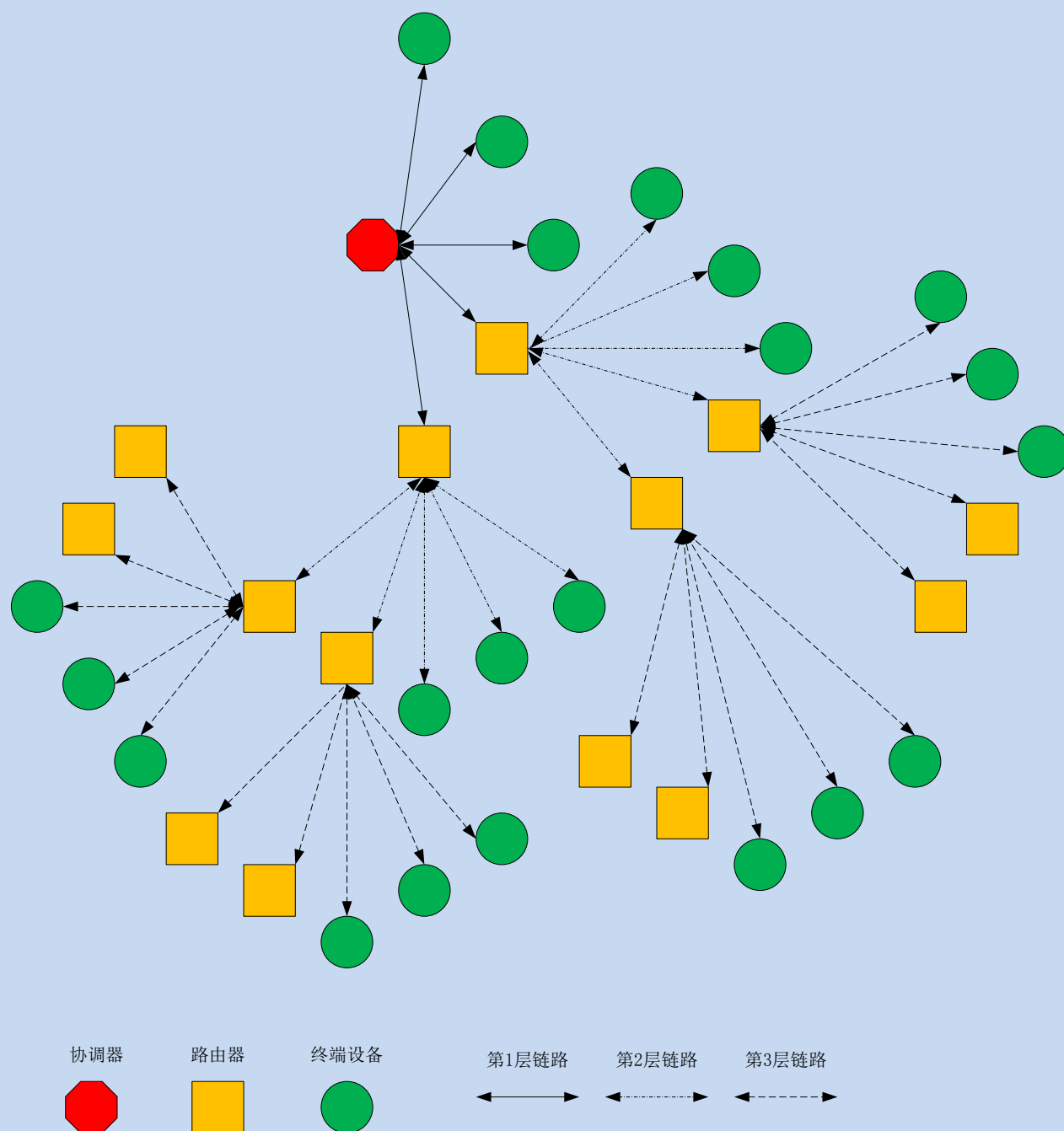
第二章 ZigBee 基本概念

2.1 设备类型(Device Types)

在 ZigBee 网络中存在三种逻辑设备类型：Coordinator(协调器)，Router(路由器)和 End-Device(终端设备)。ZigBee 网络由一个 Coordinator 以及多个 Router 和多个 End_Device 组成。

注意：在 ZStack-CC2530-2.3.1-1.4.0 中一个设备的类型通常在编译的时候通过编译选项确定。所有的应用例子都提供独立的项目文件来编译每一种设备类型。对于协调器，在 Workspace 区域的下拉菜单中选择 CoordinatorEB-Pro；对于路由器，在 Workspace 区域的下拉菜单中选择 RouterEB-Pro；对于终端设备，在 Workspace 区域的下拉菜单中选择 EndDeviceEB-Pro。

节点类型	.cfg 配置文件
协调器	-DZDO_COORDINATOR -DRTR_NWK
路由器	-DRTR_NWK
终端设备	空



上图是一个简单的 ZigBee 网络示意图。其中红色节点为 Coordinator，黄色节点为 Router，绿色节点为 End-Device。

2.1.1 Coordinator (协调器)

协调器负责启动整个网络。它也是网络的第一个设备。协调器选择一个信道和一个网络

ID(也称之为 PAN ID, 即 Personal Area Network ID), 随后启动整个网络。协调器也可以用来协助建立网络中安全层和应用层的绑定(bindings)。

注意, 协调器的角色主要涉及网络的启动和配置。一旦这些都完成后, 协调器的工作就像一个路由器(或者消失 go away)。由于 ZigBee 网络本身的分布特性, 因此接下来整个网络的操作就不在依赖协调器是否存在。

2.1.2 Router (路由器)

路由器的功能主要是: 允许其他设备加入网络, 多跳路由和协助它自己的由电池供电的终端设备的通讯。

通常, 路由器希望是一直处于活动状态, 因此它必须使用主电源供电。但是当使用树状网络拓扑结构时, 允许路由间隔一定的周期操作一次, 这样就可以使用电池给其供电。

2.1.3 End-Device (终端设备)

终端设备没有特定的维持网络结构的责任, 它可以睡眠或者唤醒, 因此它可以是一个电池供电设备。通常, 终端设备对存储空间(特别是 RAM 的需要)比较小。

2.2 协议栈规范 (Stack Profile)

协议栈规范由 ZigBee 联盟定义指定。在同一个网络中的设备必须符合一个协议栈规范(同一个网络中所有设备的协议栈规范必须一致)。

ZigBee 联盟为 ZigBee 协议栈 2007 定义了 2 个规范: ZigBee 和 ZigBee PRO。所有的设备只要遵循该规范, 即使在不同厂商买的不同设备同样可以形成网络。

如果应用开发者改变了规范, 那么他的产品将不能与遵循 ZigBee 联盟定义规范的产品组成网络, 也就是说该开发者开发的产品具有特殊性, 我们称之为“关闭的网络”, 也就是说它的

设备只有在自己的产品中使用，不能与其他产品通信。更改后的规范可以称之为“特定网络”规范。

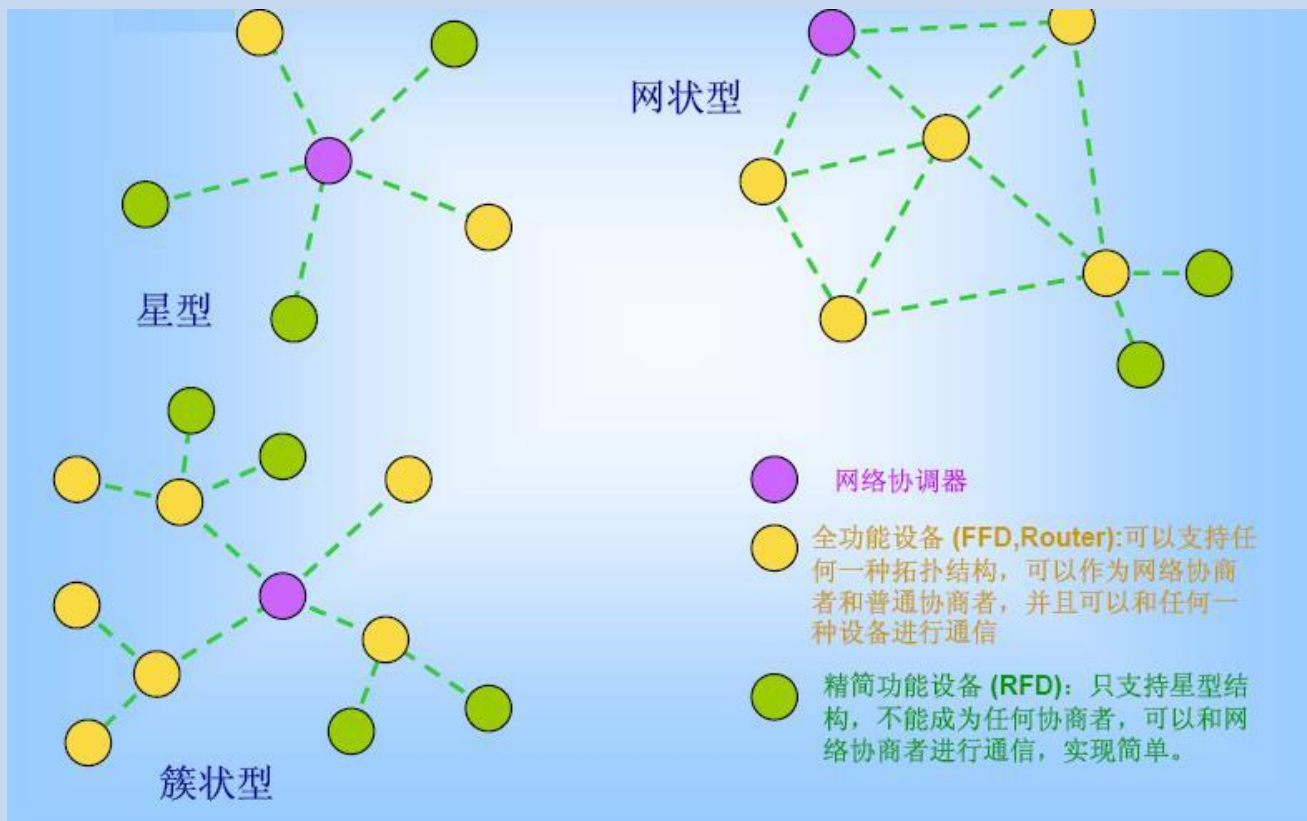
协议栈规范的 ID 号可以通过查询设备发送的 beacon 帧获得。在设备加入网络之前，首先需要确认协议栈规范的 ID。“特定网络”规范 ID 号为 0；ZigBee 协议栈规范的 ID 号为 1；ZigBee PRO 协议栈规范的 ID 号为 2。协议栈规范的 ID（STACK_PROFILE_ID）在 `nwk_globals.h` 中定义：

```
#define NETWORK_SPECIFIC      0
#define HOME_CONTROLS        1
#define ZIGBEEPRO_PROFILE    2
#define GENERIC_STAR          3
#define GENERIC_TREE          4

#if defined ( ZIGBEEPRO )
    #define STACK_PROFILE_ID    ZIGBEEPRO_PROFILE
#else
    #define STACK_PROFILE_ID    HOME_CONTROLS
#endif
```

2.3 拓扑结构

ZigBee 网络支持星状、树状和网状三种网络拓扑结构，如下图所示，分别依次是星状网络，树（簇）状网络和网状网络。



星状网络由一个 PAN 协调器和多个终端设备组成，只存在 PAN 协调器与终端的通讯，终端设备间的通讯都需通过 PAN 协调器的转发。

树状网络由一个协调器和一个或多个星状结构连接而成，设备除了能与自己的父节点或子节点进行点对点直接通讯外，其他只能通过树状路由完成消息传输。

网状网络是树状网络基础上实现的，与树状网络不同的是，它允许网络中所有具有路由功能的节点直接互连，由路由器中的路由表实现消息的网状路由。该拓扑的优点是减少了消息延时，增强了可靠性，缺点是需要更多的存储空间开销。

在 Z-Stack 中网络拓扑结构定义如下：

```
#define NWK_MODE_STAR      0
#define NWK_MODE_TREE     1
#define NWK_MODE_MESH     2

#if ( STACK_PROFILE_ID == ZIGBEEPRO_PROFILE )
    #define NWK_MODE      NWK_MODE_MESH
#elif ( STACK_PROFILE_ID == HOME_CONTROLS )
    #define NWK_MODE      NWK_MODE_MESH
```

```
#elif ( STACK_PROFILE_ID == GENERIC_STAR )  
    #define NWK_MODE                NWK_MODE_STAR  
#elif ( STACK_PROFILE_ID == NETWORK_SPECIFIC )  
    #define NWK_MODE                NWK_MODE_MESH  
#endif
```

2.4 信标与非信标模式

ZigBee 网络的工作模式可以分为信标（Beacon）和非信标（Non-beacon）两种模式。信标模式实现了网络中所有设备的同步工作和同步休眠，以达到最大限度的功耗节省，而非信标模式则只允许终端设备进行周期性休眠，协调器 和所有路由器 设备必须长期处于工作状态。

信标模式下，协调器负责以一定的间隔时间（一般在 15ms-4mins 之间）向网络广播信标帧，两个信标帧发送间隔之间有 16 个相同的时槽，这些时槽分为网络休眠区和网络活动区两个部分，消息只能在网络活动区的各时槽内发送。

非信标模式下，ZigBee 标准采用父节点为终端设备子节点缓存数据，终端设备主动向其父节点提取数据的机制，实现终端设备的周期性（周期可设置）休眠。网络中所有父节点需为自己的终端设备子节点缓存数据帧，所有终端设备子节点的大多数时间都处于休眠模式，周期性的醒来与父节点握手以确认自己仍处于网络中，其从休眠模式转入数据传输模式一般只需要 15ms。

2.5 地址

2.5.1 地址定义

ZigBee 设备有两种类型的地址。一种是 64 位 IEEE 地址，即 MAC 地址，另一种是 16 位网络地址。

64 位地址使全球唯一的地址，设备将在它的生命周期中一直拥有它。它通常由制造商或者

被安装时设置。这些地址由 IEEE 来维护和分配。

16 位网络地址是当设备加入网络后分配的。它在网络中是唯一的，用来在网络中鉴别设备和发送数据。其中，协调器的网络地址为 0x00

```
#define NWK_PAN_COORD_ADDR 0x0000
```

2.5.2 网络地址分配

ZigBee2006 和 ZigBee 2007 使用分布式寻址方案来分配网络地址。这个方案保证在整个网络中所有分配的地址是唯一的。这一点是必须的，因为这样才能保证一个特定的数据包能够发给它指定的设备，而不出现混乱。同时，这个寻址算法本身的分布特性保证设备只能与他的父辈设备通讯来接受一个网络地址。不需要整个网络范围内通讯的地址分配，这有助于网络的可测量性。

假设父设备可拥有的最大子设备数为 C_m ，其拥有的最大路由子设备数为 R_m ，网络的最大深度为 L_m ，则父设备所能分配子区段地址数为：

若 $R_m=1$ ， $C_{skip}(d) = 1 + C_m * (L_m - d - 1)$ ；

若 R_m 不为 1，则 $C_{skip}(d) = (1 + C_m - R_m - C_m * (R_m)^{(L_m - d - 1)}) / (1 - R_m)$ 。

子节点为父设备的第 n 个子路由器的短地址分配：

$$A_{child} = A_{parent} + (n-1) * C_{skip}(d) + 1, n=1$$
$$A_{child} = A_{parent} + (n-1) * C_{skip}(d), n>1$$

子节点为父设备的第 n 个子终端设备的短地址分配：

$$A_{child} = A_{parent} + R_m * C_{skip}(d) + n$$

ZigBee 2007 PRO 使用的随机地址分配机制，对新加入的节点使用随机地址分配，为保证网络内地址分配不重复，使用其余的随机地址再进行分配。当一个节点加入时，将接收到父节点的随机分配地址，然后产生“设备声明”（包含分配到的网络地址和 IEEE 地址）发送至网

络中的其余节点。如果另一个节点有着同样的网络地址，则通过路由器广播“网络状态-地址冲突”至网络中的所有节点。所有发生网络地址冲突的节点更改自己的网络地址，然后再发起“设备声明”检测新的网络地址是否冲突。

终端设备不会广播“地址冲突”，他们的父节点会帮助完成。如果一个终端设备发生了“地址冲突”，他们的父节点发送“重新加入”消息至终端设备，并要求他们更改网络地址。然后，终端设备再发起“设备声明”检测新的网络地址是否冲突。

当接收到“设备声明”后，关联表和绑定表将被更新使用新的网络地址，但是路由表不会被更新。

在每个路由加入网络之前，寻址方案需要知道和配置一些参数。这些参数是 MAX_DEPTH（最大网络深度）、MAX_ROUTERS（最多路由数）和 MAX_CHILDREN（最多子节点数）。这些参数是栈配置的一部分，ZigBee2007 协议栈已经规定了这些参数的值：

```
#if ( STACK_PROFILE_ID == ZIGBEEPRO_PROFILE )
    #define MAX_NODE_DEPTH      20
#elif ( STACK_PROFILE_ID == HOME_CONTROLS )
    #define MAX_NODE_DEPTH      5
#elif ( STACK_PROFILE_ID == GENERIC_STAR )
    #define MAX_NODE_DEPTH      5
#elif ( STACK_PROFILE_ID == NETWORK_SPECIFIC )
    #define MAX_NODE_DEPTH      5
#endif

#define NWK_MAX_ROUTERS        6

#define NWK_MAX_DEVICES        21
```

MAX_DEPTH 决定了网络的最大深度。协调器(Coordinator)位于深度 0，它的儿子位于深度 1，他的儿子的儿子位于深度 2，以此类推。MAX_DEPTH 参数限制了网络在物理上的长度。

MAX_CHILDREN 决定了一个路由(Router)或者一个协调器节点可以处理的儿子节点的最大个数。

MAX_ROUTER 决定了一个路由(Router)或者一个协调器(Coordinator)节点可以处理的具有路由功能的儿子节点的最大个数。这个参数是 MAX_CHILDREN 的一个子集，终端节点使用 (MAX_CHILDREN - MAX_ROUTER)剩下的地址空间。

本章中 ZigBee 网络示意图所示的 MAX_DEPTH=3，MAX_CHILDREN=5，MAX_ROUTER=2。

如果开发人员想改变这些值，则需要完成以下几个步骤：

首先，你要保证这些参数新的赋值要合法。即，整个地址空间不能超过 2^{16} ，这就限制了参数能够设置的最大值。可以使用 projects\ZStack\tools 文件夹下的 CSkip.xls 文件来确认这些值是否合法。当在表格中输入了这些数据后，如果你的数据不合法的话就会出现错误信息。

当选择了合法的数据后，开发人员还要保证不再使用标准的栈配置，取而代之的是网络自定义栈配置(例如：在 nwk_globals.h 文件中将 STACK_PROFILE_ID 改为 NETWORK_SPECIFIC)。然后 nwk_globals.h 文件中的 MAX_DEPTH 参数将被设置为合适的值。

此外，还必须设置 nwk_globals.c 文件中的 Cskipchldrn 数组和 CskipRtrs 数组。这些数组的值由 MAX_CHILDREN 和 MAX_ROUTER 构成。

```
#if ( STACK_PROFILE_ID == ZIGBEEPRO_PROFILE )
    uint8 CskipRtrs[1] = {0};
    uint8 CskipChldrn[1] = {0};
#elif ( STACK_PROFILE_ID == HOME_CONTROLS )
    uint8 CskipRtrs[MAX_NODE_DEPTH+1] = {6,6,6,6,6,0};
    uint8 CskipChldrn[MAX_NODE_DEPTH+1] = {20,20,20,20,20,0};
#elif ( STACK_PROFILE_ID == GENERIC_STAR )
    uint8 CskipRtrs[MAX_NODE_DEPTH+1] = {5,5,5,5,5,0};
    uint8 CskipChldrn[MAX_NODE_DEPTH+1] = {5,5,5,5,5,0};
#elif ( STACK_PROFILE_ID == NETWORK_SPECIFIC )
    uint8 CskipRtrs[MAX_NODE_DEPTH+1] = {5,5,5,5,5,0};
    uint8 CskipChldrn[MAX_NODE_DEPTH+1] = {5,5,5,5,5,0};
#endif // STACK_PROFILE_ID
```

2.5.3 寻址

为了向一个在 ZigBee 网络中的设备发送数据，应用程序通常使用 AF_DataRequest()函数。

数据包将要发送给一个 afAddrType_t(在 ZComDef.h 中定义)类型的目标设备。

```
typedef struct
{
    union
    {
        uint16_t shortAddr;
        ZLongAddr_t extAddr;
    } addr;
    afAddrMode_t addrMode;
    byte endPoint;
    uint16_t panId; // used for the INTER_PAN feature
} afAddrType_t;
```

注意，除了网路地址之外，还要指定地址模式参数。目的地址模式可以设置为以下几个值：

```
typedef enum
{
    afAddrNotPresent = AddrNotPresent,
    afAddr16Bit      = Addr16Bit,
    afAddr64Bit      = Addr64Bit,
    afAddrGroup       = AddrGroup,
    afAddrBroadcast  = AddrBroadcast
} afAddrMode_t;
```

因为在 Zigbee 中，数据包可以单点传送(unicast)，多点传送(multicast)或者广播传送，所以必须有地址模式参数。一个单点传送数据包只发送给一个设备，多点传送数据包则要传送给一组设备，而广播数据包则要发送给整个网络的所有节点。这个将在下面详细解释。

- 单点传送(Unicast)

Uicast 是标准寻址模式，它将数据包发送给一个已经知道网络地址的网络设备。将 afAddrMode 设置为 Addr16Bit 并且在数据包中携带目标设备地址。

- 间接传送(Indirect)

当应用程序不知道数据包的目标设备在哪里的时候使用的模式。将模式设置为

AddrNotPresent 并且目标地址没有指定。取代它的是从发送设备的栈的绑定表中查找目标设备。这种特点称之为源绑定。

当数据向下发送到达栈中，从绑定表中查找并且使用该目标地址。这样，数据包将被处理成为一个标准的单点传送数据包。如果在绑定表中找到多个设备，则向每个设备都发送一个数据包的拷贝。

上一个版本的 ZigBee(ZigBee2004)，有一个选项可以讲绑定表保存在协调器(Coordinator)当中。发送设备将数据包发送给协调器，协调器查找它栈中的绑定表，然后将数据发送给最终的目标设备。这个附加的特性叫做协调器绑定(Coordinator Binding)。

● 广播传送(broadcast)

当应用程序需要将数据包发送给网络的每一个设备时，使用这种模式。地址模式设置为 AddrBroadcast。目标地址可以设置为下面广播地址的一种：

NWK_BROADCAST_SHORTADDR_DEVALL(0xFFFF)——数据包将被传送到网络上的所有设备，包括睡眠中的设备。对于睡眠中的设备，数据包将被保留在其父亲节点直到查询到它，或者消息超时(NWK_INDIRECT_MSG_TIMEOUT 在 f8wConifg.cfg 中)。

NWK_BROADCAST_SHORTADDR_DEVRXON(0xFFFD)——数据包将被传送到网络上的所有在空闲时打开接收的设备(RXONWHENIDLE)，也就是说，除了睡眠中的所有设备。

NWK_BROADCAST_SHORTADDR_DEVZCZR(0xFFFC)——数据包发送给所有的路由器，包括协调器。

● 组寻址(Group Addressing)

当应用程序需要将数据包发送给网络上的一组设备时，使用该模式。地址模式设置为 afAddrGroup 并且 addr.shortAddr 设置为组 ID。

在使用这个功能呢之前，必须在网络中定义组。(参见 Z-stack API 文档中的 aps_AddGroup()函数)

数)。

注意组可以用来关联间接寻址。再绑定表中找到的目标地址可能是单点传送或者是一个组地址。另外，广播发送可以看做是一个组寻址的特例。

下面的代码是一个设备怎样加入到一个 ID 为 1 的组当中：

```
aps_Group_t group;  
// Assign yourself to group 1  
group.ID = 0x0001;  
group.name[0] = 0; // This could be a human readable string  
aps_AddGroup( SAMPLEAPP_ENDPOINT, &group );
```

2.5.4 重要设备地址(Important Device Addresses)

应用程序可能需要知道它的设备地址和父亲地址。使用下面的函数获取设备地址(在 ZStack API 中定义)：

NLME_GetShortAddr()——返回本设备的 16 位网络地址

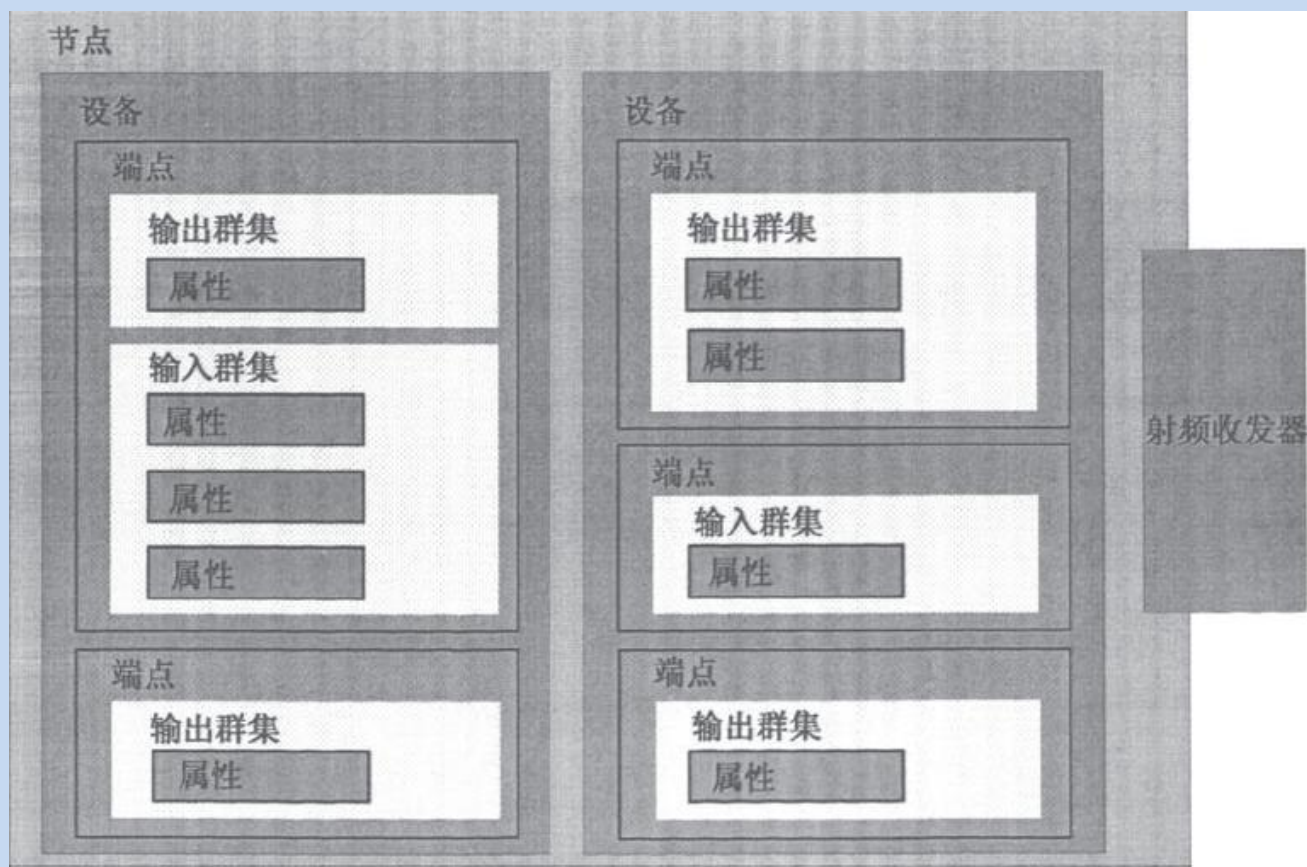
NLME_GetExtAddr()—— 返回本设备的 64 位扩展地址

使用下面的函数获取该设备的父亲设备的地址：

NLME_GetCoordShortAddr()——返回本设备的父亲设备的 16 位网络地址

NLME_GetCoordExtAddr()—— 返回本设备的父亲设备的 64 位扩展地址

2.6 ZigBee 术语



2.6.1 属性

属性 Attribute 是一个反映物理数量或状态的数据值，比如开关值 (On/Off) ,温度值、百分比等。

2.6.2 群集

群集 Cluster 是包含一个或多个属性 (attribute) 的群组。简单的说，群集就是属性的集合。每个群集都被分配一个唯一的群集 ID 且每个群集最多有 65536 个属性。

2.6.3 设备描述

设备描述 Device Description 是指一个大型目标应用的一部分，包括一个或多个群集，并且

指定群集是输入还是输出。描述符有：节点描述符、电源描述符、简单描述符、端点描述符。

端点描述符：

```
typedef struct
{
    byte endPoint;
    byte *task_id; // Pointer to location of the Application task ID.
    SimpleDescriptionFormat_t *simpleDesc;
    afNetworkLatencyReq_t latencyReq;
} endPointDesc_t;
```

简单描述符：

```
typedef struct
{
    byte          EndPoint;
    uint16        AppProfId;
    uint16        AppDevicId;
    byte          AppDevVer:4;
    byte          Reserved:4;           // AF_V1_SUPPORT uses for AppFlags:4.
    byte          AppNumInClusters;
    cld_t         *pAppInClusterList;
    byte          AppNumOutClusters;
    cld_t         *pAppOutClusterList;
} SimpleDescriptionFormat_t;
```

2.6.4 端点

端点 EndPoint 是协议栈应用层的入口，也可以理解应用对象（Application Object）存在的地方，它是为实现一个设备描述而定义的一组群集。每个 ZigBee 设备可以最多支持 240 这样的端点，这也意味着在每个设备上可以定义 240 个应用对象。端点 0 被保留用于与 ZDO 接口，这是每个 ZigBee 设备必须使用的端点，而端点 255 被保留用于广播，端点 241-254 则被保留用于将来做扩展使用。

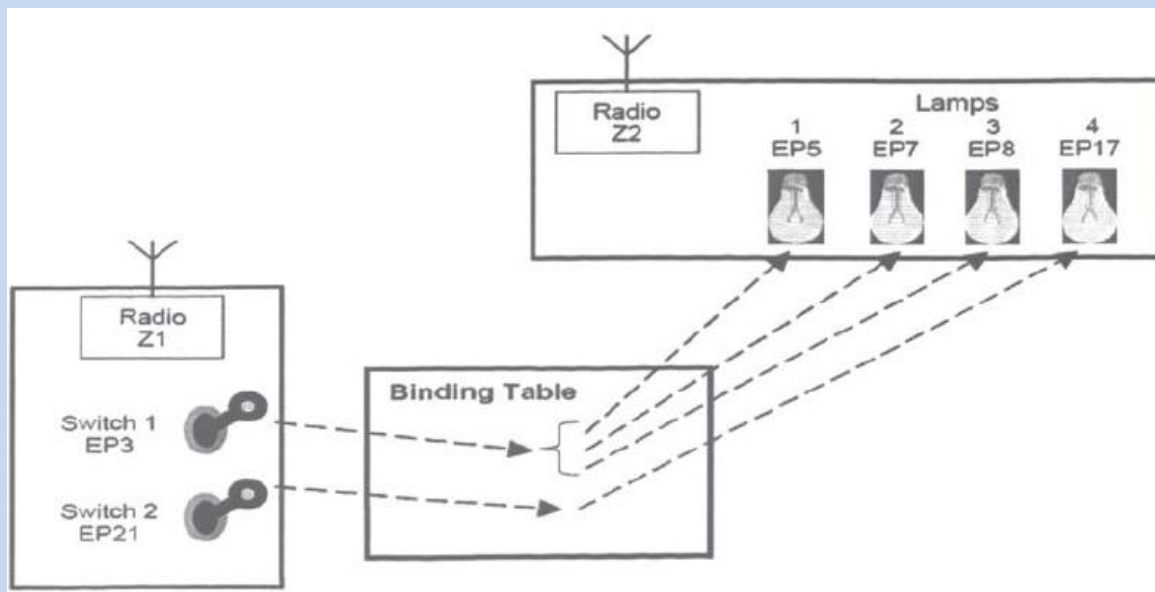
2.6.5 节点

节点 Node 也可以理解为一个容器，包含一组 ZigBee 设备，分享一个无线信道。每个节点

有且只有一个无线信道使用。

2.7 绑定 (binding)

在 zigbee 协议中定义了一种特殊的操作,叫做绑定(binding)操作。它能够通过使用 ClusterID 为不同节点上的独立端点建立一个逻辑上的连接。以下图为例来说明绑定操作:



图中 ZigBee 网络中的两个节点分别为 Z1 和 Z2, 其中 Z1 节点中包含两个独立端点分别是 EP3 和 EP21, 它们分别表示开关 1 和开关 2。Z2 节点中有 EP5、EP7、EP8、EP17 四个端点分别表示从 1 到 4 这四盏灯。在网络中, 通过建立 ZigBee 绑定操作, 可以将 EP3 和 EP5、EP7、EP8 进行绑定, 将 EP21 和 EP17 进行绑定。这样开关 1 便可以同时控制电灯 1、2、3, 开关 2 便可以控制电灯 4。利用绑定操作, 还可以更改开关和电灯之间的绑定关系, 从而形成不同的控制关系。从这个例子, 可以看出绑定操作能够使用户的应用变得更加方便灵活。

要实现绑定操作, 端点必须向协调器发送绑定请求, 协调器在有限的时间间隔内接收到两个端点的绑定请求后, 便通过建立端点之间的绑定表在这两个不同的端点之间形成了一个逻辑链路。因此, 在绑定后的两个端点之间进行消息传送的过程属于消息的间接传送。其中一个端点首先会将信息发送到 ZigBee 协调器中, ZigBee 协调器在接收到消息后会通过查找绑定表,

将消息发送到与这个端点绑定的所有端点中，从而实现了绑定端点之间的通信。

2.8 路由(Routing)

2.8.1 概述(Overview)

路由对与应用层来说是完全透明的。应用程序只需简单的向下发送去往任何设备的数据到栈中，栈会负责寻找路径。这种方法，应用程序不知道操作是在一个多跳的网络当中的。

路由还能够自愈 ZigBee 网络，如果某个无线连接断开了，路由功能又能自动寻找一条新的路径避开那个断开的网络连接。这就极大的提高了网络的可靠性，同时也是 ZigBee 网络的一个关键特性。

2.8.2 路由协议(Routing Protocol)

ZigBee 执行基于用于 AODV 专用网络的路由协议。简化后用于传感器网络。ZigBee 路由协议有助于网络环境有能力支持移动节点，连接失败和数据包丢失。

当路由器从他自身的应用程序或者别的设备那里收到一个单点发送的数据包，则网络层(NWK Layer)根据一下程序将它继续传递下去。如果目标节点是它相邻路由器中的一个，则数据包直接被传送给目标设备。否则，路由器将要检索它的路由表中与所要传送的数据包的目标地址相符合的记录。如果存在与目标地址相符合的活动路由记录，则数据包将被发送到存储在记录中的下一级地址中去。如果没有发现任何相关的路由记录，则路由器发起路径寻找，数据包存储在缓冲区中知道路径寻找结束。

ZigBee 终端节点不执行任何路由功能。终端节点要向任何一个设备传送数据包，它只需简单的将数据向上发送给它的父亲设备，由它的父亲设备以它自己的名义执行路由。同样的，任何一个设备要给终端节点发送数据，发起路由寻找，终端节点的父节点都以它的名义来回应。

注意 ZigBee 地址分配方案使得对于任何一个目标设备，根据它的地址都可以得到一条路径。

在 Z-Stack 中，如果万一正常的路径寻找过程不能启动的话(通常由于缺少路由表空间)，那么 Z-Stack 拥有自动回退机制。

此外，在 Z-Stack 中，执行的路由已经优化了路由表记录。通常，每一个目标设备都需要一条路由表记录。但是，通过把一定父亲节点记录与其子所有子结点的记录合并，这样既可以优化路径也可以不丧失任何功能。

ZigBee 路由器，包括协调器执行下面的路由函数：(i)路径发现和选择；(ii)路径保持维护；(iii)路径期满。

2.8.3 路径的发现和选择(Route Discovery and Selection)

路径发现是网络设备凭借网络相互协作发现和建立路径的一个过程。路由发现可以由任意一个路由设备发起，并且对于某个特定的目标设备一直执行。路径发现机制寻找源地址和目标地址之间的所有路径，并且试图选择可能的最好的路径。

路径选择就是选择出可能的最小成本的路径。每一个结点通常持有跟它所有邻接点的“连接成本(link costs)”。通常，连接成本的典型函数是接收到的信号的强度。沿着路径，求出所有连接的成本总和，便可以得到整个路径的“路径成本”。路由算法试图寻找到拥有最小路径成本的路径。

路径通过一系列的请求和回复数据包被发现。源设备通过向它的所有邻接节点广播一个路由请求数据包，来请求一个目标地址的路径。当一个节点接收到 RREQ 数据包，它依次转发 RREQ 数据包。但是在转发之前，它要加上最新的连接成本，然后更新 RREQ 数据包中的成本值。这样，沿着所有它通过的连接，RREQ 数据包携带着连接成本的总和。这个过程一直持续到 RREQ 数据包到达目标设备。通过不同的路由器，许多 RREQ 副本都将到达目标设备。目

标设备选择最好的 RREQ 数据包,然后发回一个路径答复数据包(a Route Reply)RREP 给源设备。RREP 数据包是一个单点发送数据包,它沿着中间节点的相反路径传送直到它到达原来发送请求的节点为止。

一旦一条路径被创建,数据包就可以发送了。当一个结点与它的下一级相邻节点失去了连接(当它发送数据时,没有收到 MAC ACK),该节点向所有等待接收它的 RREQ 数据包的节点发送一个 RERR 数据包,将它的路径设为无效。各个结点根据收到的数据包 RREQ、RREP 或者 RERR 来更新它的路由表。

2.8.4 路径保持维护(Route maintenance)

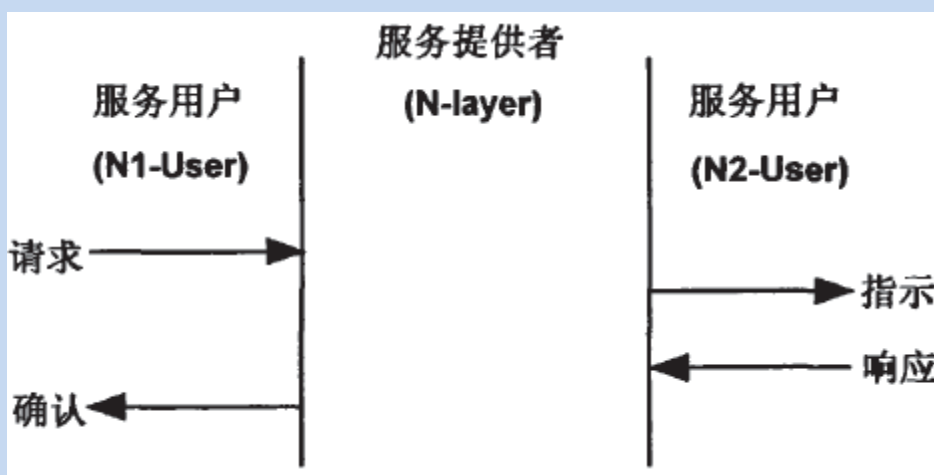
网状网提供路径维护和网络自愈功能。中间节点沿着连接跟踪传送失败,如果一个连接被认定是坏链,那么上游节点将针对所有使用这条连接的路径启动路径修复。节点发起重新发现直到下一次数据包到达该节点,标志路径修复完成。如果不能够启动路径发现或者由于某种原因失败了,节点则向数据包的源节点发送一个路径错误包(RERR),它将负责启动新路径的发现。这两种方法,路径都自动重建。

2.8.5 路径期满(Route expiry)

路由表为已经建立连接路径的节点维护路径记录。如果在一定的时间周期内,没有数据通过沿着这条路径发送,这条路径将被表示为期满。期满的路径一直保留到它所占用的空间要被使用为止。这样,路径在绝对不使用之前不会被删除掉的。在配置文件 f8wConfig.cfg 文件中配置自动路径期满时间。设置 ROUTE_EXPIRY_TIME 为期满时间,单位为秒。如果设置为 0,则表示关闭自动期满功能。

2.9 ZigBee 原语

ZigBee 协议按照开放系统互联的 7 层模型将协议分成了一系列的层结构，各层之间通过相应的服务访问点来提供服务。这样使得处于协议中的不同层能够根据各自的功能进行独立的运作，从而使整个协议栈的结构变得清晰明朗。另一方面，由于 ZigBee 协议栈是一个有机的整体，任何 ZigBee 设备要能够正确无误的工作，就要求协议栈各层之间共同协作。因此，层与层之间的信息交互就显得十分重要。ZigBee 协议为了实现层与层之间的关联，采用了称为服务“原语”的操作。利用下图来说明原语操作的概念。



服务由 N 用户和 N 层之间信息流的描述来指定。这个信息流由离散瞬时事件构成，以提供服务的特征。每个事件由服务原语组成，它将在一个用户的某一层，通过与该层相关联的层服务访问 A(SAP)与建立对等连接的用户的相同层之间传送。层与层之间的原语一般情况下可以分为 4 种类型：

- ◆ 请求：请求原语从 NI 用户发送到它的 N 层，请求发起一个服务。
- ◆ 指示：指示原语从 N 层到 N2 用户，指示一个对 N2 用户有重要意义外部 N 层事件。这个事件可能与一个远程的服务请求有关，或者由内部事件产生。
- ◆ 响应：响应原语由 N2 用户向它的 N 层传递，用来响应上一个由指示原语引起的过程。

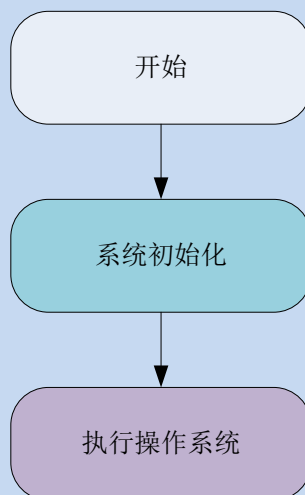
◆ 确认：确认原语由 N 层向 NI 用户传递，用来传递与前面一个或多个服务请求相关的执行结果。

第三章 Z-Stack 协议栈总体设计

ZigBee 协议栈依据 IEEE 802.15.4 标准和 ZigBee 协议规范。ZigBee 网络中的各种操作需要利用协议栈各层所提供的原语操作来共同完成。原语操作的实现过程往往需要向下一层发起一个原语操作并且通过下层返回的操作结果来判断出下一条要执行的原语操作。IEEE 802.15.4 标准和 ZigBee 协议规范中定义各层原语操作多达数十条，原语的操作过程也比较复杂，它已经不是一个简单的单任务软件。对于这样一个复杂的嵌入式通信软件来说，其实现通常需要依靠嵌入式操作系统来完成。

挪威半导体公司 Chipcon(目前已经被 TI 公司收购)作为业界领先的 ZigBee 一站式方案供应商，在推出其 CC2530 开发平台时，也向用户提供了自己的 ZigBee 协议栈软件-Z-Stack。这是一款业界领先的商业级协议栈，使用 CC2530 射频芯片，可以使用户很容易的开发出具体的应用程序来。Z-Stack 使用瑞典公司 IAR 开发的 IAR Embedded Workbench for MCS-51 作为它的集成开发环境。Chipcon 公司为自己设计的 Z-Stack 协议栈中提供了一个名为操作系统抽象层 OSAL 的协议栈调度程序。对于用户来说，除了能够看到这个调度程序外，其它任何协议栈操作的具体实现细节都被封装在库代码中。用户在进行具体的应用开发时只能够通过调用 API 接口来进行，而无权知道 ZigBee 协议栈实现的具体细节。

Z-Stack 由 main () 函数开始执行，main () 函数共做了 2 件事：一是系统初始化，另外一件是开始执行轮转查询式操作系统，如下图所示：



3.1 任务初始化

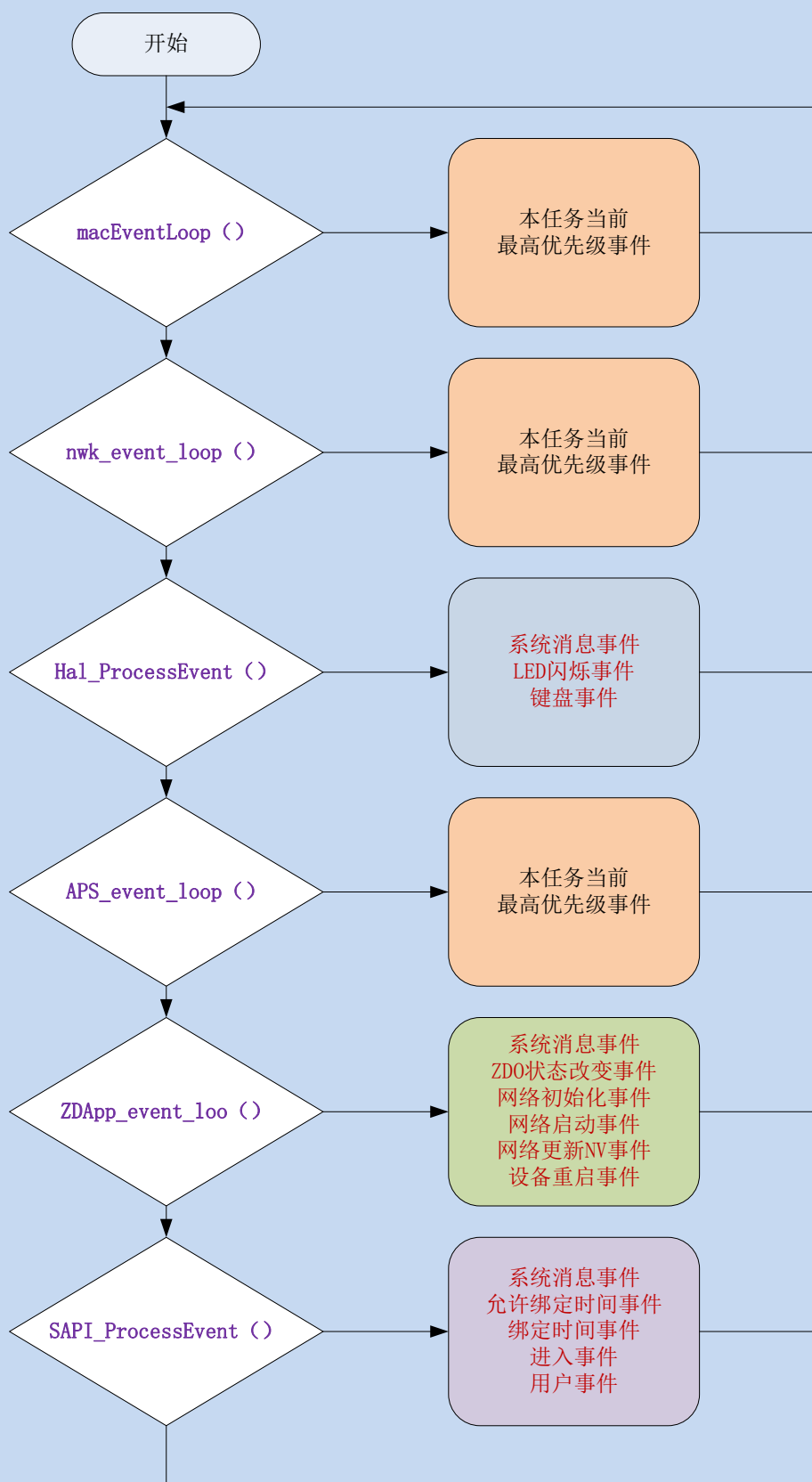
```
void osallInitTasks( void )
{
    uint8 taskID = 0;

    tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);
    osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));

    macTaskInit( taskID++ );
    nwk_init( taskID++ );
    Hal_Init( taskID++ );
    #if defined( MT_TASK )
        MT_TaskInit( taskID++ );
    #endif
    APS_Init( taskID++ );
    #if defined ( ZIGBEE_FRAGMENTATION )
        APSF_Init( taskID++ );
    #endif
    ZDApp_Init( taskID++ );
    #if defined ( ZIGBEE_FREQ_AGILITY ) || defined ( ZIGBEE_PANID_CONFLICT )
        ZDNwkMgr_Init( taskID++ );
    #endif
    SampleApp_Init( taskID );
}
```

3.2 任务调度

ZigBee 协议栈中的每一层都有很多原语操作要执行，因此对于整个协议栈来说，就会有很多并发操作要执行。协议栈中的每一层都设计了一个事件处理函数，用来处理与这一层操作相关的各种事件。将这些事件处理函数看成是与协议栈每一层相对应的任务，由 ZigBee 协议栈中调度程序 OSAL 来进行管理。这样，对于协议栈来说，无论何时发生了何种事件，我们都可以通过调度协议栈相应层的任务，即事件处理函数来进行处理。这样，整个协议栈便会按照时间顺序有条不紊的运行。



ZigBee 协议栈的实时性要求并不高，因此在设计任务调度程序时，OSAL 只采用了轮询任务调度队列的方法来进行任务调度管理。

- 任务列表:

```
const pTaskEventHandlerFn tasksArr[] = {
    macEventLoop,
    nwk_event_loop,
    Hal_ProcessEvent,
#ifdef MT_TASK
    MT_ProcessEvent,
#endif
    APS_event_loop,
#ifdef ZIGBEE_FRAGMENTATION
    APSF_ProcessEvent,
#endif
    ZDApp_event_loop,
#ifdef ZIGBEE_FREQ_AGILITY || defined ( ZIGBEE_PANID_CONFLICT )
    ZDNwkMgr_event_loop,
#endif
    SampleApp_ProcessEvent
};
```

- 任务调度主循环

```
void osal_start_system( void )
{
    for(;;) // Forever Loop
    {
        uint8 idx = 0;

        do {
            if (tasksEvents[idx]) // Task is highest priority that is ready.
            {
                break;
            }
        } while (++idx < tasksCnt);

        if (idx < tasksCnt)
        {
            uint16 events;
            halIntState_t intState;

            HAL_ENTER_CRITICAL_SECTION(intState);
            events = tasksEvents[idx];
            tasksEvents[idx] = 0; // Clear the Events for this task.
            HAL_EXIT_CRITICAL_SECTION(intState);
```

```
events = (tasksArr[idx])( idx, events );

HAL_ENTER_CRITICAL_SECTION(intState);
tasksEvents[idx] |= events; // Add back unprocessed events to the current task.
HAL_EXIT_CRITICAL_SECTION(intState);
}
}
}
```

● 设置事件发生标志

当协议栈中有任何事件发生时，我们可以通过设置 event_flag 来标记有事件发生，以便主循环函数能够及时加以处理。其函数说明如下：

```
uint8 osal_set_event( uint8 task_id, uint16 event_flag )
{
    if ( task_id < tasksCnt )
    {
        halIntState_t intState;
        HAL_ENTER_CRITICAL_SECTION(intState); // Hold off interrupts
        tasksEvents[task_id] |= event_flag; // Stuff the event bit(s)
        HAL_EXIT_CRITICAL_SECTION(intState); // Release interrupts
        return ( SUCCESS );
    }
    else
    {
        return ( INVALID_TASK );
    }
}
```

3.3 时间管理

在协议栈中的每一层都会有很多不同的事件发生，这些事件发生的时间顺序各不相同。很多时候，事件并不要求立即得到处理，而是要求过一定的时间后再进行处理。因此，往往会遇到下面情况：假设 A 事件发生后要求 10 秒之后执行，B 事件在 A 事件发生 1 秒后产生，且 B 事件要求 5 秒后执行。为了按照合理的时间顺序来处理不同事件的执行，这就需要对各种不同的事件进行时间管理。OSAL 调度程序设计了与时间管理相关的函数，用来管理各种不同的要被处理的事件。

对事件进行时间管理,OSAL 也采用了链表的方式进行,每当发生一个要被处理的事件后,就启动一个逻辑上的定时器,并将此定时器添加到链表之中。利用硬件定时器作为时间操作的基本单元。设置时间操作的最小精度为 lms, 每 lms 硬件定时器便产生一个时间中断, 在时间中断处理程序中去更新定时器链表。每次更新, 就将链表中的每一项时间计数减 1, 如果发现定时器链表中有某一表项时间计数已经减到 0, 则将这个定时器从链表中删除, 并设置相应的事件标志。这样任务调度程序便可以根据事件标志进行相应的事件处理。根据这种思路, 来自协议栈中的任何事件都可以按照时间顺序得到处理。从而提高了协议栈设计的灵活性。

在设计过程中需要经常使用这样一个时间管理函数, 其函数说明如下:

```
uint8 osal_start_timerEx( uint8 taskID, uint16 event_id, uint16 timeout_value )
{
    halIntState_t intState;
    osalTimerRec_t *newTimer;

    HAL_ENTER_CRITICAL_SECTION( intState ); // Hold off interrupts.

    // Add timer
    newTimer = osalAddTimer( taskID, event_id, timeout_value );

    HAL_EXIT_CRITICAL_SECTION( intState ); // Re-enable interrupts.

    return ( (newTimer != NULL) ? SUCCESS : NO_TIMER_AVAIL );
}
```

这个函数为事件 event id 设置超时等待时间 timeout value。一旦等待结束, 便为 task id 所对应的任务设置相应的事件发生标记, 从而达到对事件进行延迟处理的目的。

3.4 原语通信

从前面可以知道, 为了使 ZigBee 网络能够正常工作, IEEE 802. 15. 4 标准和 ZigBee 协议规范在协议的各层分别定义了大量的原语操作。其中, 请求(Request)、响应(Response)原语分别由协议栈中处于较高位置的层向较低层发起; 确认(Confirm)、指示(Indication)原语则从较低层

向较高层返回结果或信息。

在协议栈设计时，对请求(Request)、响应(Response)原语可以直接使用函数调用来实现。对于确认(Confirm)、指示(Indication)原语则需要采用间接处理的机制来完成。这是因为在协议栈设计的过程中，一个原语的操作往往需要逐层调用下层函数并根据下层返回的结果来进行进一步操作。在这种情况下，一个原语的操作从发起到完成需要很长的时间。因此，如果让程序一直等待下层返回的结果再来做进一步处理，这样就会使微处理器大部分时间处于循环等待之中，从而无法及时处理其他请求。

因此，在设计与请求、响应原语操作相对应的函数时，一旦调用了下层相关函数后，就立即返回。下层处理函数在操作结束后，将结果以消息的形式发送到上层并产生一个系统事件，调度程序发现这个事件后就会调用相应的事件处理函数对它进行处理。OSAL 调度程序设计了两个相关的函数来完成这个过程，下面分别介绍。

◆ 向目标任务发送消息的函数

这个函数主要用来将原语操作的结果以消息的形式向上层任务发送，并产生一个系统事件来通知调度程序。其函数说明如下：

```
uint8 osal_msg_send( uint8 destination_task, uint8 *msg_ptr )
{
    if ( msg_ptr == NULL )
        return ( INVALID_MSG_POINTER );

    if ( destination_task >= tasksCnt )
    {
        osal_msg_deallocate( msg_ptr );
        return ( INVALID_TASK );
    }

    // Check the message header
    if ( OSAL_MSG_NEXT( msg_ptr ) != NULL ||
         OSAL_MSG_ID( msg_ptr ) != TASK_NO_TASK )
    {
        osal_msg_deallocate( msg_ptr );
    }
}
```



```
    return ( INVALID_MSG_POINTER );
}

OSAL_MSG_ID( msg_ptr ) = destination_task;

// queue message
osal_msg_enqueue( &osal_qHead, msg_ptr );

// Signal the task that a message is waiting
osal_set_event( destination_task, SYS_EVENT_MSG );

return ( SUCCESS );
}
```

其中参数 destination-task 是目标任务的任务号，参数指针 msg_ptr 指向要被发送的消息，参数 len 为消息的长度。

◆ 消息提取函数

这个函数用来从内存空间中提取相应的消息。其消息结构和函数说明如下：

```
uint8 *osal_msg_receive( uint8 task_id )
{
    osal_msg_hdr_t *listHdr;
    osal_msg_hdr_t *prevHdr = NULL;
    osal_msg_hdr_t *foundHdr = NULL;
    halIntState_t    intState;

    // Hold off interrupts
    HAL_ENTER_CRITICAL_SECTION(intState);

    // Point to the top of the queue
    listHdr = osal_qHead;

    // Look through the queue for a message that belongs to the asking task
    while ( listHdr != NULL )
    {
        if ( (listHdr - 1)->dest_id == task_id )
        {
            if ( foundHdr == NULL )
            {
                // Save the first one
                foundHdr = listHdr;
            }
        }
    }
}
```

```
    else
    {
        // Second msg found, stop looking
        break;
    }
}
if ( foundHdr == NULL )
{
    prevHdr = listHdr;
}
listHdr = OSAL_MSG_NEXT( listHdr );
}

// Is there more than one?
if ( listHdr != NULL )
{
    // Yes, Signal the task that a message is waiting
    osal_set_event( task_id, SYS_EVENT_MSG );
}
else
{
    // No more
    osal_clear_event( task_id, SYS_EVENT_MSG );
}

// Did we find a message?
if ( foundHdr != NULL )
{
    // Take out of the link list
    osal_msg_extract( &osal_qHead, foundHdr, prevHdr );
}

// Release interrupts
HAL_EXIT_CRITICAL_SECTION(intState);

return ( (uint8*) foundHdr );
}
```

这个函数返回一个指向所需提取信息的指针。

第四章 开发工具的安装及使用

4.1 IAR 安装

应用及开发 ZigBee 2007 系统主要使用的软件工具是 IAR 7.51A，它好比于开发单片机系统所用的 keil 软件。这里请注意：ZigBee 2006 所用的软件工具为 IAR 7.30B，这 2 个不同版本的 IAR 可以同时安装在一台 PC 上，但打开工程文件时，需要首先从开始菜单->IAR 7.51 启动 IAR，然后再选择工程文件打开，否则会导致 IAR 版本与工程文件的不匹配。

IAR 7.51A 的安装源文件目录为：ZigBee2007 系统\开发工具\IAR7.51A\CD-EW8051-751A

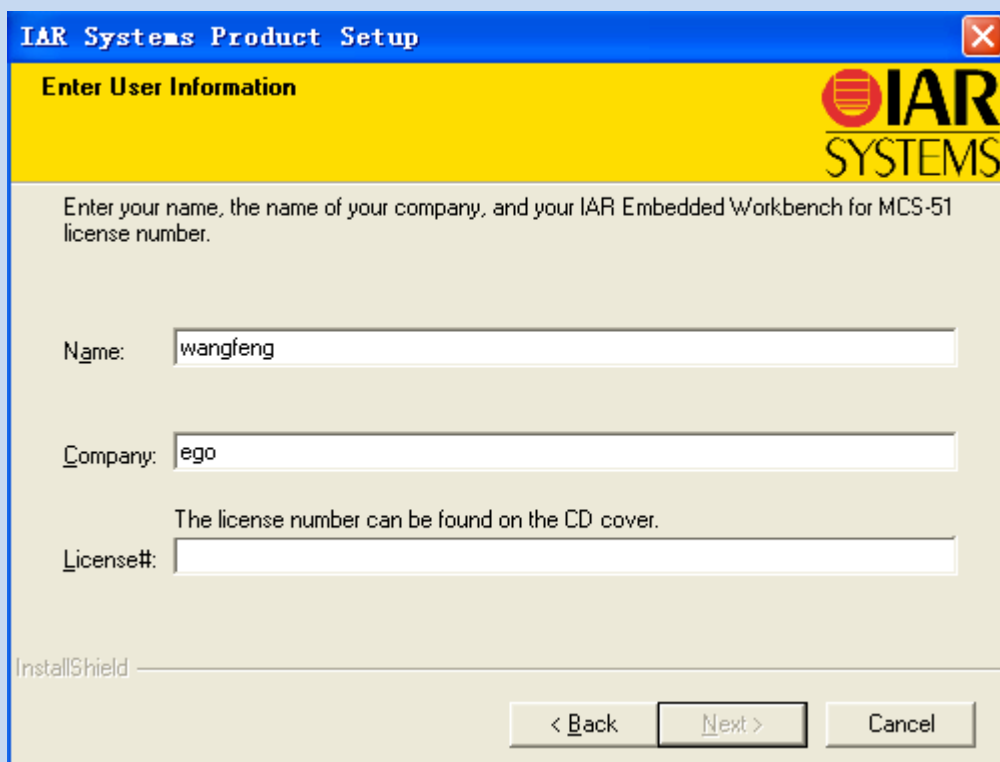
双击“autorun”：



点击“Install IAR Embedded Workbench”：



点击“Next”直到：

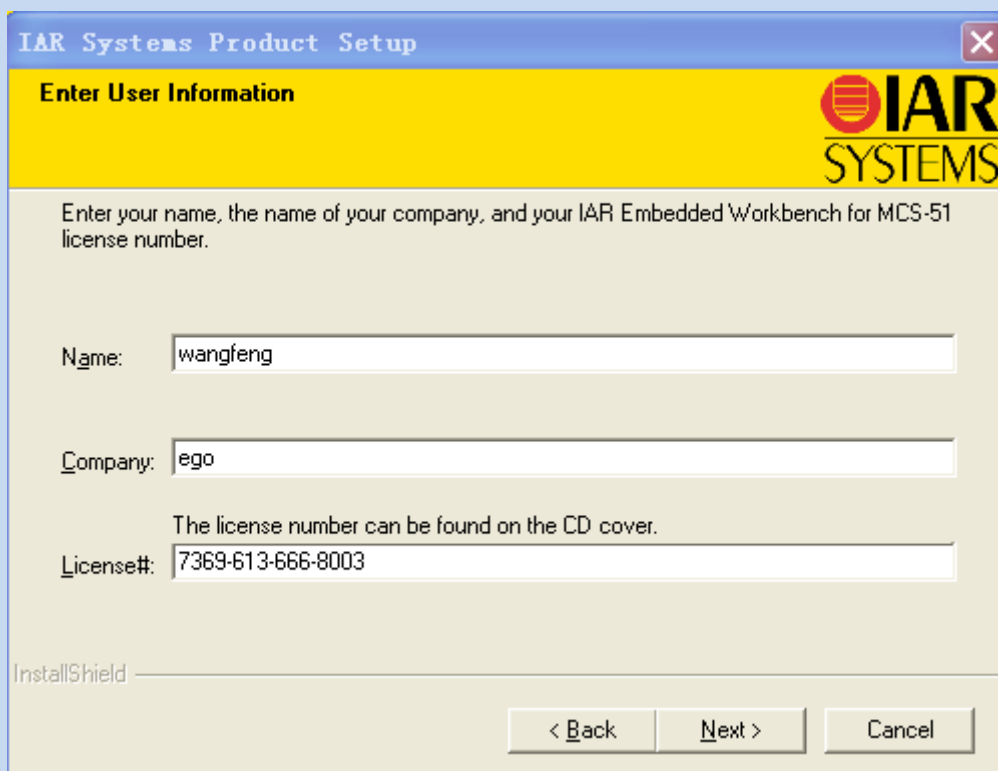


这里需要输入 License，License 由注册机生成，打开 IAR 7.51A 注册机文件，文件目录为：

ZigBee2007 系统\开发工具\IAR7.51A\key:



点击“Generate”将生成的 License number 输入到 IAR License 输入框中：



The screenshot shows the 'Enter User Information' window of the IAR Systems Product Setup. The window has a yellow header with the IAR Systems logo. Below the header, there is a text box for 'Name' containing 'wangfeng', a text box for 'Company' containing 'ego', and a text box for 'License#' containing '7369-613-666-8003'. A note states: 'The license number can be found on the CD cover.' At the bottom, there are three buttons: '< Back', 'Next >', and 'Cancel'.

IAR Systems Product Setup

Enter User Information

Enter your name, the name of your company, and your IAR Embedded Workbench for MCS-51 license number.

Name: wangfeng

Company: ego

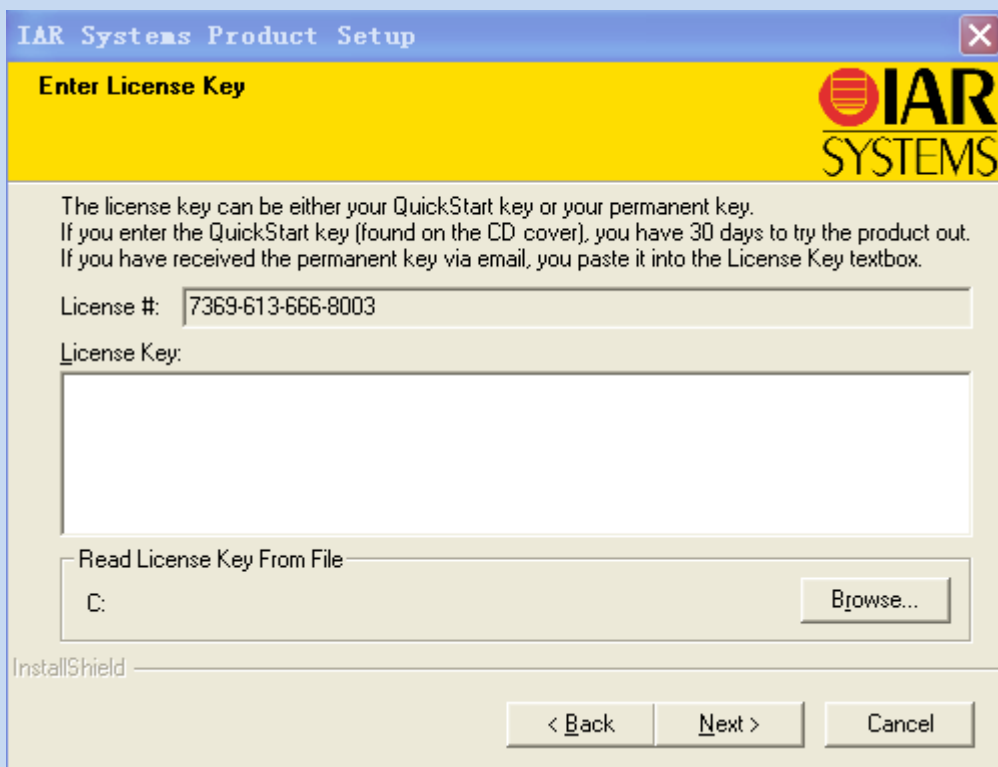
The license number can be found on the CD cover.

License#: 7369-613-666-8003

InstallShield

< Back Next > Cancel

然后点击 “Next”：



The screenshot shows the 'Enter License Key' window of the IAR Systems Product Setup. The window has a yellow header with the IAR Systems logo. Below the header, there is a text box for 'License #' containing '7369-613-666-8003' and a large text box for 'License Key'. A note states: 'The license key can be either your QuickStart key or your permanent key. If you enter the QuickStart key (found on the CD cover), you have 30 days to try the product out. If you have received the permanent key via email, you paste it into the License Key textbox.' At the bottom, there is a section for 'Read License Key From File' with a text box containing 'C:' and a 'Browse...' button. At the bottom right, there are three buttons: '< Back', 'Next >', and 'Cancel'.

IAR Systems Product Setup

Enter License Key

The license key can be either your QuickStart key or your permanent key. If you enter the QuickStart key (found on the CD cover), you have 30 days to try the product out. If you have received the permanent key via email, you paste it into the License Key textbox.

License #: 7369-613-666-8003

License Key:

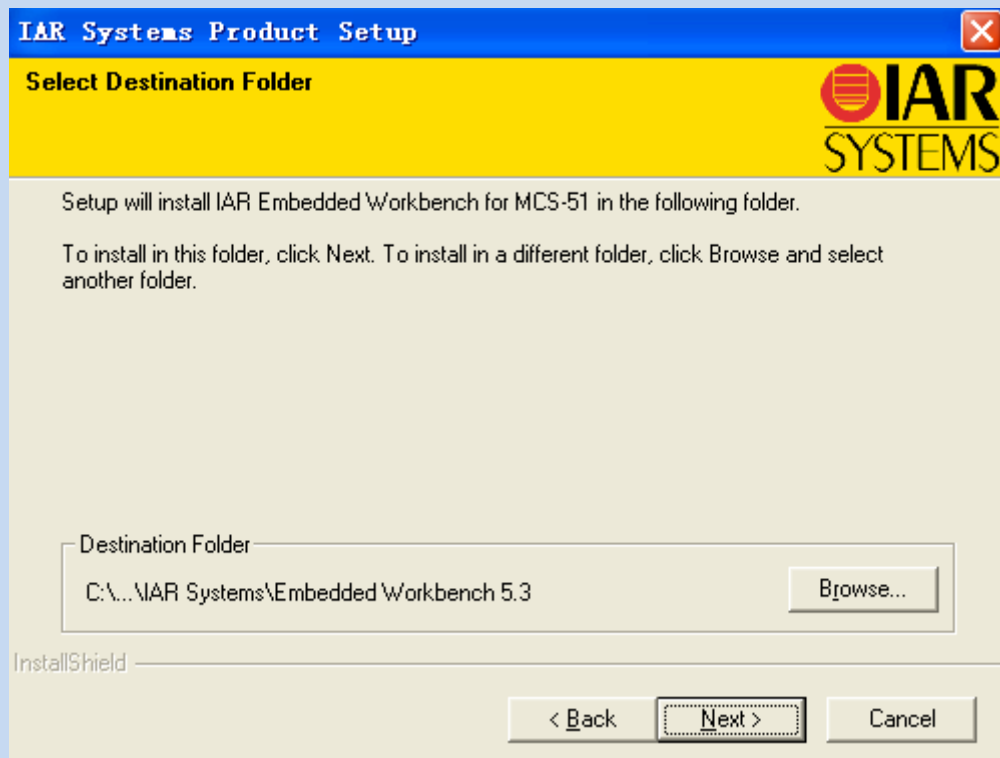
Read License Key From File

C: Browse...

InstallShield

< Back Next > Cancel

将注册机的 License key 复制并输入到上图的 License Key 输入框中，然后点击 “Next”：



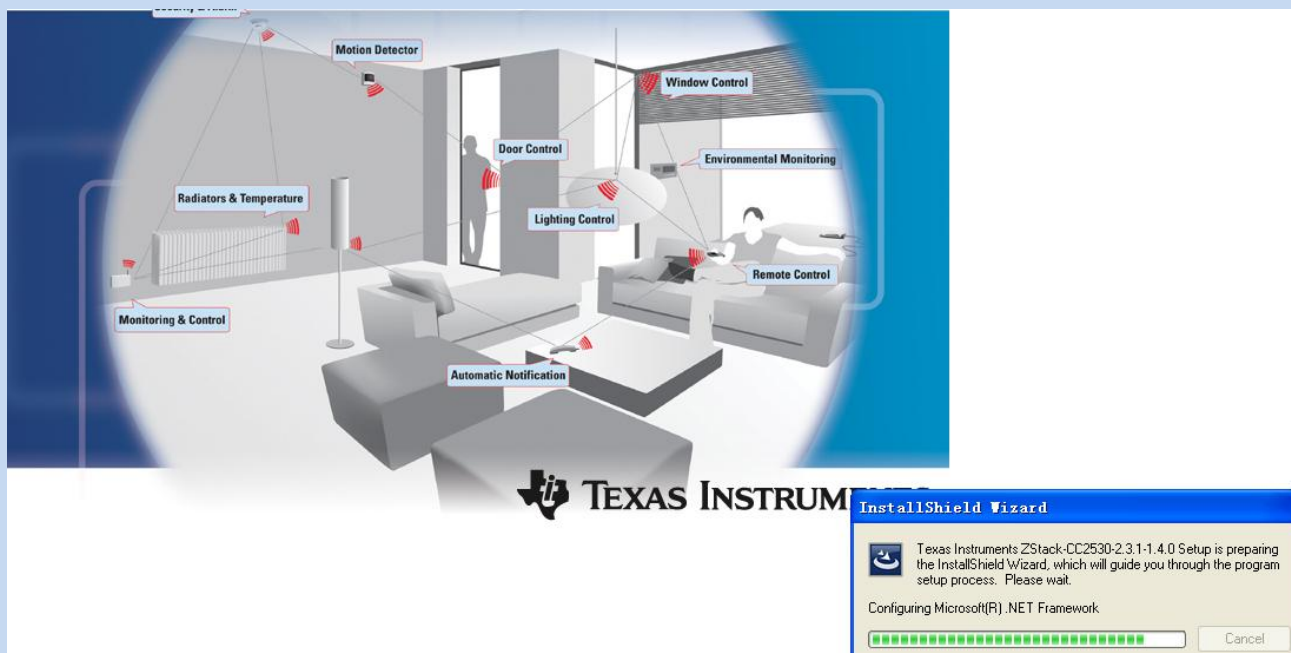
点击“Next”直到最后安装结束，点击“Finish”：



至此，我们成功安装了 IAR 7.51A。

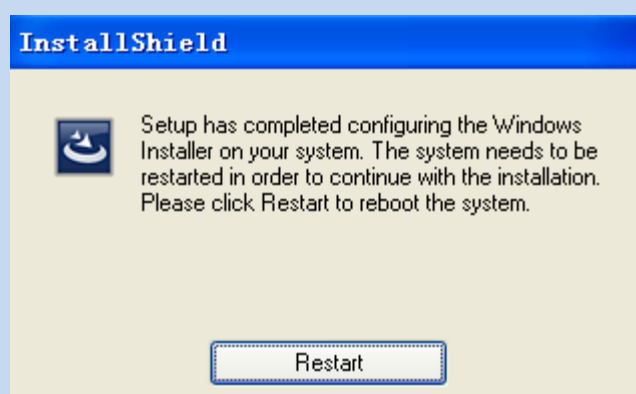
4.2 ZigBee 2007 协议栈安装

ZigBee2007 协议栈的安装目录为：ZigBee2007 系统\开发工具\ZigBee2007 协议栈，双击“ZStack-CC2530-2.3.1-1.4.0”：



当进行到.NET Framework 这个步骤时需要花费一些时间，请耐心等待。如果此步骤一直不能进行，那就需要我们首先安装较高版本的.NET Framework，然后再来安装 ZigBee2007 协议栈。

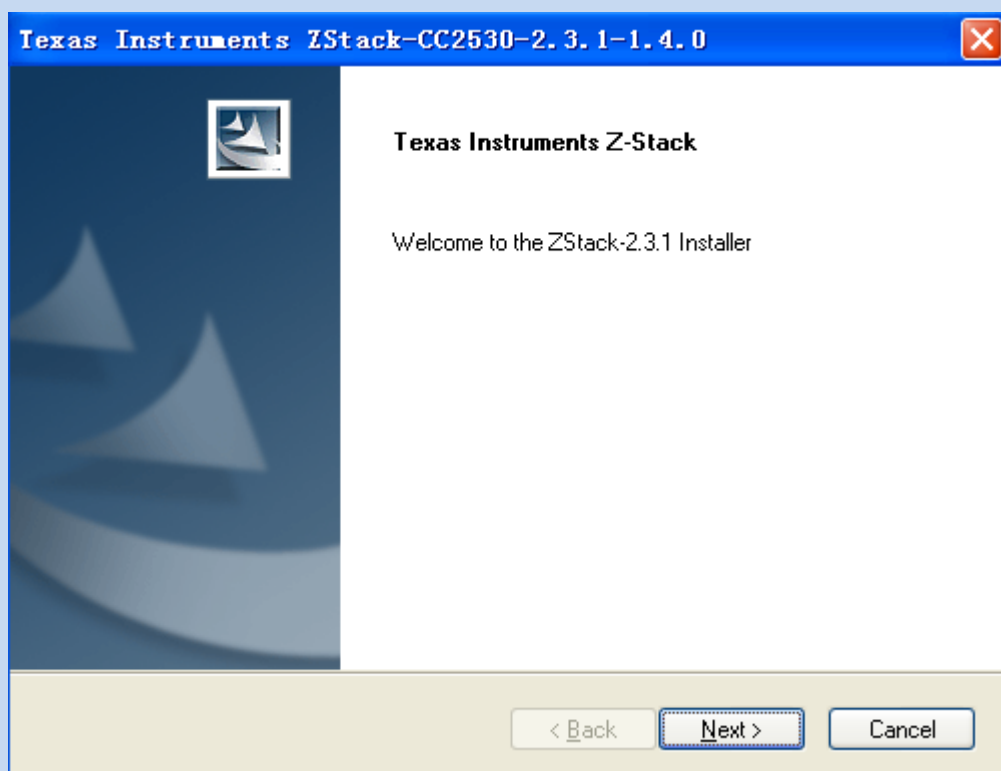
安装过程中，杀毒软件都应允许所有操作。



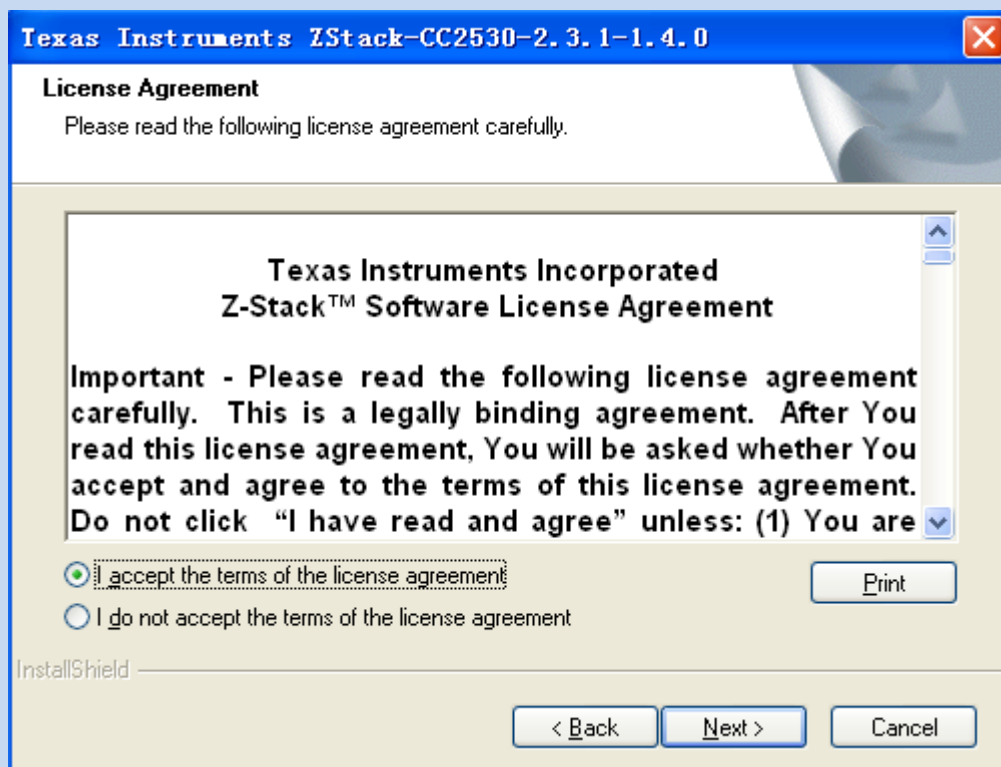
点击“Restart”，重新启动计算机，重启后继续安装。再次进入 ZigBee2007 系统\开发工具\ZigBee2007 协议栈，双击“ZStack-CC2530-2.3.1-1.4.0”：



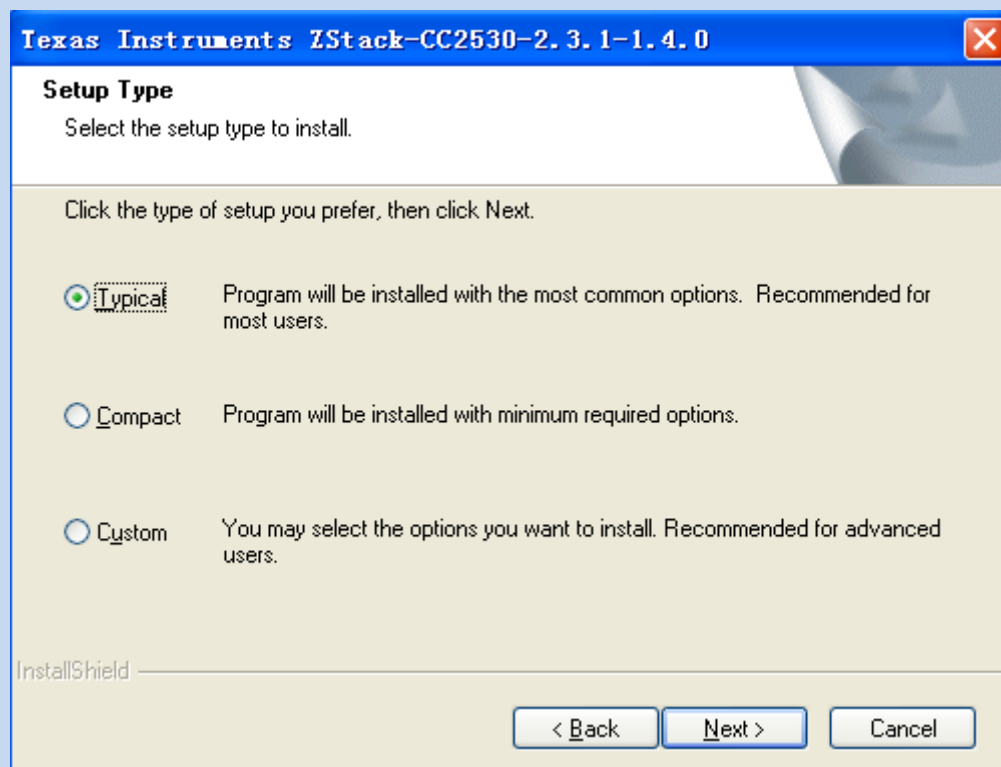
直至:



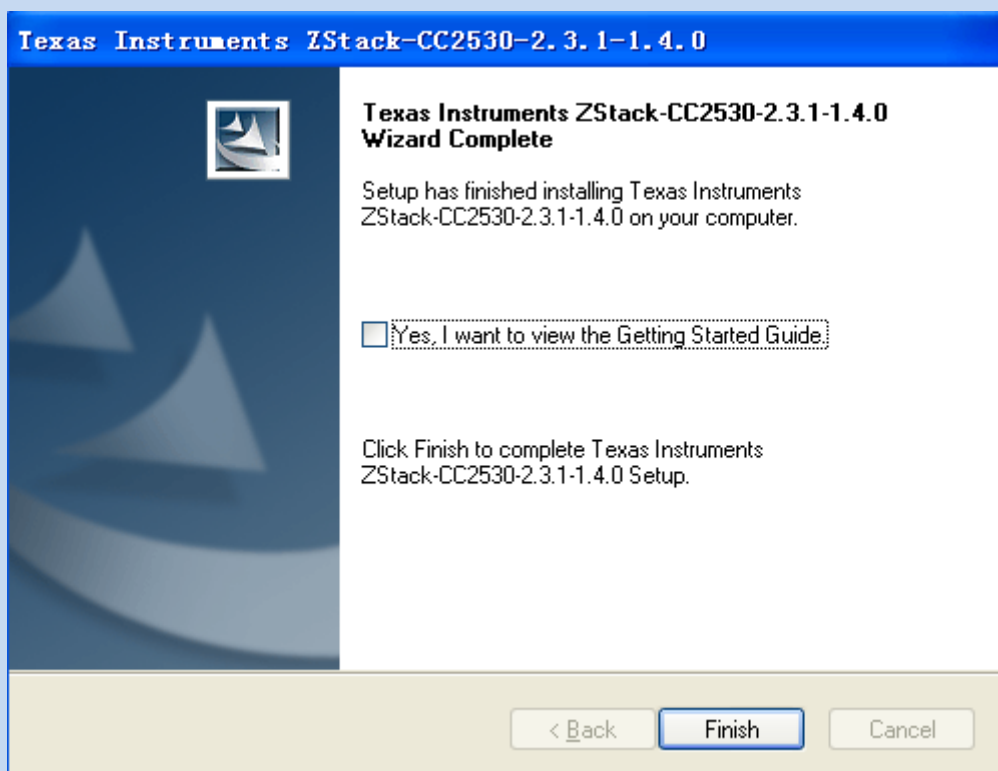
点击 “Next”:



选择 “I accept the terms of the license agreement”：



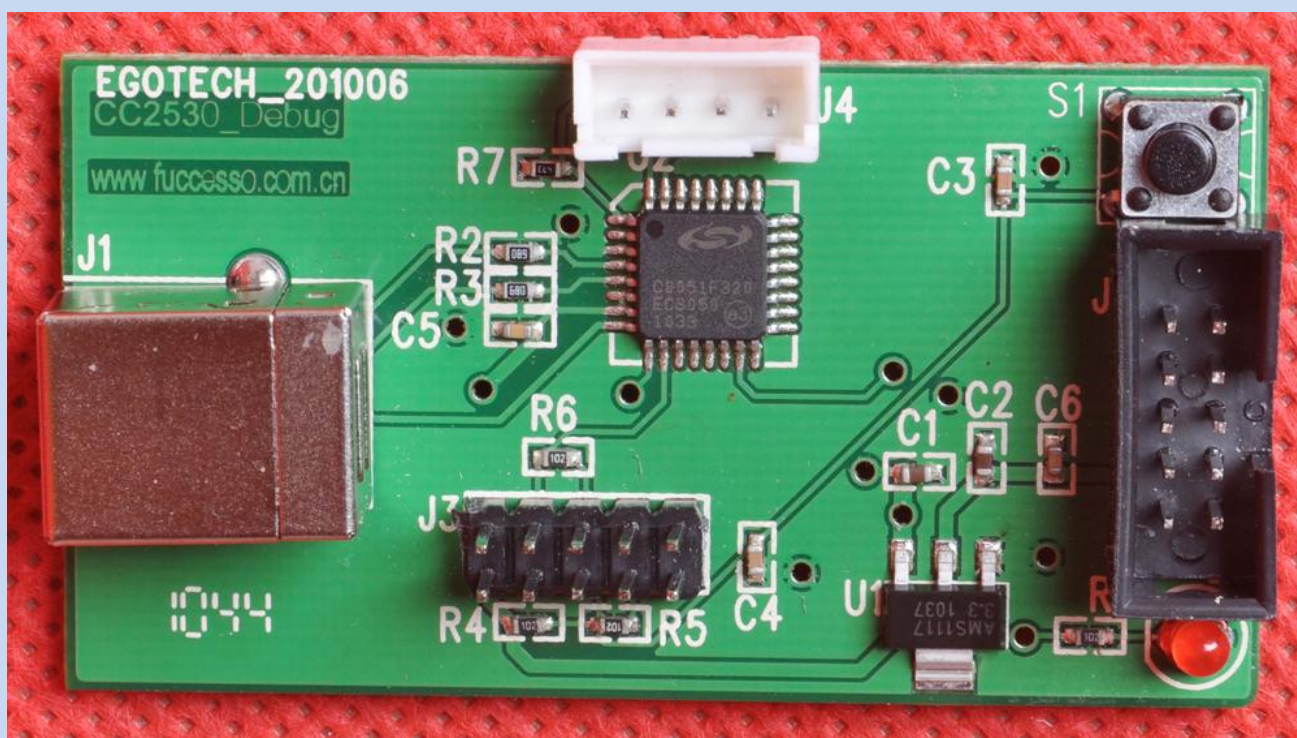
选择 “Typical”，然后点击 “Next”：



直至最后，点击“Finish”完成 ZigBee2007 协议栈安装。

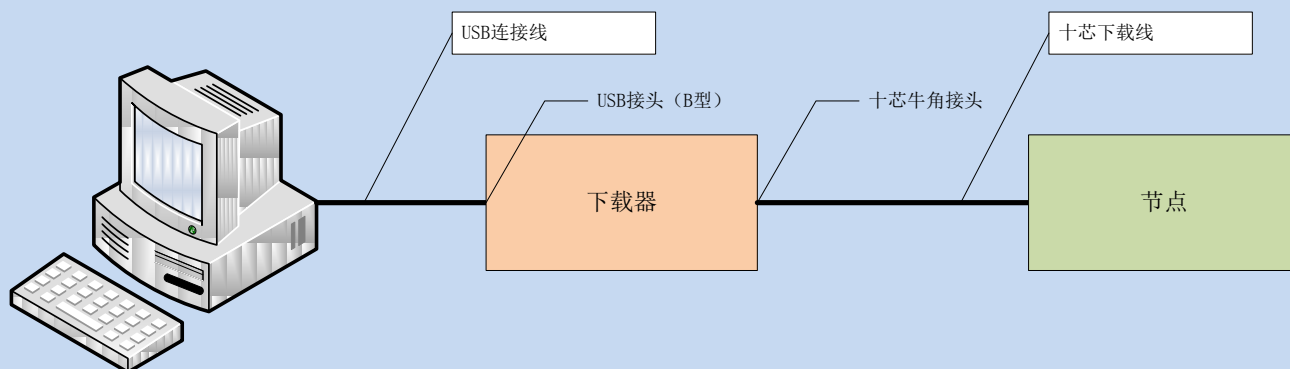
4.3 下载器硬件连接和驱动程序安装

下载器的实物图如下图所示：



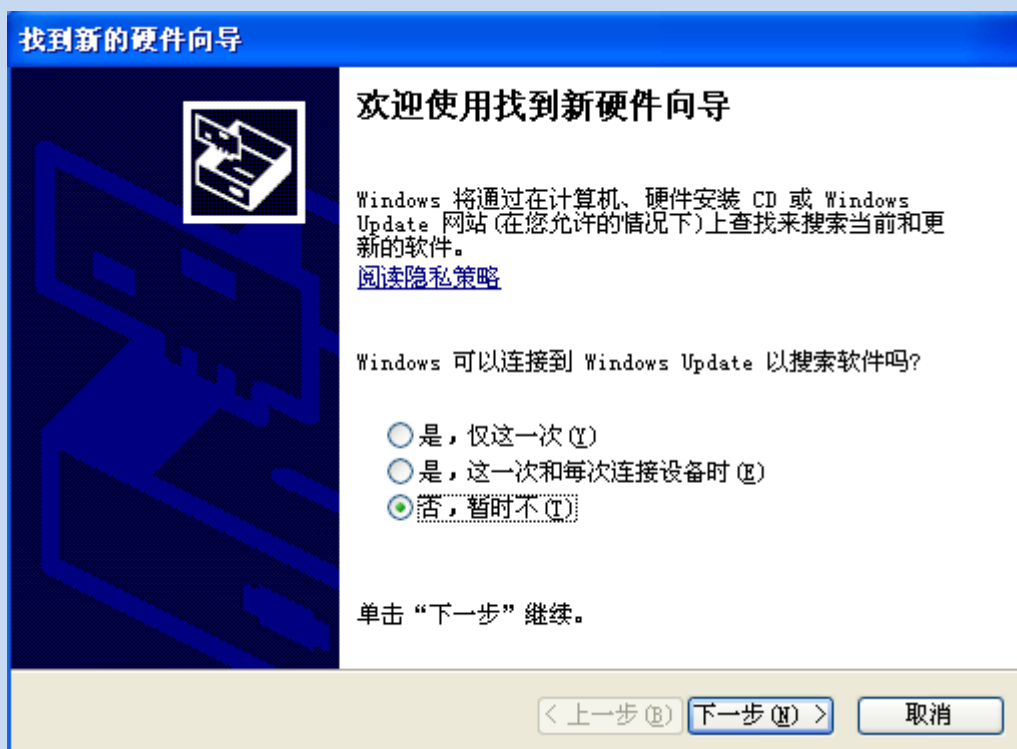
其中：J1 为 USB 接头（B 型），通过 USB 线可以连接下载器和电脑（PC）；J2 为十芯牛角下载接头（配备防呆缺口），通过十芯下载线可以连接下载器和各个节点；J3 是十芯插针接头，用于更新下载器主芯片的固件（客户不需要使用）；J4 是白色四芯接头（配备防呆缺口），通过四芯连接线可以连接下载器和终端节点，构成协议分析仪。

电脑、下载器和节点的连接方法如下图所示：

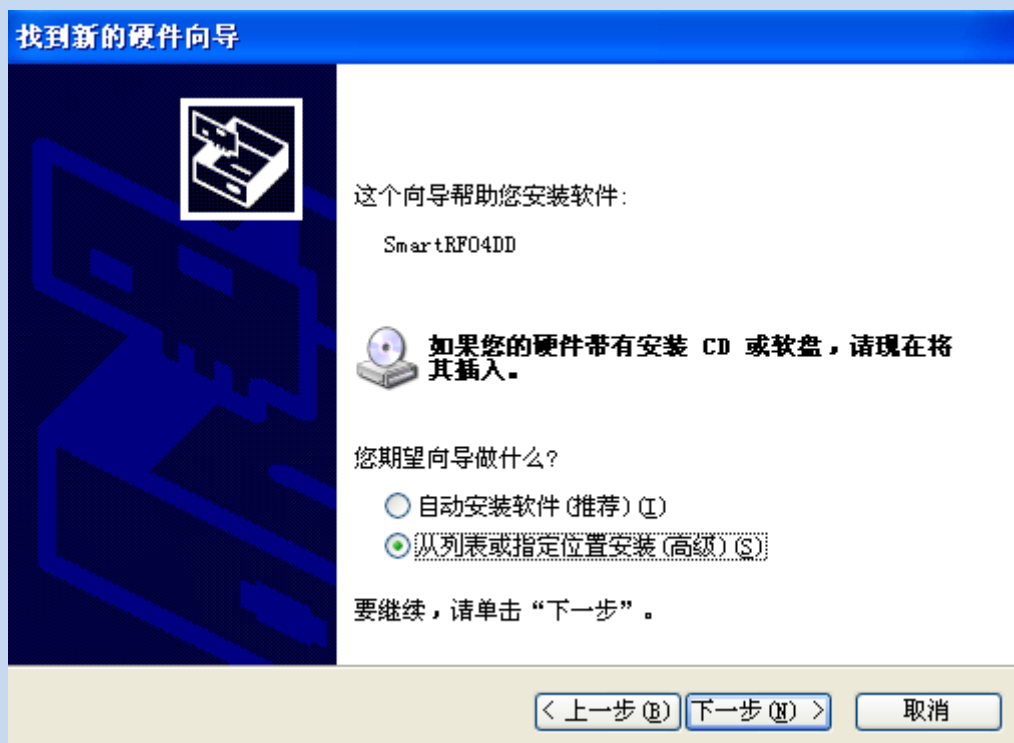


注意：十芯下载线与节点连接时，需要保持下载线接头突出部位朝外。

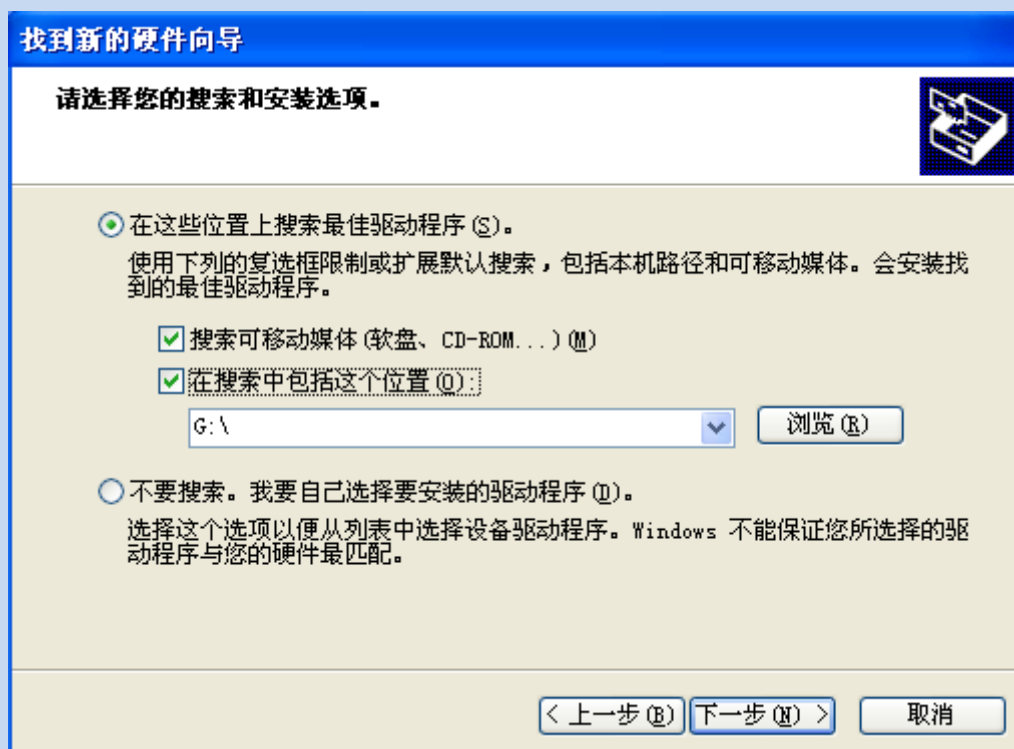
接下来我们开始安装下载器的驱动程序，首先保证硬件连接正常，将 USB 线的 A 型接头插入电脑的 USB 接口，此时出现：



选择“否，暂时不”，点击“下一步”：



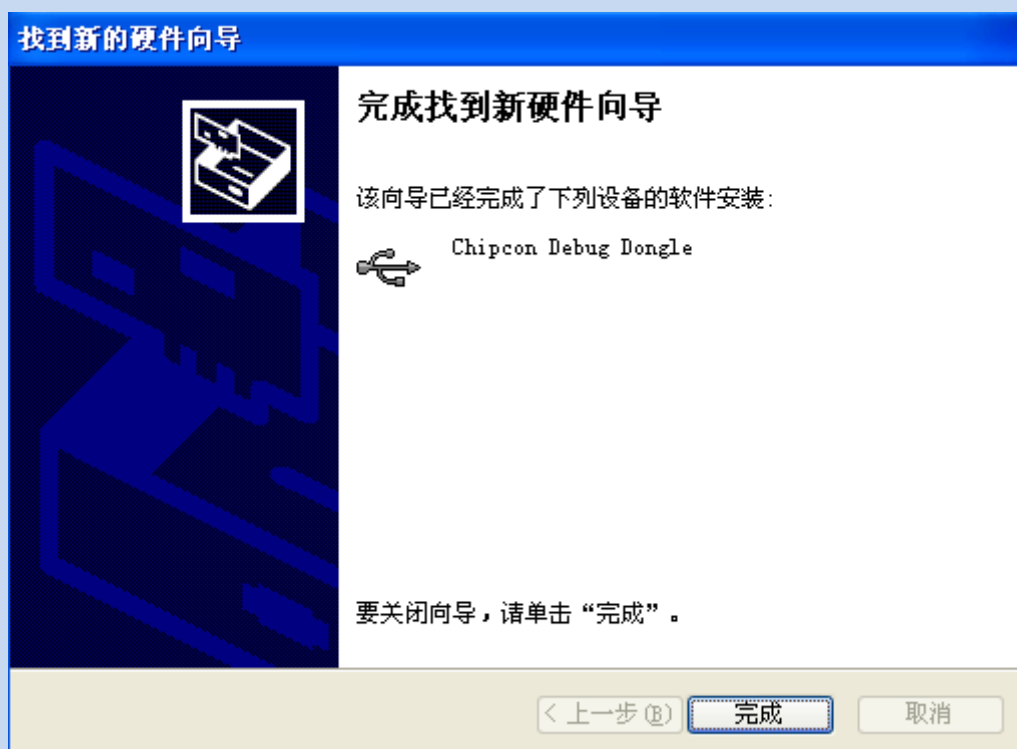
选择“从列表或指定位置安装”，点击“下一步”：



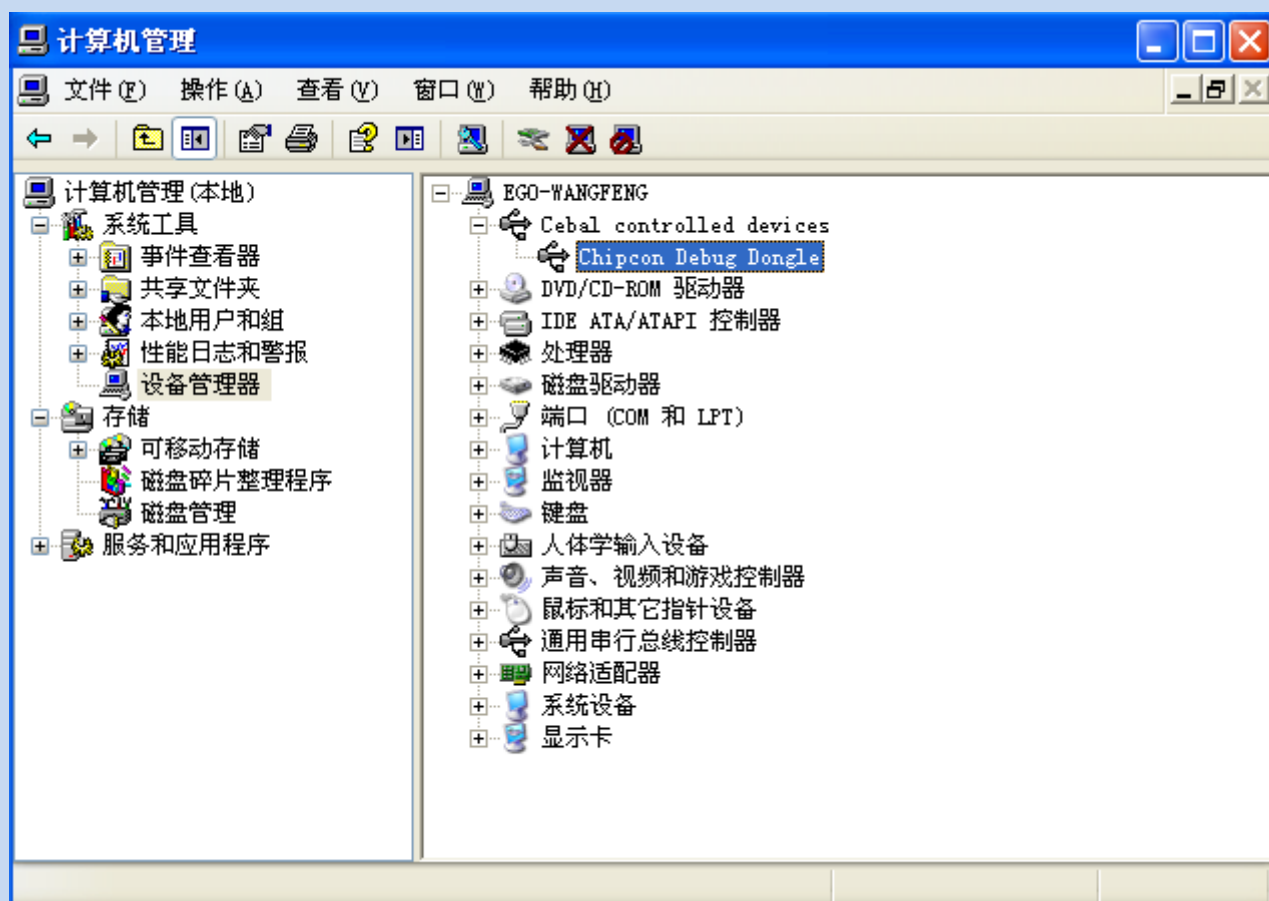
点击“浏览”，查找下载器的驱动程序，目录为 C:\Program Files\IAR Systems\Embedded Workbench 5.3\8051\drivers\Texas Instruments：



直至：



最后点击“完成”。打开电脑的计算机管理，可以发现增加了一个新的设备（下图高亮部分）：



第五章 ZigBee 开发套件

锋硕电子最新推出的无线 ZigBee2007 传感器网络系统主要有：1、下载器，2、协调器节点，3、路由节点，4、终端节点。用户可以根据自己的需求选择合适的节点，很方便的实现传感器网络，无线化，网络化，规模化的演示，观测和再次开发。

1. 下载器：提供 USB 接口连接电脑（PC），通过下载器 download 程序至各个节点。可以实现单步调试、断点调试、观察寄存器、观测程序执行和数据流。下载器与终端节点通过 4 芯 SPI 接口线连接可组成协议栈分析仪。
2. 协调器节点：完成通过计算机发送的指令发送或接收路由节点或者终端节点数据，并将接收到的数据发送给计算机。**锋硕电子**首次将 GPRS 功能集成至协调器节点，通过 GPRS 的功能可以方便的将数据无线传输至异地的服务器。
3. 路由节点：在协调器节点不能和所有的终端节点通信时，路由节点作为一种中介使协调器节点和终端节点通信，实现路由通信功能，同时路由器具有采集传感器数据功能。
4. 终端节点：完成对设备的控制和数据的采集，包括灯的控制温度、光敏、湿度、加速度、可调电阻等数据值等。

锋硕电子推出的最新版 ZigBee2007 系统适合用于无线传感监控，工业监控、楼宇自动化、数据中心、制冷监控、设备监控、社区安防、环境数据检测、仓库货物监控、农业蔬菜大棚等现在化农业数据监控、煤气水电抄表、智能家庭等领域。

在 ZigBee 网络中，每个节点都有指定的配置参数，从而确定其设备类型，不同的设备类型，在网络中有着不一样网络任务。在属于多跳网络的 ZigBee 网络中，两个节点需要完成数据传输，可能需要经过其他中间节点的协助，所以节点的类型参数配络是非常必要的。从而使每个节点完成 (i) 执行指定的网络功能函数 (ii) 配置确定的参数到指定的值。网络功能的设置确

定了该节点的类型；参数配置为指定的值确定了堆栈的模式。

在 ZigBee 网络中，设备类型分为三类：协调器节点，路由器节点和终端设备

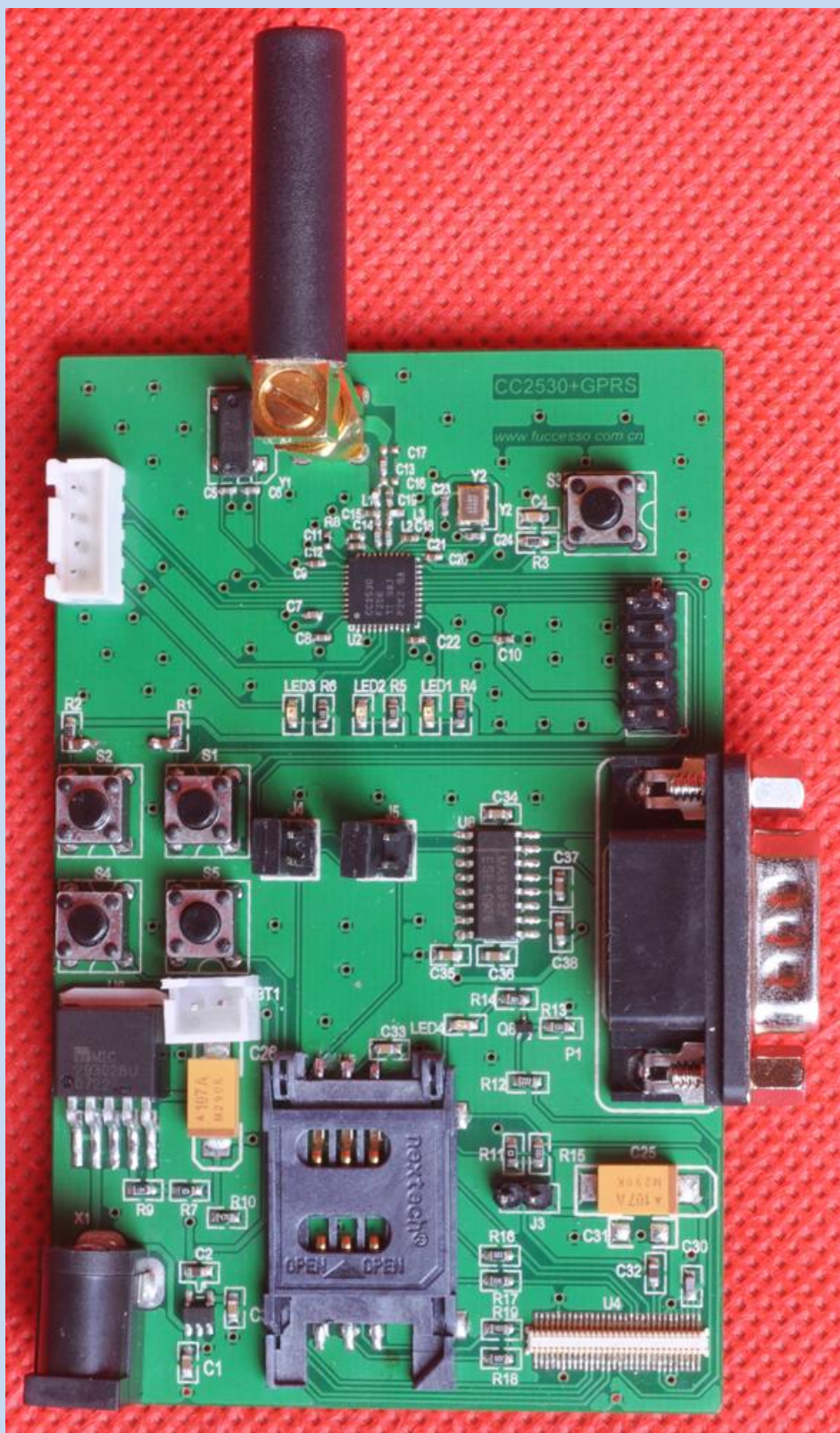
5.1 协调器节点

协调器节点是一个 ZigBee 网络的第一个开始的设备，或者是一个 ZigBee 网络的启动或建立设备。协调器节点选择一个信道和网络标志符（也叫 PAN ID），然后开始建立一个网络。

协调器节点在网络中可以使用，比如建立安全机制、网络中的绑定的建立等等。

注意：协调器节点主要的作用是建立一个网络和配置该网络的性质参数。一旦这些完成，该协调器节点就成为一个路由器节点，网络中的其他操作并不依赖该协调器节点，因为 ZigBee 是分布式网络。

下图为锋硕电子最新推出的协调器节点实物图：



(图 3.1 协调器节点实物图)

其硬件资源清单如表 3.1 所示：

(表 3.1)

项目	名称	型号	说明
1	CPU	CC2530	内存 256M, 最大发射功率 4.5dBm
2	系统时钟	32M	高精度无源晶振
3	实时时钟	32.768kHz	无源晶振
4	电池	3.7V	3.7V 手机锂电池
5	LED	3 色	红、黄、绿 3 色 LED 灯
6	键盘	4 路	3 路按键
7	复位	2 路	CC2530 复位和 SIM900B 复位
6	SPI 接口	4 芯	
7	下载接口	10 芯	
8	串口接口	5 芯	UART 接口
9	ADC 接口	2 芯	
10	天线接口	SMA	
11	天线	2.4G 和 900M	2 根杆状天线
12	功放	CC2591	最高放大倍数 22dBm
13	GPRS 接口	SIM900B	GPRS 模块
14	SIM 卡座	SIM 卡	
15	电源接口	5V	5V 转 3.3V, 5V 转 3.7V, 2A

其性能指标如下表 3.2 所示:

(表 3.2)

项目	特性	备注
厚度	0.8mm	

形状	矩形	
尺寸	60mm*95mm	
频率	2.4GHz—2.485GHz	
通信速率	250kbit/s	
通信距离	>500m	视距条件下，点对点通信
穿透能力	3 堵钢筋混凝土	
射频瞬时功耗	<1mW	
电流	<166mA 发送数据 <27mA 接收数据	
电压	2V—3.6V	
运行时间	>100 天	每 10s 发送一次报文
灵敏度	-98dBm	
射频输出功率	10dBm—20dBm	9 级功率切换
工作模式切换时间	<120us	
运行温度	-40℃—85℃	
GPRS 模块	SIM900B	短信、彩信和数据收发

5.2 路由器节点

一个路由器的功能有（1）允许其他设备加入网络（2）多跳路由（3）辅助它的子节点完成通信。

一般来说，路由器需要一直处于工作状态，所以需要主干线供电（区别于电池供电）。但是在某指定的网络结构中可以采用电池供电，如“串树型”网络模式中，允许路由器周期的运

行操作，所以可以采用电池供电。

下图为锋硕电子最新推出的路由器节点实物图：



图 3.2 路由器节点实物图

其硬件资源清单如表 3.3 所示：

(表 3.3)

项目	名称	型号	说明
1	CPU	CC2530	内存 256M，最大发射功率 4.5dBm
2	系统时钟	32M	高精度无源晶振
3	实时时钟	32.768kHz	无源晶振
4	电池	3.7V	3.7V 手机锂电池
5	LED	3 色	红、黄、绿 3 色 LED 灯
6	键盘	2 路	2 路按键
7	复位	1 路	
6	SPI 接口	4 芯	
7	下载接口	10 芯	
8	串口接口	5 芯	UART 接口
9	ADC 接口	2 芯	
10	天线接口	SMA	
11	天线	2.4G	杆状天线和 PCB 天线
12	功放	CC2591	最高放大倍数 22dBm
13	电源接口	5V	5V 转 3.3V，200mA

其性能指标如下表 3.4 所示：

(表 3.4)

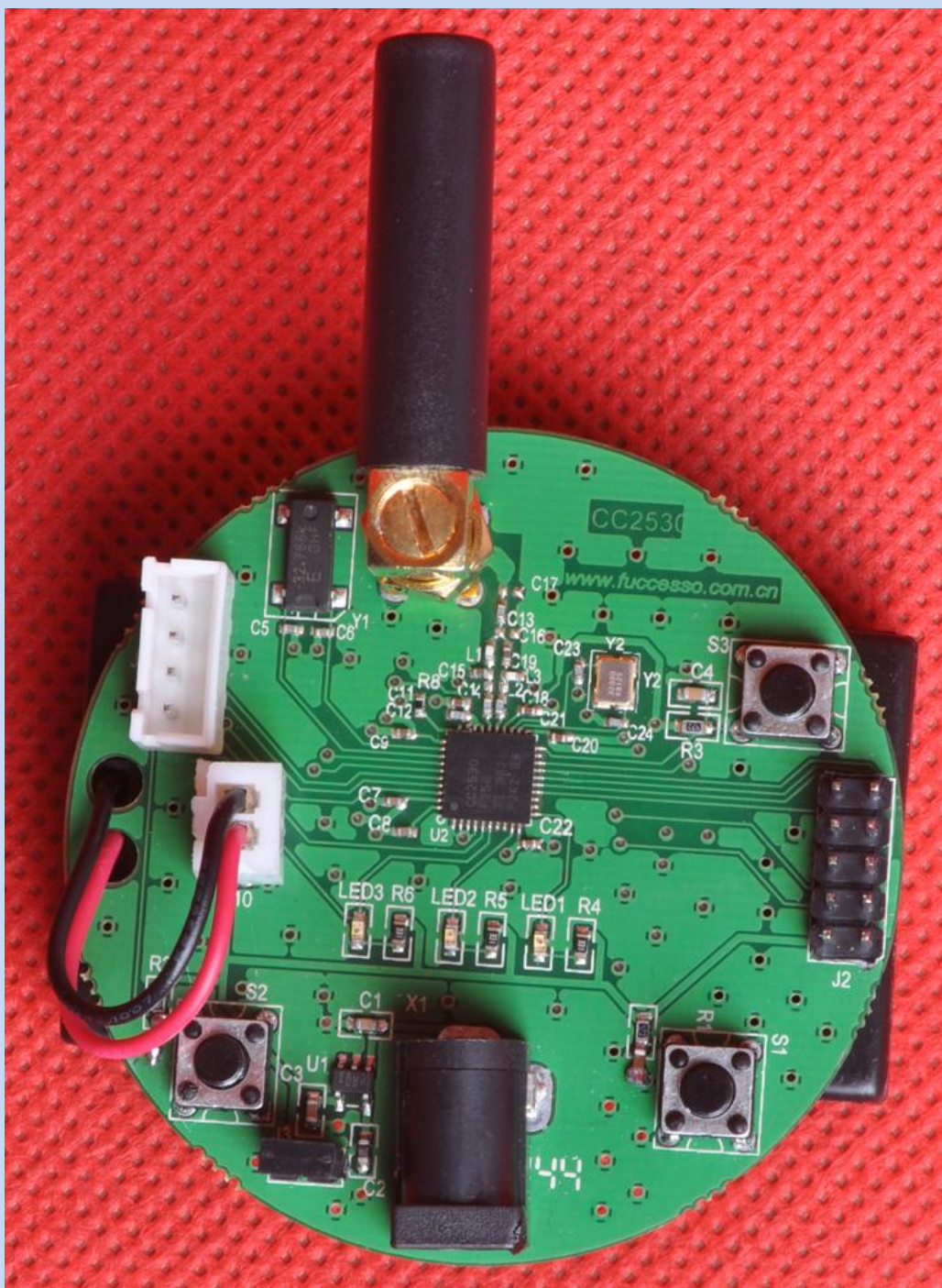
项目	特性	备注
厚度	1.6mm	

形状	矩形	
尺寸	43mm*55mm	
频率	2.4GHz—2.485GHz	
通信速率	250kbit/s	
通信距离	>500m	视距条件下，点对点通信
穿透能力	3 堵钢筋混凝土	
射频瞬时功耗	<1mW	
电流	<166mA 发送数据 <27mA 接收数据	
电压	2V—3.6V	
运行时间	>100 天	每 10s 发送一次报文
灵敏度	-98dBm	
射频输出功率	10dBm—20dBm	9 级功率切换
工作模式切换时间	<120us	
运行温度	-40℃—85℃	

5.3 终端设备

为了维持网络最基本的运行，对于终端设备没有指定的责任。也就是说，在一个基本网络中，终端设备没有必不可少性。所以它可以根据自己功能需要休眠或唤醒，因此为电池供电设备。一般来说，该设备需要的内存较少（特别是内部 RAM）。

下图为锋硕电子最新推出的终端设备实物图：



(图 3.3 终端设备实物图)

其硬件资源清单如表 3.5 所示：

(表 3.5)

项目	名称	型号	说明
----	----	----	----

1	CPU	CC2530	内存 256M, 最大发射功率 4.5dBm
2	系统时钟	32M	高精度无源晶振
3	实时时钟	32.768kHz	无源晶振
4	电池	1.5V	2 节 1.5V 干电池
5	LED	3 色	红、黄、绿 3 色 LED 灯
6	键盘	2 路	2 路按键
7	复位	1 路	
6	SPI 接口	4 芯	
7	下载接口	10 芯	
8	串口接口	5 芯	UART 接口
9	ADC 接口	2 芯	
10	天线接口	SMA	
11	天线	2.4G	杆状天线
12	电源接口	5V	5V 转 3.3V, 200mA

其性能指标如下表 3.6 所示:

(表 3.6)

项目	特性	备注
厚度	0.8mm	
形状	圆形	
尺寸	直径: 56mm	
频率	2.4GHz—2.485GHz	

通信速率	250kbit/s	
通信距离	>100m	视距条件下，点对点通信
射频瞬时功耗	<1mW	
电流	<29mA 发送数据 <24mA 接收数据	
电压	2V—3.6V	
运行时间	>100 天	每 10s 发送一次报文
灵敏度	-92dBm	
射频输出功率	0dBm—35dBm	
工作模式切换时间	<120us	
运行温度	-40℃—85℃	

5.4 外部接口

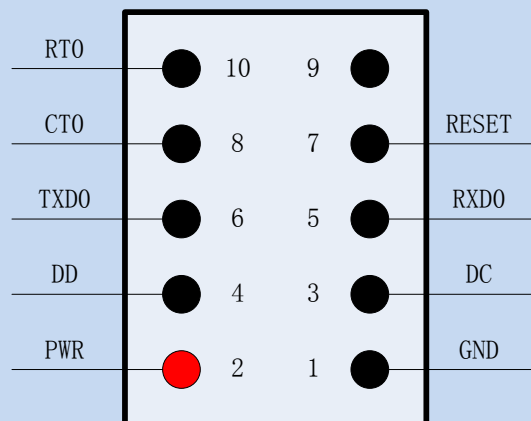
在协调器节点、路由节点和终端节点的电路板上都有十芯的下载器接口，其中 Pin3 和 Pin4 是下载线，Pin1 为 GND（接地），Pin2 为 PWR（3.3V 电源），Pin7 为 RESET（复位）。其余接口是扩展的 I/O 口。功能描述如下图所示，对应到 CC2530 的接口分别为：

Pin5：RXD0-CC2530 P0_2;

Pin6：RXD0-CC2530 P0_3;

Pin8：RXD0-CC2530 P0_4;

Pin10：RXD0-CC2530 P0_5;



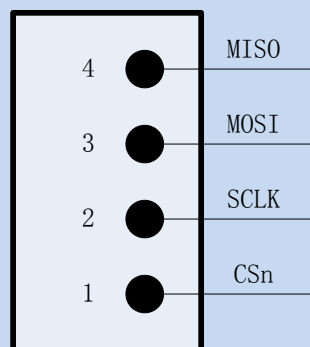
在协调器节点、路由节点和终端节点的电路板上都有四芯的 SPI 接口，功能描述如下图所示，对应到 CC2530 的接口分别为：

Pin1: RXD0-CC2530 P1_4;

Pin2: RXD0-CC2530 P1_5;

Pin3: RXD0-CC2530 P1_6;

Pin4: RXD0-CC2530 P1_7;



第六章 CC2530 概述

ZigBee 新一代 SOC 芯片 CC2530 是真正的片上系统解决方案，支持 IEEE 802.15.4 标准 /ZigBee/ZigBee RF4CE 和能源的应用。拥有庞大的快闪记忆体多达 256 个字节，CC2530 是理想 ZigBee 专业应用。支持新 RemoTI 的 ZigBee RF4CE，这是业界首款符合 ZigBee RF4CE 兼容的协议栈，和更大内存大小将允许芯片无线下载，支持系统编程。此外，CC2530 结合了一个完全集成的，高性能的 RF 收发器与一个 8051 微处理器，8 kB 的 RAM，32/64/128/256 KB 闪存，以及其他强大的支持功能和外设。

CC2530 提供了 101dB 的链路质量，优秀的接收器灵敏度和健壮的抗干扰性，四种供电模式，多种闪存尺寸，以及一套广泛的外设集——包括 2 个 USART、12 位 ADC 和 21 个通用 GPIO，以及更多。除了通过优秀的 RF 性能、选择性和业界标准增强 8051MCU 内核，支持一般的低功耗无线通信，CC2530 还可以配备 TI 的一个标准兼容或专有的网络协议栈(RemoTI，Z-Stack，或 SimpliciTI) 来简化开发，使你更快的获得市场。CC2530 可以用于的应用包括远程控制、消费型电子、家庭控制、计量和智能能源、楼宇自动化、医疗以及更多领域。

6.1 特性描述

● 强大无线前端

2.4 GHz IEEE 802.15.4 标准射频收发器。

出色的接收器灵敏度和抗干扰能力。

可编程输出功率为 +4.5 dBm，总体无线连接 102dbm。

极少量的外部元件。

支持运行网状网系统，只需要一个晶体。

6 毫米×6 毫米的 QFN40 封装。

适合系统配置符合世界范围的无线电频率法规：欧洲电信标准协会 ETSI EN300 328 和 EN 300 440（欧洲），FCC 的 CFR47 第 15 部分（美国）和 ARIB STD-T-66（日本）。

● 低功耗

接收模式：24 毫安。

发送模式 1dBm：29 毫安。

功耗模式 1（4 微秒唤醒）：0.2 毫安。

功率模式 2（睡眠计时器运行）：1 微安。

功耗模式 3（外部中断）：0.4 微安。

宽电源电压范围（2 V—3.6V）。

● 微控制器

高性能和低功耗 8051 微控制器内核。

32 /64 / 128 /或 256 /kB 系统可编程闪存。

8 KB 的内存保持在所有功率模式。

硬件调试支持。

● 外设

强大五通道 DMA。

IEEE 802.15.4 标准的 MAC 定时器，通用定时器（一个 16 位，2 个 8 位）。

红外发生电路。

32KHZ 的睡眠计时器和定时捕获。

CSMA/CA 硬件支持。

精确的数字接收信号强度指示/ LQI 支持。

电池监视器和温度传感器。

8 通道 12 位 ADC 在，可配置分辨率。

AES 加密安全协处理器。

两个强大的通用同步串口。

21 个通用 I/O 引脚。

看门狗定时器。

6.2 应用范围

2.4 GHz IEEE 802.15.4 标准系统。

RF4CE 遥控控制系统（需要大于 64 KB）。

ZigBee 系统/楼宇自动化。

照明系统。

工业控制和监测。

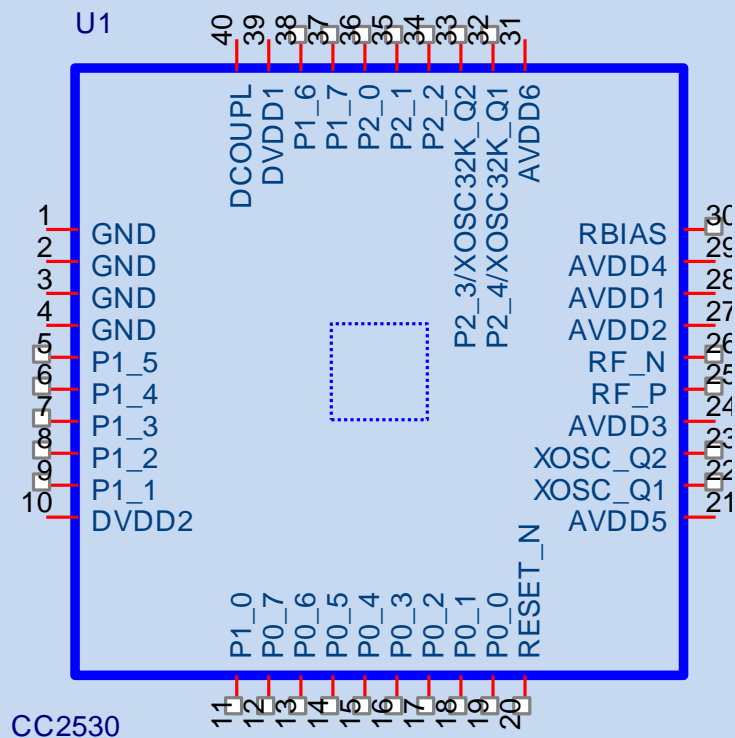
低功率无线传感器网络。

消费电子。

健康照顾和医疗保健。

6.3 引脚描述

CC2530 采用 40 脚 QFN 封装，其引脚图如下：



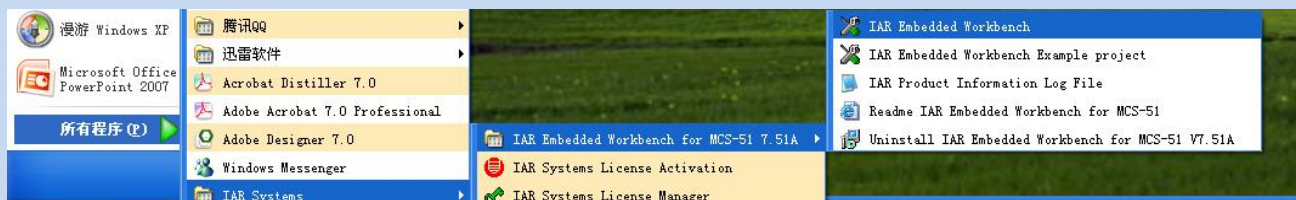
引脚说明如下表所示:

名称	序号	特性	描述
AVDD1	28	Power (A)	连接 2~3.6V 模拟电压源
AVDD2	27	Power (A)	连接 2~3.6V 模拟电压源
AVDD3	24	Power (A)	连接 2~3.6V 模拟电压源
AVDD4	29	Power (A)	连接 2~3.6V 模拟电压源
AVDD5	21	Power (A)	连接 2~3.6V 模拟电压源
AVDD6	31	Power (A)	连接 2~3.6V 模拟电压源
DCOUPL	40	Power (D)	1.8V 数字电源去耦, 不使用外部电源供电
DVDD1	39	Power (D)	连接 2~3.6V 数字电压源
DVDD2	10	Power (D)	连接 2~3.6V 数字电压源
GND	41	Ground	连接接地面
GND	1, 2, 3, 4	Ground	接地
P0_0	19	数字 I/O	端口 0.0 (ADC0)
P0_1	18	数字 I/O	端口 0.1 (ADC1)
P0_2	17	数字 I/O	端口 0.2 (ADC2/SPI0-MI/UART0-RX/SPI1-SS/UART1-CT/T1-0)
P0_3	16	数字 I/O	端口 0.3 (ADC3/SPI0-MO/UART0-TX/SPI1-C/UART1-RT/T1-1)
P0_4	15	数字 I/O	端口 0.4 (ADC4/SPI0-SS/UART0-CT/SPI1-MO/UART1-TX/T1-2)
P0_5	14	数字 I/O	端口 0.5 (ADC5/SPI0-C/UART0-RT/SPI1-MI/UART1-RX)
P0_6	13	数字 I/O	端口 0.6 (ADC6)
P0_7	12	数字 I/O	端口 0.7 (ADC7)
P1_0	11	数字 I/O	端口 1.0 (具有 20mA 驱动能力/T1-2)
P1_1	9	数字 I/O	端口 1.1 (具有 20mA 驱动能力/T1-1)
P1_2	8	数字 I/O	端口 1.2 (SPI0-SS/UART0-CT/T1-0)

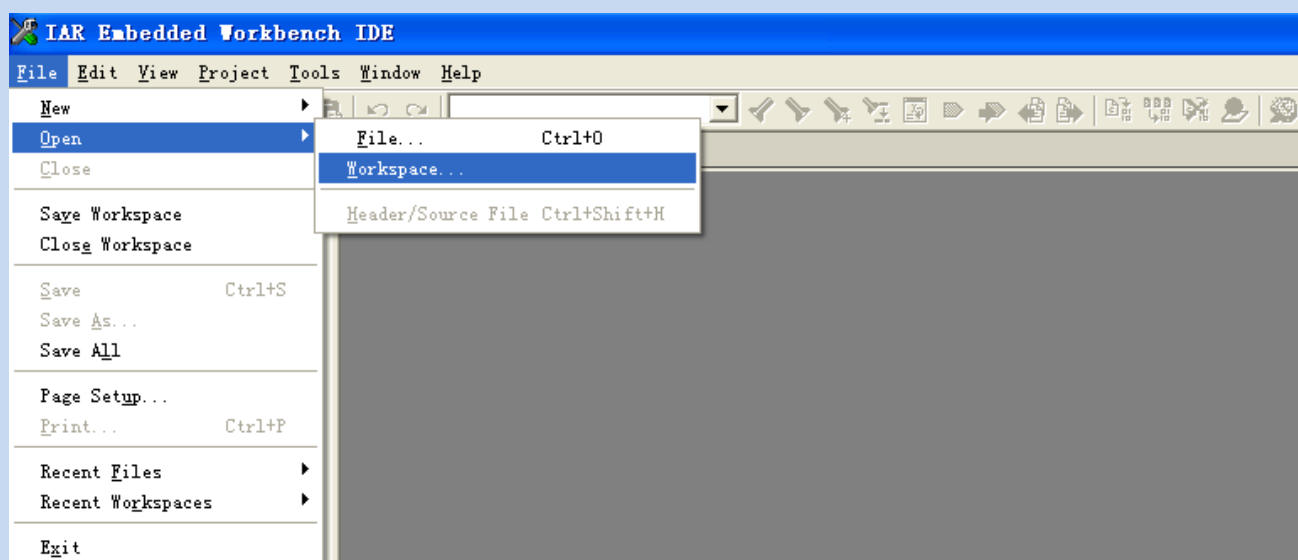
P1_3	7	数字 I/O	端口 1.3 (SPI0-C/UART0-RX/T3-0)
P1_4	6	数字 I/O	端口 1.4 (SPI0-MI/UART0-RX/SPI1-SS/UART1-CT/T3-1)
P1_5	5	数字 I/O	端口 1.5 (SPI0-MO/UART0-TX/SPI1-C/UART1-RT)
P1_6	38	数字 I/O	端口 1.6 (SPI1-MO/UART1-TX/T3-0)
P1_7	37	数字 I/O	端口 1.7 (SPI1-MI/UART1-RX/T3-1)
P2_0	36	数字 I/O	端口 2.0 (T4-0)
P2_1	35	数字 I/O	端口 2.1 (DD)
P2_2	34	数字 I/O	端口 2.2 (DC)
P2_3	33	数字 I/O	端口 2.3 (32.768kHz XOSC/T4-1)
P2_4	32	数字 I/O	端口 2.4 (32.768kHz XOSC)
RBIAS	30	模拟 I/O	参考电流的外部精密偏置电阻
RESET_N	20	数字 0	复位，低电平有效
RF_N	26	射频 I/O	差分射频端口
RF_P	25	射频 I/O	差分射频端口
XOSC_Q1	22	模拟 I/O	32MHz 晶振引脚 1
XOSC_Q2	23	模拟 I/O	32MHz 晶振引脚 2

第七章 IAR 使用介绍

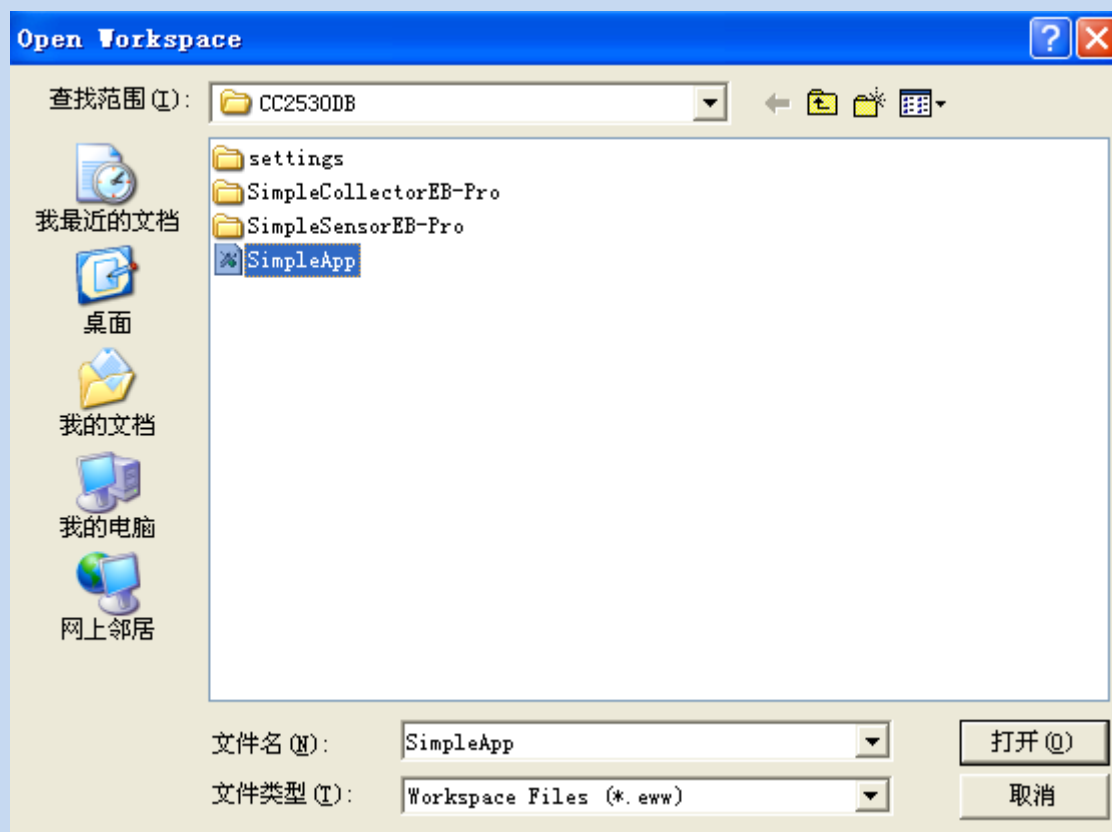
安装完 IAR7.51A 后，从开始菜单的打开 IAR Embedded Workbench，如下图所示：



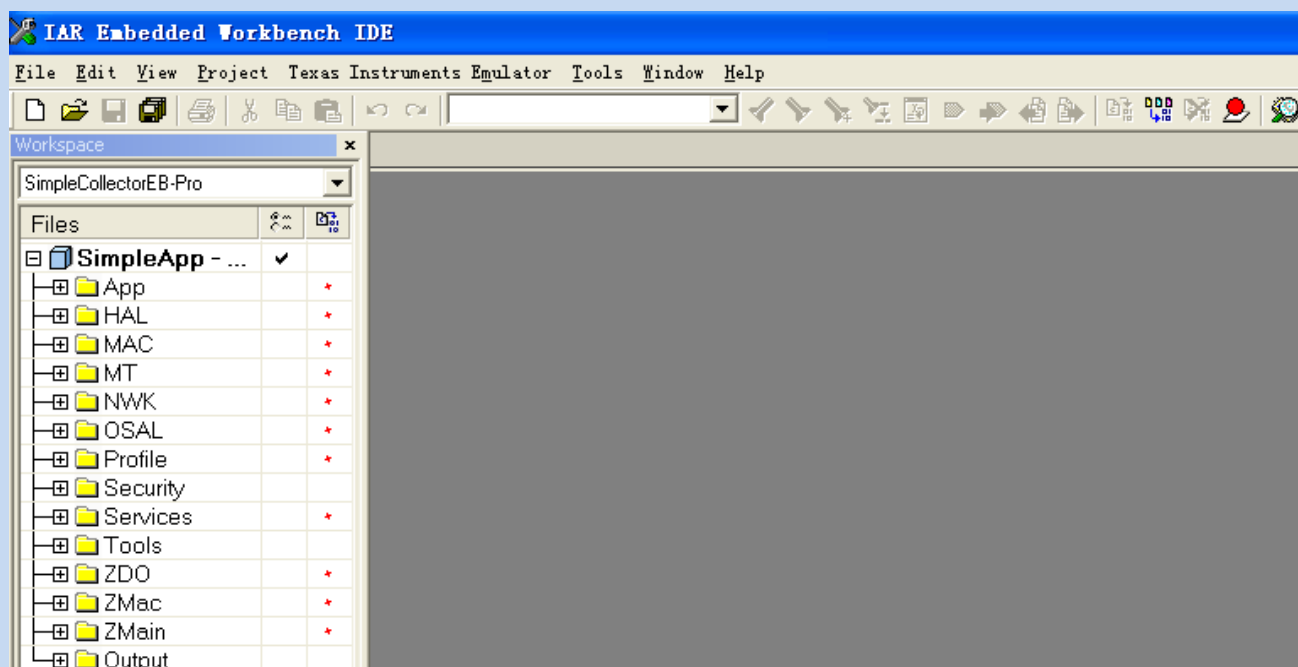
从 File->Open->Workspace 打开工程文件：



选择我们将要打开的工程文件，例如：需要打开《Collector_Sensor》工程文件，路径为：ZigBee2007 系统\源代码\ZStack-CC2530-2.3.1-1.4.0\Projects\zstack\Samples\Collector_Sensor\CC2530DB，如下图所示：



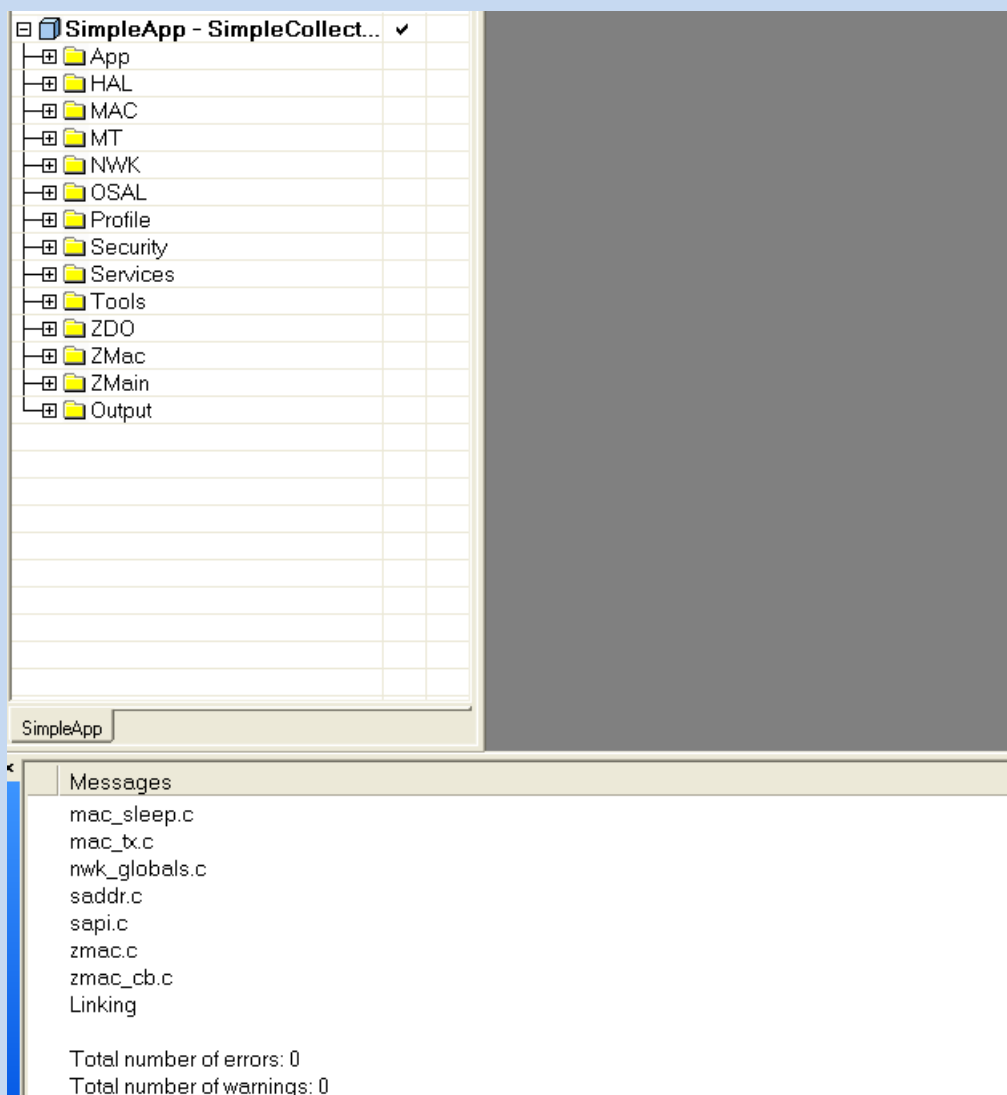
通过使用左方的 Workspace 下拉菜单，选择不同的设备类型。在 Collector_Sensor 工程中，采集节点 collector 是作为协调器节点，传感节点 Sensor 是作为终端节点。



另外，在首次打开时，会出现一系列的红色“*”，这是因为我们还没有编译工程。点击编译按钮：



可以发现红色 “*” 将消失，并且在下方出现编译的结果：



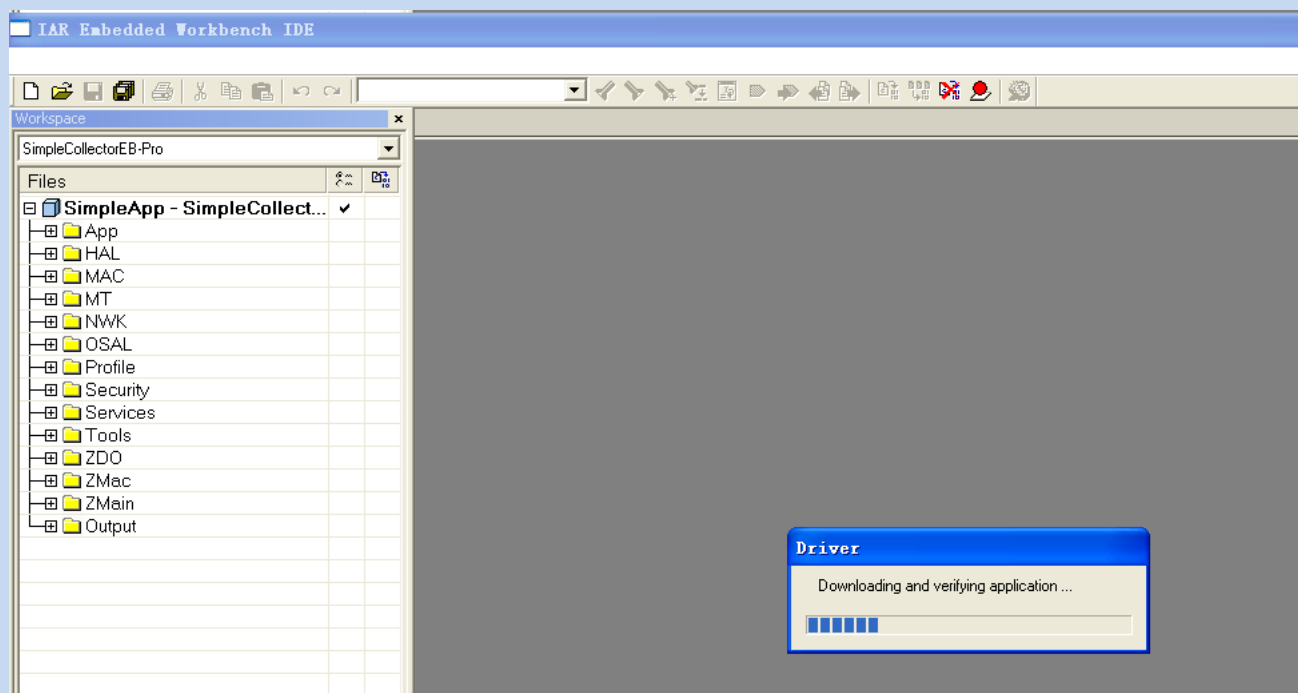
此时，我们就可以将此工程文件下载到节点的 CC2530 内存中，即，通过下载器烧写程序。请

特别注意：在烧写程序之前，必须手动按下下载器的复位按钮，然后再烧写。

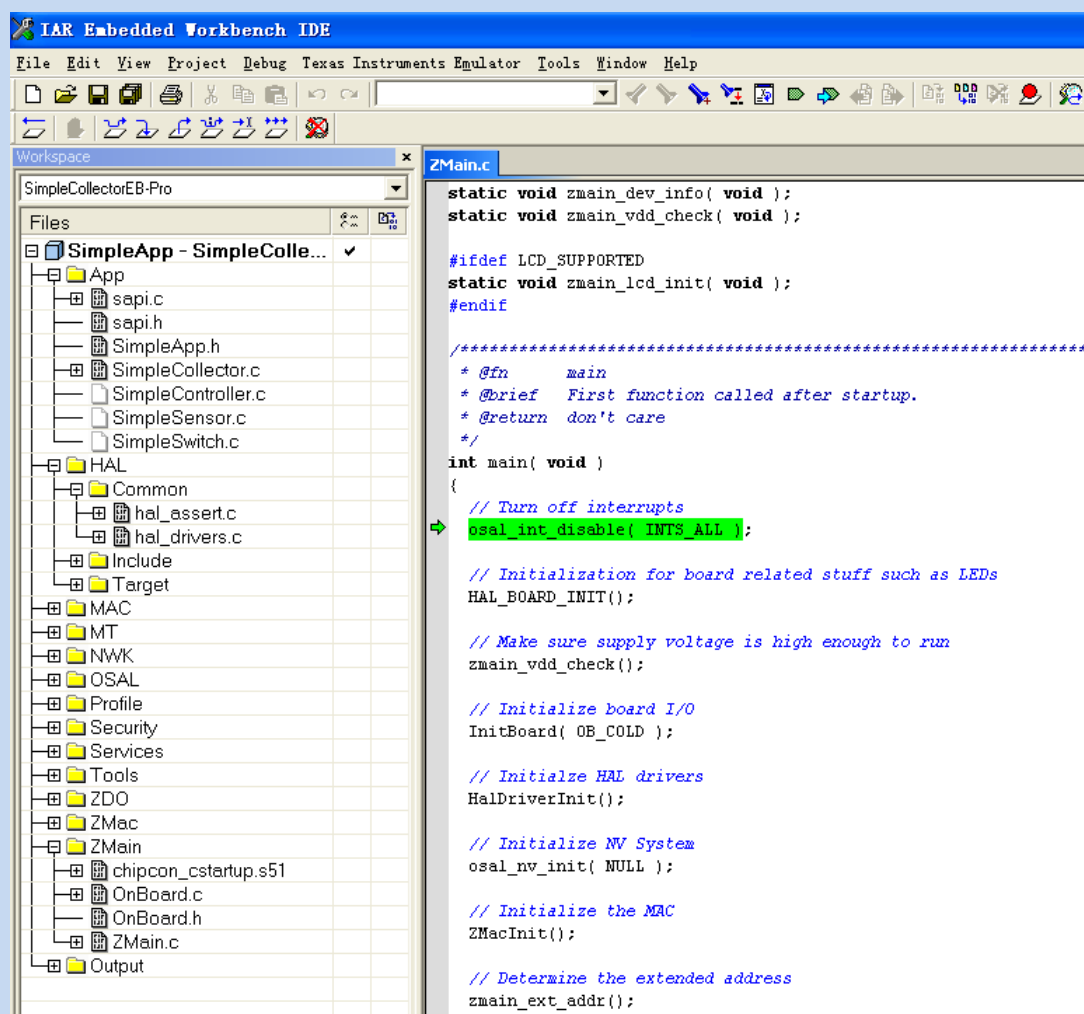
其实烧写的过程只须点击 IAR 的 “Debug” 按钮：



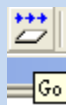
由于程序代码量较大，所以下载时请耐心等待：



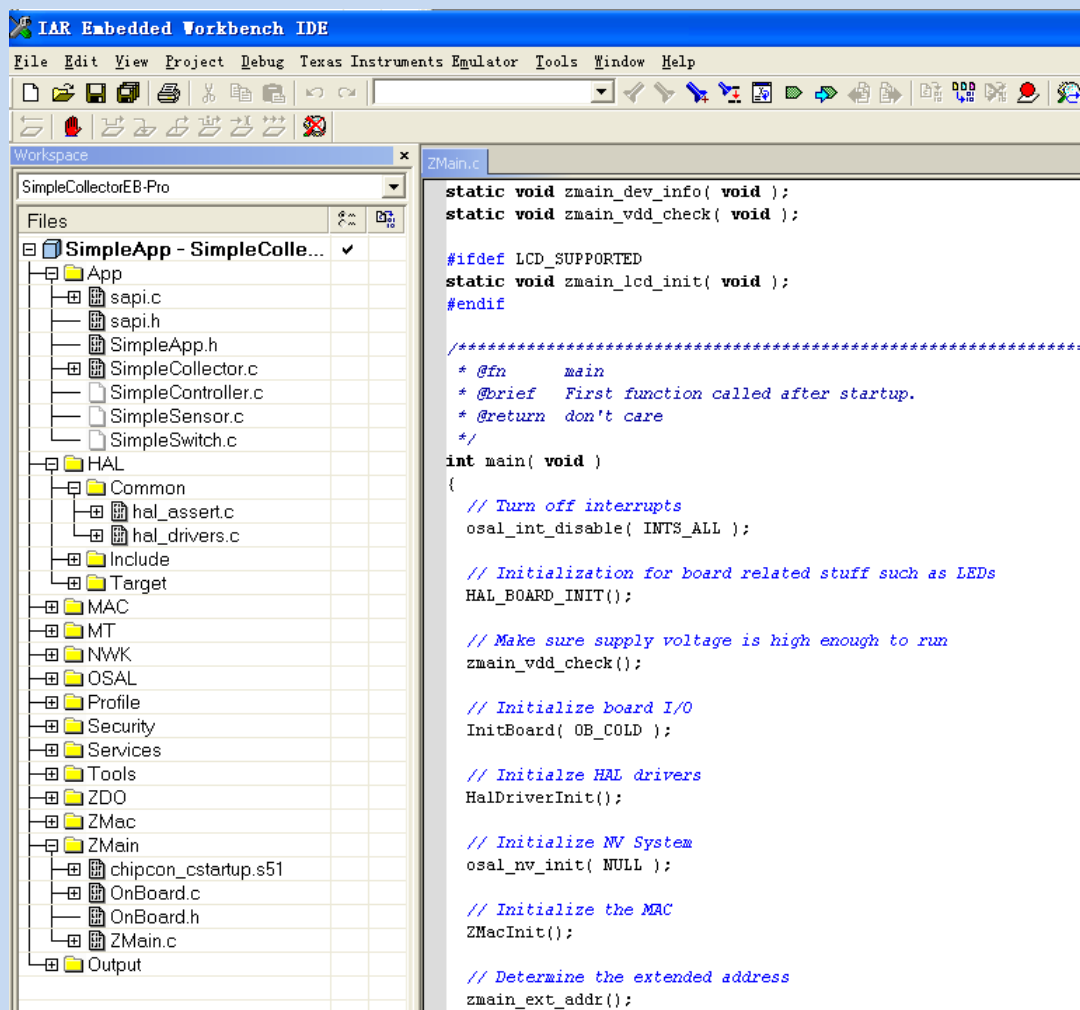
当烧写程序结束后，程序会跳到 main 函数的第 1 行开始执行：



点击左上方的“全速运行”按钮：



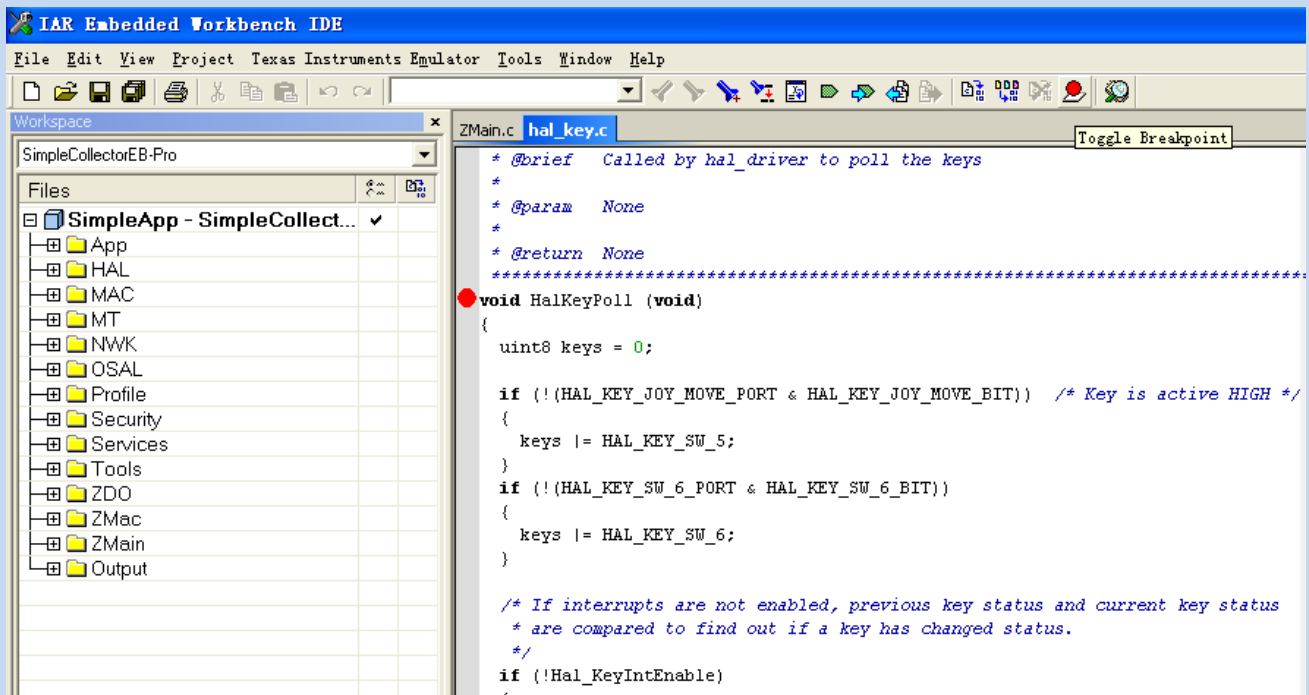
程序将运行起来：



在代码的调试阶段，可以使用 IAR 的断点调试功能。首先，设置需要观察的程序部分，插入断点。例如，需要调试按键轮询程序，鼠标的光标移动到需要插入断点的位置，点击断点插入按钮：



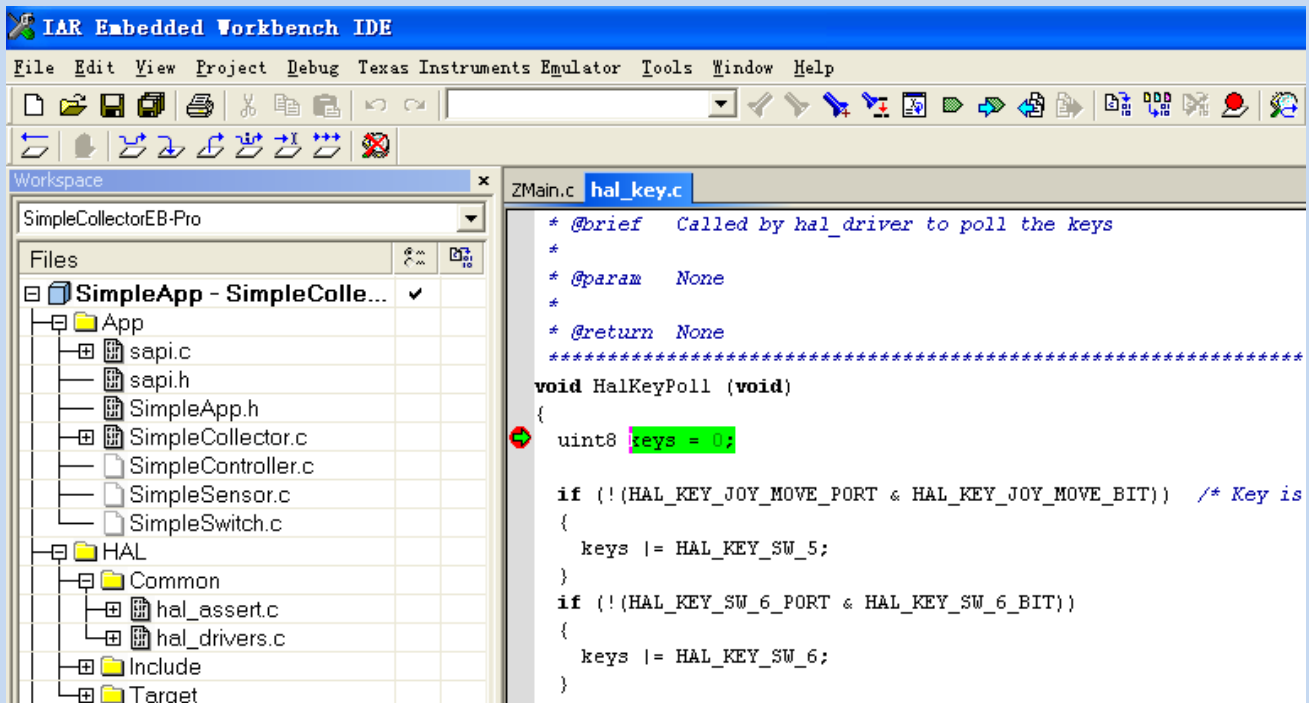
此时，在程序 HalKeyPoll(void)的左方出现红色标记：



然后点击下载程序，点击断点调试运行：



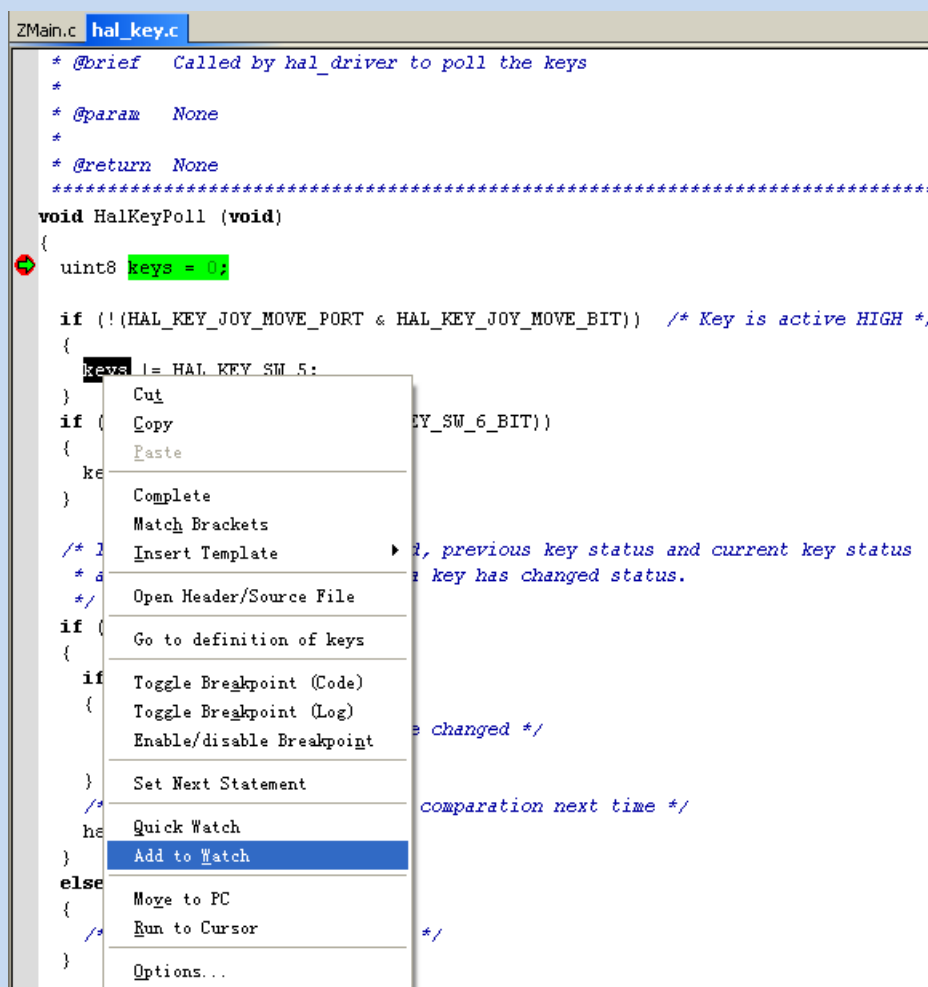
程序将运行到调试部分：



然后点击“单步运行”按钮，观察程序运行情况。



如果需要知道某些关键变量的值，可以将该变量添加到观察框中。例如，关心“keys”这个变量，可以用鼠标将“keys”选中，然后鼠标右键打开提示菜单，再点击“Add to Watch”：



此时，在右侧将出现观察框以及增添的变量：

