第 5 章 基于ZigBee的无线数据采集

5.1 ZigBee 技术介绍

ZigBee 是一种近距离、低复杂度、低功耗、低成本的双向无线通信技术。它主要用于距离短、功耗低且传输速率不高的各种电子设备之间的数据传输(包括典型的周期性数据、间歇性数据和低反应时间数据)。

ZigBee 的基础是 IEEE 802.15.4,但是 IEEE 802.15.4 仅处理低级的 MAC(媒体接入控制协议)层和物理层协议,ZigBee 联盟对网络层协议和应用层协议进行了标准化。

▶ 5.1.1 ZigBee 概述

1. ZigBee 的由来和发展

ZigBee 名字起源于蜜蜂之间传递信息的方式。蜜蜂通过一种特殊的肢体语言告知同伴新发现的事物源的位置信息,这种肢体语言是 Zigzag(之字形,Z 字形)舞蹈,借此意义以 ZigBee 作为新一代无线通信技术的命名。在此之前,ZigBee 也被称为 HomeRF Lite、RF-EasyLink 或 FireFly 无线电技术,现在统一称为 ZigBee 技术。ZigBee 模块类似于移动网络的基站,通信距离从几十米到几百米,并支持无线扩展。ZigBee 理论上可以是一个由 65 536 个无线模块组成的无线网络平台,在整个网络覆盖范围内,每一个 ZigBee 模块之间可以互相通信。

2003 年 12 月,Chipcon 公司推出第一款符合 2.4GHz IEEE 802.15.4 标准的射频收发器 CC2420,而后又有很多家公司推出与 CC2420 收发芯片相匹配的处理器,其中以 ATMEI 公司的 Atmega128 最成功(即常用方案是 Atmega128+CC2420)。

2004 年 12 月,Chipcon 公司推出全球第一个 IEEE 802.15.4 ZigBee 片上系统解决方案——CC2430 无线单片机。该芯片内部集成了一款增强型的 8051 内核,以及当时业内性能卓越的射频收发器 CC2420。2005 年 12 月,Chipcon 公司推出内嵌定位引擎的 ZigBee IEEE 802.15.4 解决方案 CC2431。2006 年 2 月,TI 公司收购 Chipcon 公司,又相继推出一系列的 ZigBee 芯片,比较有代表性的片上系统有 CC2530 等。

TI 公司在软件方面发展得比较快。2007 年 1 月,TI 公司宣布推出 ZStack 协议栈,目前已被全球众多 ZigBee 开发商所采用。ZStack 协议栈符合 ZigBee 2006 规范,支持多种平台,其中包括面向 IEEE 802.15.4/ZigBee 的 CC2430 片上系统解决方案、基于 CC2420 收发器的新平台以及 TI 公司的 MSP430 超低功耗控制器(MCU)。除此之外,ZStack 还支持具备定位感知特性的 CC2431。

2. 无线传感器网络与 ZigBee 的关系

(1) 无线传感器网络。

无线传感器网络是指大量的静止或移动的传感器以自组织和多跳的方式构成的无线网

络。其目的是协作地感知、采集和处理传输网络覆盖地理区域内感知对象的监测信息，并报告给用户。无线传感器网络起源于20世纪70年代，是一种特殊的无线网络，最早应用于美国军方，如空中预警控制系统。这种原始的传感器网络只能捕获单一信号，传感器节点只能进行简单的点对点通信。

1980年，美国国防部高级研究计划局提出了分布式传感器网络项目，开启了现代无线传感器网络研究的先例。此项目由美国国防部高级研究计划局信息处理技术办公室主任Robert Kahn主导，并由卡耐基·梅隆大学、匹兹堡大学和麻省理工学院等大学研究人员配合，旨在建立一个由空间分布的低功耗传感器节点构成的网络。这些节点之间相互协作并自主运行，将信息送达处理的节点。

20世纪80～90年代，无线传感器网络的研究依旧主要应用于军事领域，并成为网络中心站思想中的关键技术。1994年，加州大学洛杉矶分校的Willian J. Kaiser教授向美国国防部高级研究计划局提交了研究建议书《低功率无线集成微传感器》，以便于深入研究无线传感器网络。1998年，G. J. Pottie从网络的研究角度重新阐释了无线传感器网络的科学意义。同年，美国国防部高级研究计划局投入巨资，启动SensIT项目，目标是实现“超视距”战场监测。1999年9月，美国《商业周刊》将无线传感器网络列入21世纪最重要的21项技术之一，被认为是21世纪人类信息研究领域所面临的重要挑战之一。

(2) 无线传感器网络与ZigBee的关系。

无线传感器网络的应用，一般不需要很高的带宽，但对功耗要求却很严格，大部分时间必须保持低功耗。传感器节点通常使用存储容量不大的嵌入式处理器，对协议栈的大小也有严格的限制。另外，无线传感器网络对网络安全性、节点自动配置和网络动态重组等方面也有一定的要求。无线传感器网络的特殊性对应用于该技术的协议提出了较高的要求，无线传感器网络的物理层和MAC层目前使用最广泛的协议为IEEE 802.15.4。

无线传感器网络与ZigBee技术之间的关系可以从两方面进行分析：一是协议标准；二是应用。其具体关系如下所述。

① 从协议标准来讲，目前大多数无线传感器网络的物理层和MAC层都采用IEEE 802.15.4协议标准。IEEE 802.15.4描述了低速率无线个人局域网的物理层和媒体接入控制(MAC)层协议，属于IEEE 802.15.4工作组，而ZigBee技术是基于IEEE 802.15.4标准的无线技术。

② 从应用上来讲，ZigBee适用于通信数据量不大、数据传输速率相对较低、成本较低的便携或移动设备。这些设备只需要很少的能量，以接力的方式通过无线电波将数据从一个传感器传到另外一个传感器，并能实现传感器之间的组网，实现无线传感器网络分布式、自组织和低功耗。

从以上两个方面来讲，ZigBee是实现无线传感器网络应用的一种重要的技术。

3. ZigBee技术的特点

ZigBee可工作在2.4GHz(全球流行)、868MHz(欧洲流行)和915MHz(美国流行)三个频段上，分别具有最高250kb/s、20kb/s和40kb/s的传输速率，它的传输距离在10～75m的范围内。ZigBee作为一种无线通信技术，其特点如下所述。

(1) 低功耗。

低功耗是ZigBee重要的特点之一。一般的ZigBee芯片有多种电源管理模式，这些管理模式可以有效地对节点的工作和休眠进行配置，从而使得系统在不工作时可以关闭射频部分，

极大地降低了系统功耗，节约了电池的能量。

（2）低成本。

ZigBee 网络协议简单，可以在计算能力和存储能力都很有限的 MCU 上运行，非常适用于对成本要求苛刻的场合。现有的 ZigBee 芯片一般都是基于 8051 单片机内核，成本较低，这对于一些需要布置大量无线传感器网络节点的应用是很重要的。

（3）大容量。

ZigBee 设备既可以使用 64 位 IEEE 网络地址，又可以使用指配的 16 位网络短地址，在一个单独的 ZigBee 网络内，理论上可以容纳最多 65 536 个设备。

（4）可靠。

无线通信是共享信道的，因而面临着众多有线网络所没有的问题。ZigBee 在物理层和 MAC 层采用 IEEE 802.15.4 协议，使用带时隙或不带时隙的“载波检测多址访问/冲突避免”（CSMA/CA）的数据传输方法，并与“确认和数据检验”等措施相结合，可保证数据的可靠传输。同时，为了提高灵活性和支持在资源匮乏的 MCU 上运行，ZigBee 支持三种安全模式。最高级安全模式采用属于高级加密标准（AES）的对称密码和公开密钥，可以大大提高数据传输的安全性。

（5）时延短。

针对时延敏感做了优化，通信时延和从休眠状态激活的时延都非常短。

（6）灵活的网络拓扑结构。

ZigBee 支持星状、树状和网状拓扑结构，既可以单跳，又可以通过路由实现多跳的数据传输。

▶ 5.1.2 ZigBee 协议栈

协议是一系列通信标准，通信双方需要按照这一标准进行正常的数据发送和接收。协议栈是协议的具体实现形式，通俗地讲，协议栈就是协议与用户之间的一个接口，开发人员通过使用协议栈来使用这个协议，进而实现无线数据的收发。

协议栈是对协议的实现，可以将其理解为代码、库函数，并可供上层的应用去调用，协议栈各层与应用之间是相互独立的。商业化的协议栈只向用户提供使用接口（这与互联网行业的 API 模式很像），就像当用户调用地图 API 时，不需要关心底层地图是如何根据位置或坐标绘制的，也不用关心协议栈底层是如何实现的，除非用户想做协议研究。每个厂家的协议栈是有区别的，如德州仪器（TI）的 BLE 协议栈与 Nordic 的 BLE 协议栈就有很大的不同。ZigBee 协议栈是 ZigBee 无线网络规范的实现，ZigBee 联盟只会制定 ZigBee 规范，而不会去编写一个 ZigBee 协议栈。ZigBee 协议栈一般都由 ZigBee 芯片厂家来实现。

1. ZigBee 协议栈概述

ZigBee 协议栈可以分为 4 层：物理层（PHY）、媒体访问控制层（MAC）、网络层（NWK）及应用层（APL）。如图 5-1 所示，物理层和媒体访问控制层是由 IEEE 标准定义的，网络层、应用层及安全服务等是由 ZigBee 联盟规定的，应用对象是由设备厂商自行定义的。在 ZigBee 协议栈的图中，还可以发现有很多圆角矩形中几乎都带有“SAP”的字样。SAP 即服务接入点（Service Access Point），是协议栈中层与层之间的接口，协议栈都是分层结构的，服务接入点就是层与层之间的沟通渠道。协议栈相邻的上下层之间一般都有两个接口，也就是有两个 SAP。名字中带字母 D 的 SAP 是数据接口，负责层间的数据传输。名字中带字母 M 的 SAP

是管理接口，供上层或协议栈的管理平面对该层进行控制，如进行一些参数配置或读取状态等。ZigBee 协议栈，如图 5-1 所示。

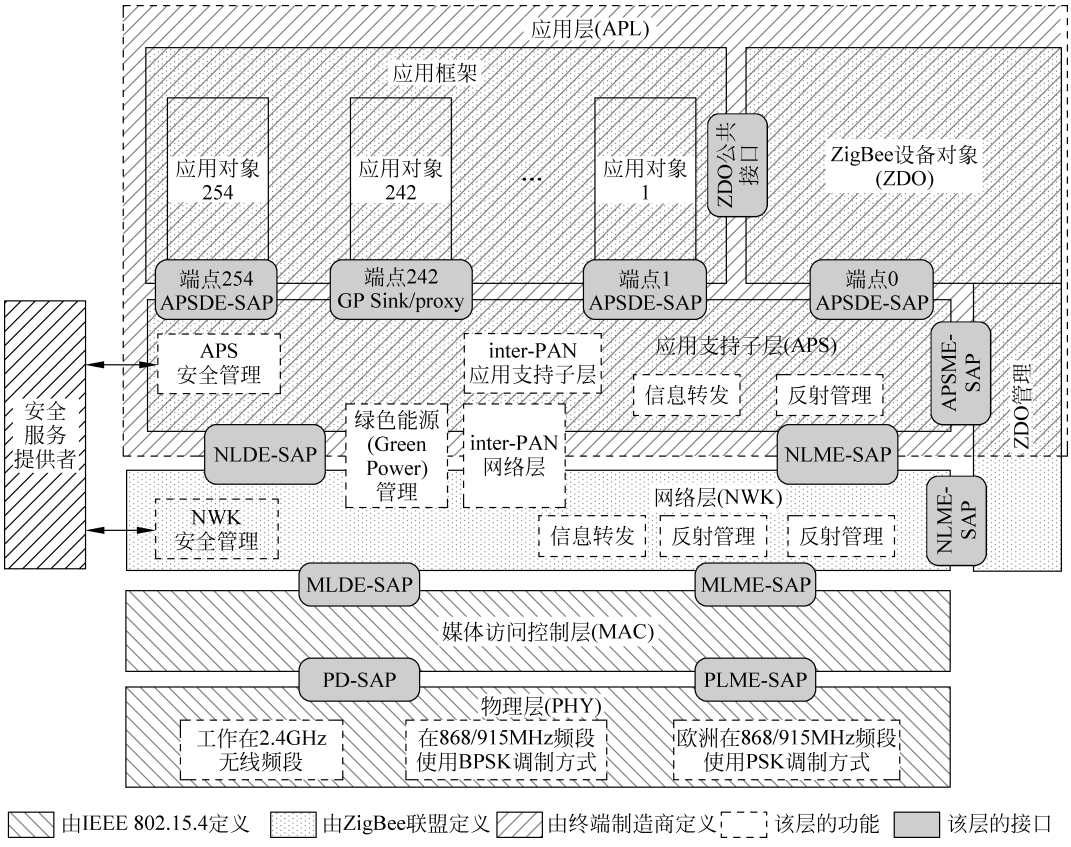

图 5-1 ZigBee 协议栈

ZigBee 协议栈的物理层和 MAC 层都是在 IEEE 802.15.4-2011 标准中定义的。PHY 层(物理层)规定了所使用的频段，以及所使用的编码、调制、扩频、调频等无线传输技术。有了物理层，就有了一个实现点到点的信号发射与接收的基础。没有物理层协议，设备之间根本无法通信，而且它们还可能根本就不在同一个频段上。MAC 层的主要作用是规定了无线信道的访问控制机制，也就是规定各个设备按照怎样的规则轮流使用信道。如果没有 MAC 层协议，当节点增多时，在通信没有规则的情况下，就会发生信号冲突，也就无法正常传输数据了。

ZigBee 协议栈在 IEEE 802.15.4 协议的基础上定义了网络层。网络层的主要作用是负责设备的连接和断开，并定义了在数据帧传递时所采用的安全机制，以及实现路由的发现和维护。简单地说，就是保障设备之间的组网和网络节点间的数据传输。ZigBee 技术支持多跳路由，可实现星状拓扑、树状拓扑和网状拓扑等不同的网络拓扑结构。

PHY、MAC、NWK 这三层协议主要是为上面的应用层服务的。在产品开发过程中，不需要深入涉及这三层协议的实现细节，应用层才是开发应关注的部分。

ZigBee 的应用层内部又分为三部分：应用框架、应用支持子层(APS)及 ZigBee 设备对象(ZDO)。应用框架中包含至少一个应用程序对象，也就是 ZigBee 设备的应用程序，是 ZigBee 产品开发人员所要实现的部分。

除了接口，协议栈中还经常会碰到实体(Entity)的概念。所谓实体，就是在协议栈中进行

数据发送或接收的一个基本组成部分，它可能是软件，也可能是硬件，具体与协议的实现有关，每一层协议都由一个或多个实体组成。

应用支持子层（APS）包括数据实体（APSDE）和管理实体（APSME），应用支持子层参考模型如图 5-2 所示，其作用就是向上层提供数据传输和管理配置的服务。应用支持子层隔离了应用框架与网络层，为应用程序的运行提供支撑。

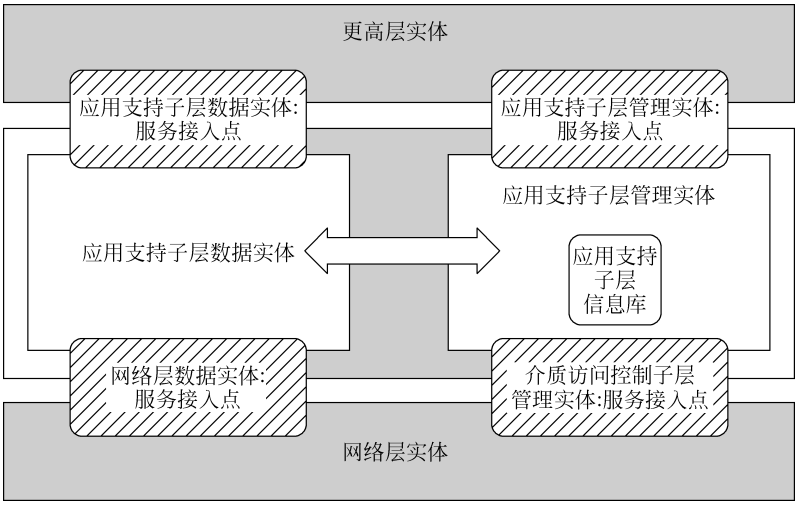

图 5-2　应用支持子层参考模型

APS 与应用程序之间的数据接口是 APSDE-SAP，最多可支持 240 个应用程序对象，通过端点（end point）来连接应用程序对象。每个端点都有一个编号，0 号端点比较特殊，是连接 ZDO 的。应用程序的端点编号从 1 号开始，一直到 240 号。一个 ZigBee 设备，如无线开关、无线灯具，一般会用 1～3 个端点。

ZigBee ZDO 可认为是一个特殊的应用程序对象，它满足了对应用程序对象的一般性要求。

ZDO 是一个特殊的应用层的端点，占用每个节点的端点 0。它是应用层其他端点与应用支持子层管理实体交互的中间件，主要功能集中在网络管理和维护上，具体功能如下。

① 初始化应用支持子层和网络层。

② 发现节点和节点功能。在无信标的网络中，加入的节点只对其父节点可见，而其他节点可以通过 ZDO 功能来确定网络的整体拓扑结构和节点所能提供的功能。

③ 安全加密管理。主要包括安全密钥的创建和发送，以及安全授权。

④ 网络维护功能。

⑤ 绑定管理。绑定功能由应用支持子层提供，但是绑定功能管理却是由 ZDO 提供。绑定管理确定了绑定表的大小、绑定的发起和绑定的解除等功能。

⑥ 节点管理。对于网络协调器和路由器，ZDO 提供网络监测、获取路由和绑定信息，以及发起离开网络过程等一系列节点管理功能。

2．原语概念

ZigBee 协议按照 OSI 开放系统互联的 7 层模型将协议分为一系列的层结构，各层之间通过相应的服务访问点来提供服务。这样，一方面，使得协议中的不同层能够根据各自的功能进行独立运作，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 设备要想正确无误地工作，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。

可以将服务原语看成 ZigBee 协议栈中层与层之间进行通信的方式。例如，如果应用层需要发送数据，它就会把数据以数据原语（如 Data_Request 原语）的形式发送给网络层（NWK），网络层（NWK）经过所定义的规则进行判断，决定是否发给 MAC 层，以此类推，直至发送给最底层的 PHY 层。

ZigBee 网络在工作时，各种不同的任务在不同的层次上执行，通过层的服务来完成所要执行的任务。一个层的服务是指通过在较低层上构建功能，为相邻的高层或子层用户所提供的功能。

各项服务通过服务原语来实现。每个事件由服务原语组成，它将在一个用户的某一层，通过该层的服务接入点（SAP）在建立了对等连接的用户的相同层之间传送。服务原语通过提供一种特定的服务来传输必需的信息。服务原语是一个抽象的概念，它们仅指出所提供服务的内容，而没有指出由谁来提供这些服务。它的定义与其他任何接口的实现无关。

由代表其特点的服务原语和参数描述来指定一种服务。一种服务可能有一个或多个相关的原语，这些原语构成了与具体服务相关的执行命令。在每种服务原语提供服务时，根据具体的服务类型，可能不带有传输信息，也可能带有多个传输所必需的信息参数。

原语通常分为如下 4 种类型（原语环境设置为：在一个具有 I 个用户的网络中，两个对等用户与 J 层或子层建立连接的原语）。

① Request。请求原语是从第 I_1 用户发送到它的第 J 层，请求服务开始。

② Indication。指示原语是从第 I_1 用户的第 J 层向第 I_2 用户发送，指出对第 I_2 用户有重要意义的内部 J 层的事件。该事件可能与一个遥远的服务请求有关，或者可能是由一个 J 层的内部事件引起。

③ Response。响应原语是从第 I_2 用户向它的第 J 层发送，用来表示用户对执行上一条原语调用过程的响应。

④ Confirm。确认原语是由第 J 层向第 I_1 用户发送，用来传递一个或多个前面服务请求原语的执行结果。

网络层确认原语通常都包括一个参数，这个参数记录了回答请求原语的状态。网络层状态参数值如表 5-1 所示。

表 5-1 网络层状态参数值

名　称	值	描　述
SUCCESS	0x00	请求执行成功
INVALID_PARAMETER	0xc1	从高层发出的原语无效或超出范围
INVALID_REQUEST	0xc2	考虑到网络层目前的状态，高层发送的请求原语无效或不能执行
NOT_PERMITTED	0xc3	NLME-JOIN. request 原语不被接受
STARTUP_FAILURE	0xc4	NLME-NETWORK-FORMATION. request 原语创建网络失败
ALREADY_PRESSENT	0xc5	产生 NLMEDIRECT-JOIN. request 原语的设备的邻居表中已经存在由地址设备提供的 NLMEDIRECT-JOIN. request 原语
SYNC_FAILURE	0xc6	用来表明在 MAC 层的 NLME-SYNC. request 原语失败

续表

名　　称	值	描　　述
NEIGHBOR_TABLE_FULL	0xc7	NLME-JOIN-DIRECTLY. request 失败，因为邻居表没有更多的空间
UNKNOWN_DEVICE	0xc8	NLME-LEAVE. request 原语失败，因为产生原语的设备地址不在邻居表中的参数列表中
UNSUPPORTED_ATTRIBUTE	0xc9	NLME-GET. request 或 NLME-SET. request 原语已以未知属性标识符的形式发出
NO_NETWORKS	0xca	在没有检测到网络的环境中发生 NLME-JOIN. request 原语
LEAVE_UNCONFIRMED	0xcb	确认“设备离开网络”失败
MAX_FRM_CNTR	0xcc	因为帧计数器达到最大值，所以输出帧安全处理失败
NO_KEY	0xcd	输出帧尝试进行安全处理且失败，因为安全处理无有效密钥
BAD_CCM_OUTPUT	0xce	输出帧尝试进行安全处理且失败，因为安全处理过程中产生了一个错误的输出
NO_ROUTING CAPACITY	0xcf	由于缺少路由表或路由发现表，因此尝试发现路由失败
ROUTE_DISCOVERY_FAILED	0xd0	尝试发现路由失败并非由于缺乏路由能力（除了缺少路由能力的原因外，其他原因也可造成发现路由失败）
ROUTE_ERROR	0xd1	由于发送设备的路由失败，因此 NLDE-DATA. request 原语失败
BT_TABLE_FULL	0xd2	由于广播事务表没有足够的空间，因此尝试发送一个广播帧或成员模式组播帧失败
FRAME_NOT_BUFFERED	0xd3	由于没有足够的缓存，使得请求传输一个数据失败。在路由发现之前，非成员模式组播帧被丢弃

3. 常见的 ZigBee 协议栈

常见的 ZigBee 协议栈分为三种：非开源的协议栈、半开源的协议栈和开源的协议栈。

（1）非开源的协议栈。

常见的非开源的 ZigBee 协议栈的解决方案包括 Freescale 解决方案和 Microchip 解决方案。

Freescale 公司最简单的 ZigBee 解决方案就是 SMAC 协议，是面向简单的点对点应用，不涉及网络概念。Freescale 公司完整的 ZigBee 协议栈为 BeeStack 协议栈，也是目前最复杂的协议栈，看不到具体的代码，只提供一些封装好的函数供直接调用。

Microchip 公司提供的 ZigBee 协议为 ZigBee PRO 和 ZigBee RF4CE，均是完整的 ZigBee 协议栈，但是收费偏高。

（2）半开源的协议栈。

TI 公司开发的 ZStack 协议栈是一个半开源的 ZigBee 协议栈，是一款免费的 ZigBee 协议栈，它支持 ZigBee 和 ZigBee PRO，并向后兼容 ZigBee 2006 和 ZigBee 2004。ZStack 内嵌了 OSAL 操作系统，使用标准的 C 语言代码和 IAR 开发平台，比较易于学习，是一款适合工业级应用的 ZigBee 协议栈。

（3）开源的协议栈。

Freakz 是一个彻底开源的 ZigBee 协议栈，它的运行需要配合 Contikj 操作系统，类似于 ZStack＋OSAL。Contikj 的代码全部用 C 语言编写，对于初学者来说比较容易上手，Freakz 适合用于学习，对于工业应用来讲 ZStack 比较实用。

▶ 5.1.3 常见的 ZigBee 模块

目前最常见的 ZigBee 芯片为 CC243X 系列、MC1322X 系列和 CC253X 系列。下面分别介绍三种系列芯片的特点。

1. CC243X 系列

CC2430/CC2431 是 Chipcon 公司(已被 TI 收购)推出的用来实现嵌入式 ZigBee 应用的片上系统。它支持 2.4GHz IEEE 802.15.4/ZigBee 协议,是世界上首个单芯片 ZigBee 解决方案。CC2430/CC2431 片上系统家族包括三个不同产品:CC2430-F32、CC2430-F64 和 CC2430-F128。它们的区别在于内置闪存的容量不同,以及针对不同 IEEE 802.15.4/ZigBee 应用做了不同的成本优化。

CC2430/CC2431 在单个芯片上整合了 ZigBee 射频前端、内存和微控制器。它使用一个 8 位 8051 内核,具有 32/64/128KB 可编程闪存和 8KB 的 RAM,还包含模拟数字转换器(ADC)、定时器、AES128 协同处理器、看门狗定时器、32kHz 晶振、休眠模式定时器、上电复位电路和掉电检测电路以及 21 个可编程 I/O 引脚。CC2430/CC2431 芯片的特点,如下所述。

(1) 高性能、低功耗 8051 微控制器内核。

(2) 极高的灵敏度及抗干扰能力。

(3) 强大的 DMA 功能。

(4) 外围电路只需极少的外接元件。

(5) 电流消耗小(当微控制器内核运行在 32MHz 时,RX 为 27mA,TX 为 25mA)。

(6) 硬件支持 CSMA/CA。

(7) 电源电压范围宽(2.0～3.6V)。

(8) 支持数字化接收信号强度指示器/链路质量指示(RSSI/LQI)。

2. MC1322X 系列

MC13224 是 MC1322X 系列的典型代表,是飞思卡尔公司研发的第三代 ZigBee 解决方案。MC13224 集成了完整的低功耗 2.4GHz 无线电收发器,内嵌了 32 位 ARM7 核的 MCU,是高密度、低元件数的 IEEE 802.15.4 综合解决方案,能实现点对点连接和完整的 ZigBee 网状网络。

MC13224 支持国际 802.15.4 标准以及 ZigBee、ZigBee PRO 和 ZigBee RF4CE 标准,提供了优秀的接收器灵敏度和较强的抗干扰性、多种供电模式以及一套广泛的外设集(包括 2 个高速 UART、12 位 ADC 和 64 个通用 GPIO、4 个定时器、I2C 等)。除了更强的 MCU 外,还改进了射频输出功率、灵敏度和选择性,提供了超越第一代 CC2430 的重要性能改进,而且支持一般低功耗无线通信,还可以配备一个标准网络协议栈(ZigBee,ZigBee RF4CE)来简化开发,因此可被广泛应用在住宅区和商业自动化、工业控制、卫生保健、消费类电子等产品中。其主要特性如下所述。

(1) 2.4GHz IEEE 802.15.4 标准射频收发器。

(2) 优秀的接收器灵敏度和抗干扰能力。

(3) 外围电路只需极少量的外部元件。

(4) 支持运行网状网系统。

(5) 128KB 系统可编程闪存。

(6) 32 位 ARM7TDMI-S 微控制器内核。

(7) 96KB 的 SRAM 及 80KB 的 ROM。

(8) 支持硬件调试。

(9) 4 个 16 位定时器及 PWM。

(10) 红外发生电路。

(11) 32kHz 的睡眠计时器和定时捕获。

(12) CSMA/CA 硬件支持。

(13) 精确的数字接收信号强度指示 LQI 支持。

(14) 温度传感器。

(15) 两个 8 通道 12 位 ADC。

(16) AES 加密安全协处理器。

(17) 两个高速同步串口。

(18) 64 个通用 I/O 引脚。

(19) 看门狗定时器。

3. CC253X 系列

CC253X 系列的 ZigBee 芯片主要是 CC2530/CC2531,它们是 CC2430/CC2431 的升级,在性能上要比 CC243X 系列稳定。CC253X 系列芯片是广泛使用于 2.4GB 片上系统的解决方案,建立在 IEEE 802.15.4 标准协议之上。其中,CC2530 支持 IEEE 802.15.4 以及 ZigBee、ZigBee PRO 和 ZigBee RF4CE 标准,且提供了 101 dB 的链路质量指示,具有优秀的接收器灵敏度和强抗干扰性。CC2531 除了具有 CC2530 强大的性能和功能外,还提供了全速的 USB 2.0 兼容操作,支持 5 个终端。

CC2530/CC2531 片上系统家族包括 4 个不同产品:CC2530-F32、CC2530-F64、CC2530-F128 和 CC2530-F256。和 CC243X 系列一样,它们的区别在于内置闪存的容量不同,以及针对不同 IEEE 802.15.4/ZigBee 应用做了不同的成本优化。

CC253X 系列芯片大致由三部分组成:CPU 和内存相关模块,外设、时钟和电源管理相关模块,以及无线电相关模块。

(1) CPU 和内存。

CC253X 系列使用的 8051CPU 内核是一个单周期的 8051 兼容内核。它有三个不同的存储器访问总线(SFR、DATA 和 CODE/XDATA),以单周期访问 SFR、DATA 和 SRAM。它还包括一个调试接口和一个中断控制器。

内存仲裁器位于系统中心,因为它通过 SFR 总线,把 CPU 和 DMA 的控制器和物理存储器与所有外设连接在一起。内存仲裁器有 4 个存取访问点,每次访问每一个可以映射到这三个物理存储器之一:8KB 的 SRAM、闪存存储器和一个 XREG/SFR 寄存器。它负责执行仲裁,并确定同时到同一个物理存储器的内存访问的顺序。

8KB SRAM 映射到 DATA 存储空间和 XDATA 存储空间的某一部分。8KB 的 SRAM 是一个超低功耗的 SRAM,当数字部分掉电时能够保留自己的内容,这对于低功耗应用是一个很重要的功能。

32/64/128/256KB 闪存块为设备提供了可编程的非易失性程序存储器,映射到 CODE 和 XDATA 存储空间。除了保存代码和常量,非易失性程序存储器允许应用程序保存必须保留的数据,这样在设备重新启动之后可以使用这些数据。

中断控制器提供了18个中断源，分为6个中断组，每组与4个中断优先级相关。当设备从空闲模式回到活动模式，也会发出一个中断服务请求。一些中断还可以从睡眠模式唤醒设备。

(2) 时钟和电源管理。

CC253X芯片内置一个16MHz的RC振荡器，外部可连接32MHz外部晶振。数字内核和外设由一个1.8V低差稳压器供电。另外，CC253X包括一个电源管理功能，可以实现使用不同的供电模式，用于延长电池的寿命，有利于低功耗运行。

(3) 外设。

CC253X系列芯片有许多不同的外设，允许应用程序设计者开发先进的应用。这些外设包括调试接口、I/O控制器、两个8位定时器、一个16位定时器和一个MAC定时器、ADC和AES协处理器、看门狗电路、两个串口和USB(仅限于CC2531)。

(4) 无线设备。

CC253X设备系列提供了一个与IEEE 802.15.4兼容的无线收发器，在CC253X内部主要由RF内核组成。RF内核提供了MCU和无线设备之间的一个接口，可以发出命令、读取状态、自动操作和确定无线设备的顺序。无线设备还包括一个数据包过滤和地址识别模块。

5.2 基于CC2530的单片机接口技术

CC2530是Chipcon公司(现已被TI收购)推出的用于IEEE 802.15.4/ZigBee和RF4CE应用的片上系统(SOC)。它能够以非常低廉的成本构建强壮的网络节点。CC2530有4种内存版本，即CC2530 F32/64/128/256，其分别具有32/64/128/256KB的内存。CC2530 F256结合了IT业界领先的ZigBee协议栈(ZStack)，提供了一个强大的完整的ZigBee解决方案。另外，CC2530 F64还提供了一个强大的完整的ZigBee RF4CE(消费电子射频通信标准)远程控制解决方案。

CC2530芯片的特点，如下所述。

(1) RF/布局。

① 支持2.4GHz IEEE 802.15.4 RF收发器。

② 极高的接收灵敏度和抗干扰性。

③ 可编程的输出功率高达4.5dBm。

④ 只需极少的外接元件。

⑤ 只需一个晶振，即可满足网状网络系统组网。

⑥ 6mm×6mm的QFN40封装。

⑦ 系统配置符合世界范围的无线电频率法规。

(2) 低功耗(低功耗是CC2530 SOC较为鲜明的特性)。

① 主动模式接收RX(CPU空闲)：24mA。

② 主动模式发送TX在1dBm(CPU空闲)：29mA。

③ 供电模式1(4μs唤醒)：0.2mA。

④ 供电模式2(睡眠定时器周期性唤醒)：1μA。

⑤ 供电模式3(外部中断，深度睡眠)：0.4μA。

⑥ 宽电源电压范围(2～3.6V)。

(3) 微控制器。

① 高性能和低功耗 8051 微控制器内核,具有代码预取功能。

② 32KB、64KB 或 128KB 的系统内可编程闪存。

③ 8KB RAM,在各种供电方式下的数据保持能力。

④ 支持硬件调试,配合 IAR 开发工具下调试。

(4) 外设。

① 强大的 5 通道 DMA,用于传输较大的数据块,如图像、语音等。

② 高性能集成运算放大器和超低功耗比较器。

③ IEEE 802.15.4 MAC 定时器(T2 定时器为 MAC 层定时器,尽量不要使用),三个通用定时器(T1:16 位定时器,T3,T4:8 位定时器)。

④ 红外(IR)发生电路。

⑤ 具有捕获功能的 32kHz 睡眠定时器,使用 32kHz 的晶振,用于低功耗模式。

⑥ 硬件支持 CSMA/CA,具有更高的可靠性。

⑦ 支持精确的数字化 RSSI/LQI。

⑧ 具有 8 通道可配置分辨率的 12 位 ADC。

⑨ 电池监视器和温度传感器。

⑩ AES 安全协处理器,硬件 AES 加解密。

⑪ 两个支持多种串行通信协议的强大 USART,并具有 SPI 和 UART 两种模式。

⑫ 21 个通用 I/O 引脚(19×4mA,2×20mA)。

⑬ 看门狗定时器,防止程序跑飞,避免系统死机现象。

(5) 应用。

① 支持 2.4GHz IEEE 802.15.4 系统。

② RF4CE 远程遥控系统(需要大于 64KB 闪存)。

③ ZigBee 系统(256KB 闪存)。

④ 家庭/楼宇自动化。

⑤ 照明系统。

⑥ 工业控制和监控。

⑦ 低功耗无线传感器网络。

⑧ 消费型电子。

⑨ 医疗保健。

(6) 开发工具。

① 低功耗 CC2530 开发套件。

② CC2530 ZigBee 开发套件。

③ 用于 RF4CE 的 CC2530RemoTI 开发套件。

④ SmartRF 程序烧写软件。

⑤ 数据包嗅探器 Packet Sniffer。

⑥ 可选用的 IAR 嵌入式工作台。

▶ 5.2.1 CC2530 硬件结构与工作原理

CC2530 组成架构如图 5-3 所示。

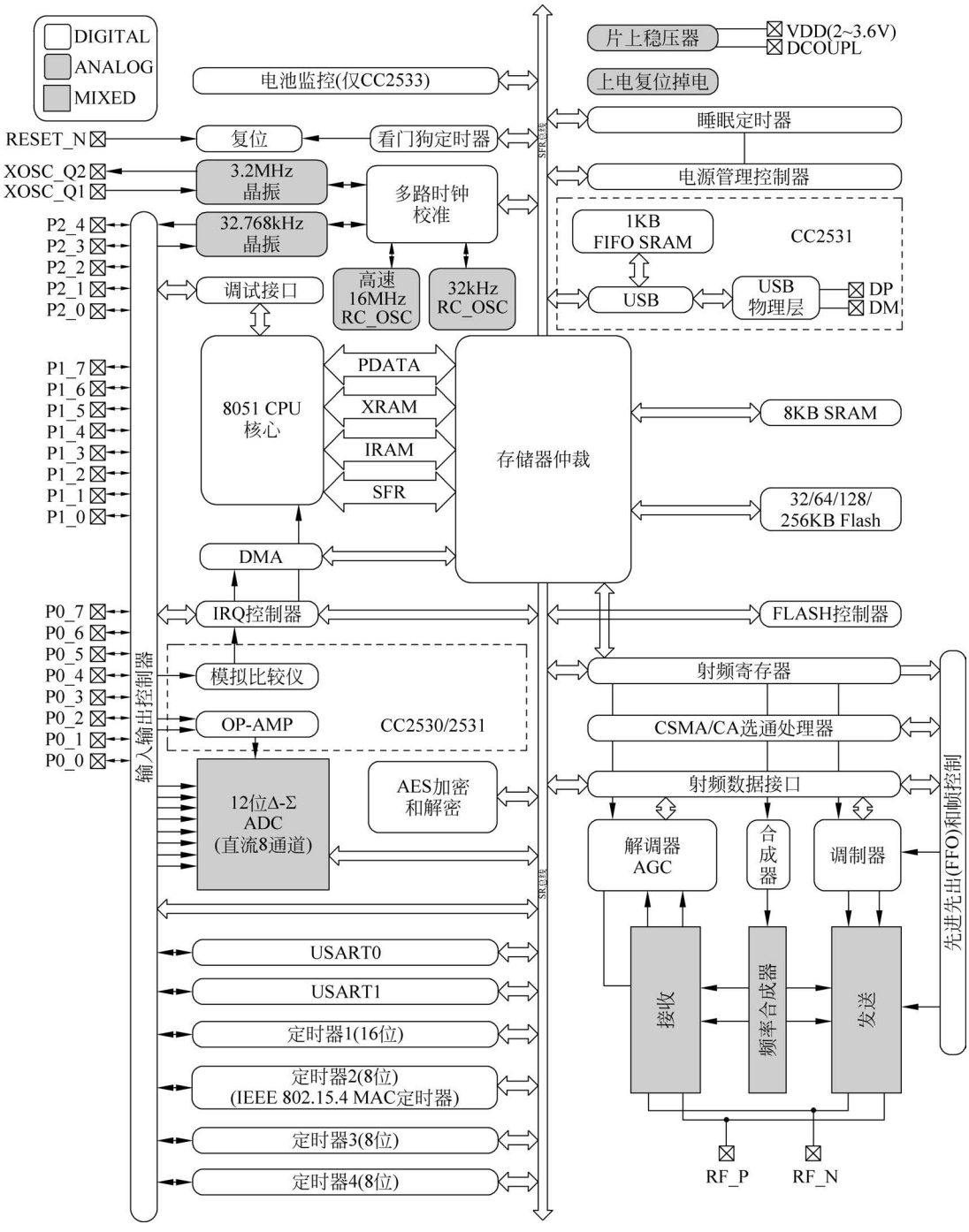

图 5-3 CC2530 组成架构

CC2530 内置业界领先的 RF 转发器，并结合了增强工业标准型 8051 MCU。CC2530 具有系统可编程的 256B 闪存、8KB RAM、两个 UART 接口并可复用 SPI 接口、8 通道 ADC、21 个 GPIO 口和其他强大功能。CC2530 具有不同的电源运行模式，非常适合超低功耗需求的系统。

CC2530 内部模块大致可以分为三种类型：CPU 和内存相关的模块；外设、时钟和电源管

理模块；射频相关模块。

1. 引脚描述

CC2530 封装以及引脚描述，如图 5-4 所示。

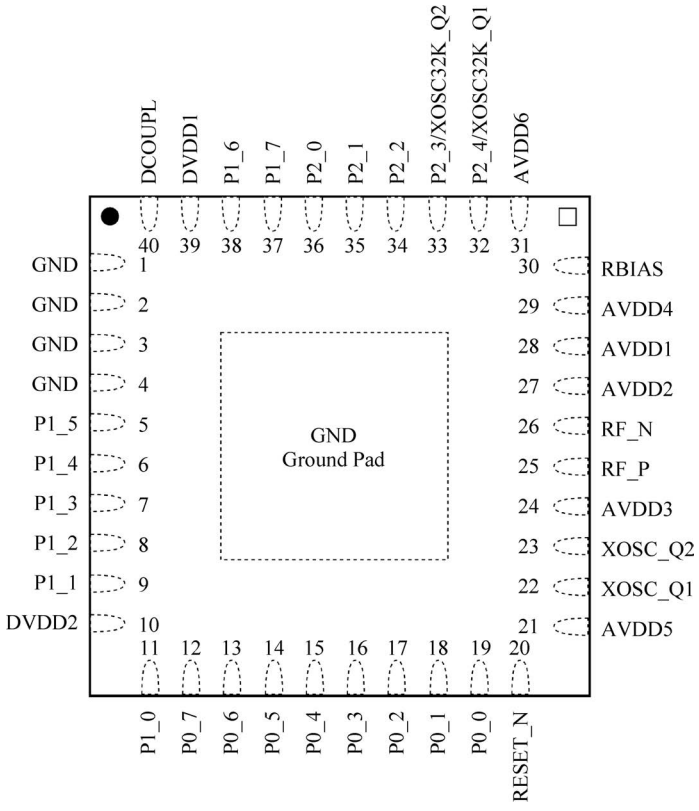

图 5-4　CC2530 RHA 封装(顶视图)

CC2530 共计 40 个引脚，主要包括 21 个 GPIO 引脚(P0_ 0～P0_7、P1_0～P1_ 7、P2_0～P2_4)、9 个模拟电源和数字电源引脚、2 个射频信号收发引脚、2 个连接在 32MHz 晶振的引脚、1 个连接偏置电阻的引脚、1 个复位引脚及 4 个未使用引脚。此外，CC2530 芯片还有一个接地引脚。CC2530 引脚如表 5-2 所示。

表 5-2　CC2530 引脚描述

引脚名称	引　脚	引脚类型	描　述
AVDD1	28	电源(模拟)	2～3.6V 模拟电源连接 (注：一般模块设计电源电压为 3.3V)
AVDD2	27	电源(模拟)	2～3.6V 模拟电源连接
AVDD3	24	电源(模拟)	2～3.6V 模拟电源连接
AVDD4	29	电源(模拟)	2～3.6V 模拟电源连接
AVDD5	21	电源(模拟)	2～3.6V 模拟电源连接
AVDD6	31	电源(模拟)	2～3.6V 模拟电源连接
DCOUPL	40	电源(数字)	1.8V 数字电源去耦 (注：引脚连接在一个电容上，用于稳压滤波)
DVDD1	39	电源(数字)	2～3.6V 数字电源连接
DVDD2	10	电源(数字)	2～3.6V 数字电源连接

续表

引脚名称	引　脚	引脚类型	描　述
GND	—	接地	接地必须连接到一个坚固的接地面，一般设计在模块的背面
GND	1,2,3,4	未使用的引脚	连接到 GND
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、T1-0)
P0_3	16	数字 I/O	端口 0.3(ADC3、SPI0-MO、UART0-TX、T1-1)
P0_4	15	数字 I/O	端口 0.4(ADC4、SPI0-SS、UART1-TX、T1-2)
P0_5	14	数字 I/O	端口 0.5(ADC5、SPI0-C、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 1.0，20mA 驱动能力
P1_1	9	数字 I/O	端口 1.1，20mA 驱动能力
P1_2	8	数字 I/O	端口 1.2
P1_3	7	数字 I/O	端口 1.3
P1_4	6	数字 I/O	端口 1.4(SPI1-SS、UART0-RX、T3-1)
P1_5	5	数字 I/O	端口 1.5(SPI1-C、UART0-TX)
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) 引脚 P2_1 和 2_2 为程序下载引脚
P2_3	33	数字 I/O	端口 2.3/32.768kHzXOSC(T4_1)
P2_4	32	数字 I/O	端口 2.4/32.768kHzXOSC(低功耗模式，晶振)
RBIAS	30	模拟 I/O	外部高精度偏置电阻
RESET_N	20	数字输入	复位引脚
RF_N	26	RF	射频信号收发引脚
RF_P	25	RF	射频信号收发引脚。 射频收发引脚连接的电路涉及高频信号处理，而将 CC2530 RF 差分信号转为单端信号发送到其他节点是最小模块设计较难的部分
XOSC_Q1	22	模拟 I/O	32MHz 石英晶振
XOSC_Q2	23	模拟 I/O	32MHz 石英晶振

2. CPU 与内存

CC253X 设备系列所使用的 8051CPU 内核是一个单周期的 8051 兼容内核。它有三个不同的存储器访问总线(SFR、DATA 和 CODE/XDATA)，以单周期访问 SFR、DATA 和主 SRAM。它还包括一个调试接口和一个 18 位输入的扩展中断单元。

内存仲裁器位于系统中心，它通过 SFR 总线把 CPU 和 DMA 控制器和物理存储器以及所有外设连接起来。内存仲裁器有 4 个内存访问点，每次访问可以映射到三个物理存储器之一，它们是 8KB SRAM、闪存存储器和 XREG/SFR 寄存器。它负责执行仲裁，并确定存储器同时访问同一个物理存储器之间的顺序。

8KB SRAM 映射到 DATA 存储空间和部分 XDATA 存储空间。8KB SRAM 是一个超低功耗的 SRAM，即使数字部分掉电，供电模式 2、3 也能保留其内容。对于低功耗应用来说，

这是很重要的一个功能。

32/64/128/256KB 闪存块为设备提供了内电路可编程的非易失性程序存储器，并映射到 XDATA 存储空间。除了保存程序代码和常量以外，非易失性存储器允许应用程序保存必须保留的数据，这样在设备重启之后还可以继续使用这些数据。当使用这个功能时，利用已经保存在网络中的具体数据，设备就不需要经过完全启动、网络寻找和加入的过程。

(1) 8051 CPU。

CC2530 集成了增强工业标准型 8051 内核(8 位字宽度)，并使用标准的 8051 指令集。但它的指令执行速度比标准的 8051 快，主要原因如下。

① CC2530 的每个指令周期是 1 个时钟，而标准的 8051 每个指令周期是 12 个时钟。

② CC2530 取消了无用的总线状态。

由于指令周期与可能的内存存取一致，所以大多数单字节指令可以在一个时钟周期内完成。除了速度提高之外，增强型 8051 内核还在结构上有如下改善。

① 第二数据指针。

② 扩展了 18 个中断源。

CC2530 核心 8051 的目标代码兼容业界标准的 8051 微控制器。也就是说，8051 内核上的目标代码编译，与工业标准的 8051 编译器或汇编执行具有同等功能。但是，其使用了与其他类型 8051 不同的指令时序，因此，微控制器带有的时序循环程序需要修改。而且，诸如定时器和串行端口外设单元也与其他的 8051 内核不同。因此，包含使用外接设备单元特殊功能寄存器的指令代码不能够正常运行。

(2) 存储器。

8051 CPU 架构有 4 个不同的存储空间，且具有单独的用于程序存储和数据存储的存储空间。8051 的存储空间，如下所述。

① 代码(CODE)。只读存储空间，用于程序存储。存储空间地址为 64KB。

② 数据(DATA)。可存取存储空间，可以直接或间接被单个周期的 CPU 指令访问，从而允许快速存取。这个存储空间地址为 256B，数据存储空间的低 128B 可以直接或间接访问，而高 128B 只能间接访问。

③ 外部数据(XDATA)。可存取存储空间，通常需要 4～5 个指令周期来访问，故为慢速访问。该存储空间地址为 64KB。在硬件里访问外部数据存储也比数据访问要慢，这是因为在 CPU 内核中的代码存储空间和外部数据存储空间共享一条公共总线，并且从代码存储空间进行指令预存取不能和外部数据访问并行。

④ 特殊功能寄存器(SFR)。可存取寄存器存储空间，可以被单个的 CPU 指令访问。该存储空间由 128B 构成。对于 SFR 寄存器，它的地址可以被分成 8 等份，每个位仍然可以单独寻址。

这 4 个不同的存储空间在 8051 架构中都截然不同，但在 CC2530 中有一部分是重叠的，以缓解 DMA 传输和硬件调试操作。

▶ 5.2.2 CC2530 数据采集与传输

在实际应用中，可能经常需要从外围设备采集数据，如温湿度环境采集、工业设备、医疗设备监控等应用。因此，像温湿度采集设备、工业设备等外围设备应该提供 UART、SPI 和 USB 接口来传输数据。CC2530 芯片通过配置相关 I/O 寄存器来读取这些通过硬件连接的外围设备的数据，并通过 ZigBee 协议栈的软件调度，将这些数据从射频收发电路发送到其他节点。

1. I/O 口

CC2530 有 21 个数字 I/O 引脚，它们可以配置为通用数字 I/O 或外设 I/O 信号，以连接到 ADC、定时器或者 USART 等外部设备。这些 I/O 口的用途，可以通过一系列寄存器配置，由用户软件加以实现。

I/O 口具备如下重要特性。

(1) 21 个数字 I/O 引脚。

(2) 可以配置为通用 I/O 或外部设备 I/O。

(3) 输入口具备上拉或下拉能力。

(4) 具有外部中断能力。

如果需要外部设备产生中断，21 个 I/O 引脚都可以用作外部中断源输入口，外部中断功能也可以从睡眠模式唤醒设备。

2. DMA 控制器

CC2530 内置直接存储器存取(DMA)控制器，该控制器可以减轻 8051 CPU 内核在传送数据时的负担，实现 CC2530 在高效利用电源条件下的高性能。只需要 CPU 的极少干预，DMA 控制器就可以将数据从 ADC 或 RF 收发器等外部设备单元传送到存储器中。

CC2530 8051 处理器在传送语音、图像等较大数据时，使用 UART 串口通过中断的方式来查询数据，查询效率很低。这时，读者可以选用 DMA 控制器来完成数据传送。如果传输效果不能满足应用需求，还可利用多路 DMA 通道来实现。CC2530 提供了 5 个独立的 DMA 通道。

DMA 控制器协调所有的 DMA 传送，以确保 DMA 请求和 CPU 存取之间按照优先等级协调、合理地进行。DMA 操作流程如图 5-5 所示。

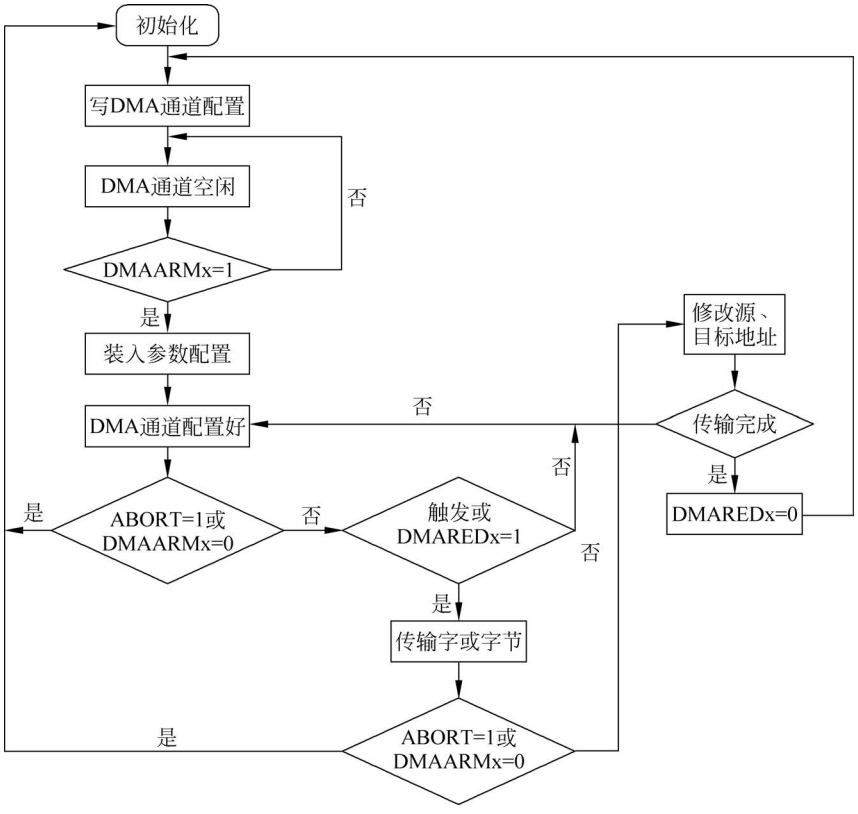

图 5-5 DMA 操作流程

DMA 控制器含有若干可编程设置的 DMA 通道，用来实现数据从存储器到存储器的传送，且 DMA 控制器控制超过整个外部数据存储器全部地址范围内的数据传输，再加上多数 SFR 寄存器映射到了 DMA 存储器空间，因此这些灵活的 DMA 通道大大减轻了 CPU 的负担。例如，从存储器传送数据到 USART，按照定下来的周期在 ADC 和存储器之间传送数据等。使用 DMA 可以保持 CPU 在不需要唤醒的低功耗模式下，能进行与外部设备单元之间的数据传送，从而降低了整个系统的功耗。

DMA 控制器的主要特性如下。

(1) 5 个独立的 DMA 通道。

(2) 3 个可以配置的 DMA 通道优先级。

(3) 32 个可以配置的传送触发事件。

(4) 源地址和目标地址独立控制。

(5) 3 种传送模式：单独传送、数据块传送和重复传送。

(6) 支持传输数据的长度域，设置可变的传输长度。

(7) 既可以在字(Word-Size)模式下运行，又可在字节(Byte-Size)模式下运行。

3. 定位器

CC2530 主要包括 4 个定时器：一般的 16 位(T1)定时器，16 位 MAC 定时器(T2)和两个 8 位(T3 和 T4)定时器。它们支持典型的定时/计数功能，例如，测量时间间隔、对外部事件计数、产生周期性中断请求、输入捕捉、比较输出和 PWM 功能。此外，还有看门狗定时器和睡眠定时器。

16 位 MAC 定时器(T2)主要用来提供用于 IEEE 802.15.4 CSMA-CA 的算法定时和在 IEEE 802.15.4 MAC 层上的一般计时。

MAC 定时器的主要特征如下。

(1) 16 位正计数，提供符号(Symbol)周期 16μs，帧(Frame)周期 320μs。

(2) 周期可调，精度为 31.25ns。

(3) 8 位计数器比较功能。

(4) 20 位溢出计数。

(5) 20 位溢出计数比较功能。

(6) 帧开始定界符的捕获功能。

(7) 定时器的开始、停止与外部 32.768kHz 时钟同步，而且由睡眠时钟保持。

(8) 中断由比较和溢出产生。

(9) DMA 的触发能力。

4. AES 加密协处理器

CC2530 数据加密是由支持高级加密标准的专用协处理器 AES 完成的。AES 协处理器的加密/解密操作，极大地减轻了 CC2530 内置 CPU 的负担。

AES 协处理器的主要特性如下。

(1) 支持 IEEE 802.15.4 全部安全机制。

(2) ECB(电子编码加密)、CBC(密码防护链)、CFB(密码反馈)、OFB(输出反馈加密)、CTR(计数模式加密)和 CBC-MAC(密码防护链消息验证代码)模式。

(3) 硬件支持 CCM(CTR+CBC-MAC)模式。

(4) 28 位密钥和初始化向量(Ⅳ)/当前时间(Nonce)。

(5) DMA 传送触发能力。

5. ADC(模数转换)

在工程应用中,可能需要采集温度、湿度、CO_2、PM2.5、光照等环境数据。直接采集它们得到的是模拟数据,此时需要使用 ADC 转换为数字信号量,以便于统计分析。CC2530 ADC 支持 14 位模拟数字转换,且具有多达 12 个有效位的 ENOB。它包括一个模拟多路转换器,该转换器具有多达 8 个各自可配置的通道,以及一个参考电压发生器。转换结果可通过 DMA 写入存储器。

ADC 的主要特性如下。

(1) 可选的采样率,能设置有效的分辨率(7～12 位)。

(2) 8 个各自独立的输入通道,可接收单端或差分信号。

(3) 参考电压可选为内部单端、外部单端、外部差分或 AVDD5。

(4) 可以产生中断请求。

(5) 转换结束时的 DMA 触发。

(6) 温度传感器输入。

(7) 电池测量功能,当监视的电池电量较低时,可在应用中设置提醒用户更换电池的功能。

6. USB 控制器

USB 控制器监控 USB 的相关活动,并处理数据包的传输工作。

固件的责任是适当响应 USB 中断和上传(下载)数据包到(从)FIFO 终端。同时,固件必须能够正确地响应所有来自 USB 主机的标准请求,并能根据 PC 上的协议工作。

USB 控制器的功能如下。

(1) 全速操作(最高 12Mb/s)。

(2) 5 个端口(除了端口 0)可以用作 IN、OUT 或 IN/OUT,也可以配置为批量/中断或同步。

(3) 可以使用 1KB SRAM 或 FIFO 存储 USB 数据包。

(4) 端口支持的数据包大小范围:8～512B。

(5) 支持 USB 数据包的双缓冲。

7. USART

USART0 和 USART1 是串行通信接口,它们能够分别运行于异步 UART 模式或者同步 SPI 模式。两个 USART 具有同样的功能,可在单独的 I/O 引脚设置。

▶ 5.2.3 CC2530 开发环境

我们在进行实验或开发的时候,必然会选择合适的硬件和软件,只有两者相互配合,才能发挥硬件的优势。这里选择 TI 公司的 ZigBee 解决方案,片上系统 CC2530 芯片,协议栈 ZStack。ZStack 栈默认的开发环境是 IAR 开发工具。TI 公司提供了许多协议栈配套工具,可选择 SmartRF Packet Sniffer 抓包工具。

ZigBee 技术应用开发环境主要包括硬件环境和软件环境。

1. 硬件环境

硬件环境主要包括一套 ZigBee 开发套件和一台 PC 设备。下面是搭建 ZigBee 技术开发环境所需要的软硬件清单,部分模块是可以选择安装的。

(1) CC2530/CC2531 ZigBee 开发套件。

① ZigBee 开发底板(必需)。

② ZigBee 最小模块(必需)。

③ 仿真器(也称为下载器、烧写器,必需)。

④ 若干串口线(或 USB 转串口线,部分实验需要)。

⑤ 若干温度、CO_2、光敏传感器(个别实验需要)。

(2) 开发主机要求。

① CPU:对于 CC2530EB 版本,CPU 的主频大于 1GHz。

② 内存:512MB 以上。

③ 硬盘:40GB 以上。

④ 串口接口:两个或更多(一般利用 USB 转串口线,将 USB 接口代替串口)。

⑤ USB 接口:两个或更多。

2. 软件环境

软件开发环境主要包括 IAR 开发工具和 ZStack 协议栈,具体如下所述。

① Windows 2000 及更高版本(Service Pack2 或更高版本)(必需)。

② EW8051-EV-730B(或更高版本)开发环境(必需)。

③ Z-Stack-CC2530-2.3.1-1.4.0(或更高版本)协议栈(必需)。

④ 串口调试助手(部分实验需要)。

⑤ FLASH Programer(可选择安装)。

⑥ Packet Sniffer(可选择安装)。

⑦ Sensor Monitor(可选择安装)。

5.3 无线传感器节点设计

随机布设的 ZigBee 无线传感器网络具有规模大、节点数量众多、无人值守的特点,为其开发设计带来了成本、技术等方面的挑战,对相关的硬件和软件开发、网络系统设计提出了不同于传统网络设计的要求。传统网络与 ZigBee 无线传感器网络的不同之处如表 5-3 所示。

表 5-3 传统网络与 ZigBee 无线传感器网络的不同之处

传 统 网 络	ZigBee 无线传感器网络
通用设计,服务于多个应用	单一设计,服务于特定应用
主要关注网络性能和延迟	功耗是主要设计关注点
器件和网络工作于可控的温和环境中	常布设于存在苛刻条件的环境中
通常容易维护与维修	与节点物理接触,很难甚至不可维修
组件故障通过维修解决	网络设计需要预计存在的组件故障,增加其冗余度
轻松获得全局网络信息和实现集中式管理	决策由本地节点完成,不支持集中式管理

ZigBee 无线传感器网络的设计主要分为节点硬件设计和节点软件设计。节点硬件设计局限于能量、通信、计算和存储,满足应用服务,追求设计尺寸小、价格低廉、更高效等目标。

ZigBee 无线传感器网络的硬件设计主要分为传感节点、网关节点和汇聚节点三种设备的设计。传感节点完成对周围环境对象的感知并进行适当处理,将有用的信息发送到目标节点。网关节点主要通过多种接入网络的方式,如以太网、Wi-Fi、移动公网等与外界进行数据交互。

汇聚节点同时将终端和网关的控制信息传送到相应的传感节点，具有承上启下的功能。

▶ 5.3.1 传感节点的设计

传感节点的设计应满足尺寸小、价格低廉、更高效等目标，为所需的传感器提供适当的接口，并提供所需的计算和存储资源，以及足够的通信功能。传感节点主要包括感知单元、控制单元、无线收发单元和电源管理单元4部分。

1. 感知单元

感知单元负责物理信号的提取。信号采集单元包括信号调理电路和模数转换模块。传感器输出的模拟信号需经信号调理才能符合模数转换要求。常见的信号调理方式有抗混叠滤波、降噪、放大、隔离、差分信号变单端信号等。信号调理的结果直接关系到信号的信噪比，影响信号的特征。模数转换模块的功能是把模拟信号转换为控制单元可接收的数字信号。近年来，随着MEMS(微机电系统)技术的发展，出现了集成信号调理电路和模数转换模块的数字传感器。这种数字传感器只需通过相应的数字接口即可实现与控制单元的通信，减小了节点的尺寸，降低了设计的复杂度。

2. 控制单元

控制单元将其他单元及外部接口连接在一起，处理有关感知、通信和自组织的指令。节点的任务调度、设备管理、功能协调、数据融合、特征提取、数据存储和能耗管理等都是在控制单元的支持下完成的。控制单元包括控制器件、非易失性存储器(通常是控制器件的片内Flash)、随机存储器、内部时钟等。大部分控制器件集成了非易失性存储器、随机存储器、内部时钟等，所以控制器件的选择应考虑以下因素。

(1) 功耗。

由于传感节点采用电池供电，因此控制器件满负荷工作的功耗应尽可能低。传感节点大部分时间处于休眠或待机状态。控制器件的休眠或待机状态功耗要低。

(2) 成本。

控制器件的成本在整个传感节点中占了很大的一部分，而ZigBee无线传感器网络需要成百上千的传感节点，控制器件的成本要尽可能低。

(3) 运行速度。

运行速度快的控制器件响应快、实时性强。但运行速度越快，功耗就越高，应权衡速度与功耗。

(4) 数据处理能力。

从功耗角度看，对于相同的数据量，对其进行数据处理所消耗的能量要远远小于对其进行无线传输所消耗的能量。

(5) 集成度。

在控制器件中集成有模数转换器、定时器、计数器、看门狗等模块的情况下，可以减少外围电路，降低成本。

(6) 存储空间。

代码的保存、运行和数据的存储都需要一定的存储空间，控制器件集成有存储空间，可以减少外围器件。存储空间的大小直接影响控制器件的性能。

(7) 通信接口和I/O口。

控制单元与其他模块的通信都是通过通信接口或通用的I/O口进行的。

(8) 中断响应。

控制器件能够在低功耗状态下进行快速中断响应,以降低网络延迟。

目前,常见的控制器件有微处理器(MCU)、数字信号处理器(DSP)、专用集成电路(ASIC)和可编程逻辑器件(FPGA)。

在进行复杂信号处理时,为了满足小波变换、快速傅里叶变换、神经网络算法、双谱分析等复杂时频运算对计算能力和存储能力的需求,控制器件宜选择数字信号处理器或可编程逻辑器件作为算法平台。

单纯考虑功耗方面的因素,宜选择微处理器作为网络控制平台。

对于功耗有特殊要求及大量节点的应用,节点需求量达到百万以上,控制器件宜选用专用集成电路。专用集成电路属于专用定制的控制器件,能够根据特定需求将功耗降到最低,并能减小电路板尺寸,但其后续扩展性较差。

在信号处理算法不复杂的情况下,控制器件宜选用微处理器。微处理器具有体积小、存储容量小、通信接口简单、功耗低、功能简单的特点。

大部分传感节点的控制器件采用的是微处理器。目前市场上主流的微处理器有 8 位、16 位和 32 位。微处理器的主流厂商有 Microchip(Atmel)、TI、瑞萨电子、恩智浦半导体、英飞凌、ADI 等。

在现有的传感节点中,应用比较多的 8 位微处理器是 Microchip 公司的 AVR 系列单片机。AVR 系列单片机采用哈佛结构,具有预取指令功能,能实现流水作业;采用超功能 RISC(精简指令集),具有 32 个通用工作寄存器;采用不可破解的锁位(Lock Bit)技术,增强代码保密性;有多个固定中断向量入口地址,可快速响应中断;片内集成多种频率的 RC 振荡器,具有上电自动复位、看门狗、启动延时等功能,外围电路更加简单,系统更加稳定可靠;具有多种省电休眠模式,且可宽电压运行,工作电压范围为 2.7～5V,抗干扰能力强;接口丰富,集成有模数转换器,差分信号模数转换器、串行接口等。上述优点使早期的传感节点大部分采用 AVR 系列单片机。

16 位微处理器与 8 位微处理器相比,字长更宽,计算能力更强。在传感节点中应用比较多的 16 位微处理器是 TI 公司的 MSP430 系列单片机。MSP430 系列单片机采用精简指令集结构,具有丰富的寻址方式、简洁的 27 条内核指令、硬件乘法器、高效的查表指令、大量的寄存器;片内资源丰富,包括看门狗、I2C、模数转换器、定时器、DMA(直接存储器访问)、UART(通用异步收发器)、SPI(串行外设接口)等;其 DMA 功能不仅显著增加了外设的数据吞吐能力,还大幅降低了系统功耗;有丰富的中断资源,当系统处于省电的低功耗状态时,电流消耗在微安级,中断唤醒时间小于 6μs;在降低芯片的电源电压和灵活可控的运行时钟方面都有其独到之处,从而实现低功耗;大部分产品都能自动工作与关闭,最大限度地减少了内核处于工作模式的时间。

对于视频、图像等高性能应用,传感节点的控制器件需采用 32 位微处理器。在 32 位嵌入式微处理器市场中,ARM 公司的 ARM 处理器占据了很大的市场份额。Cortex-M3 是 ARM 公司生产的低功耗、低成本和高性能的 32 位微处理器内核。它采用了 ARMv7-M3 架构,包括所有的 16 位 Thumb 指令集和 32 位 Thumb2 指令集架构,但不能执行 ARM 指令集。相比于 ARM 公司的 ARM7TDMI 架构,Cortex-M3 具有更小的基础内核,价格更低,速度更快。Cortex-M3 集成了睡眠模式和可选的八区域存储器保护单元,集成了中断控制器,提供基本的 32 个物理中断,具有低延迟性。目前,意法半导体、恩智浦半导体、TI、Microchip 等公司

已经开发出多款 Cortex-M3 内核的微处理器。虽然 Cortex-M3 已经取得了很大的进步，其计算能力胜于 AVR 系列单片机和 MSP430 系列单片机，但其功耗远远大于它们。

AVR 系列单片机、MSP430 系列单片机、Cortex-M3 系列单片机各有各的特点，在实际应用中需要根据不同的应用要求选用合适的控制器件。

3. 无线收发单元

传感节点之间通过无线收发单元实现互联，组成自组织传感器网络。传感节点的无线收发单元主要由无线窄带通信芯片和与其配套的滤波电路等外围电路组成。

根据所采用的通信频率的不同，目前市场上的无线窄带通信芯片可以分为 2.4GHz 和低于 1GHz 两种。2.4GHz 无线通信芯片的绕射能力较差，通信距离短，但其可靠性高，不容易受干扰，抗多径衰落能力强；低于 1GHz 无线通信芯片的绕射能力强，通信距离长，但其可靠性差，易受其他设备干扰，安全系数较低。2.4GHz 频段是国际通用的免费频段，也称为 ISM 频段，带宽为 83.5 MHz，可供多个不同通信系统的多个不同信道共同使用。

各芯片厂商根据市场需求推出了多款无线窄带通信芯片，常见的如表 5-4 所示。

表 5-4 常见无线窄带通信芯片

型号	频率范围/GHz	最大发送速率/kb·s^{-1}	接收电流/mA	接收灵敏度/dBm	最大发射功率/dBm	调制方式
CC2530	2.4	250	15	−97	4.5	DSSS-OQPSK
CC2533	2.4	250	14	−97	4.5	DSSS-OQPSK
ADF7241	2.4	250	19	−95	4.5	DSSS-OQPSK
MC13191	2.4	250	37	−91	4.5	DSSS-OQPSK

其中，TI 公司的 CC2533 是无线窄带通信芯片的典型代表。CC2533 是一款针对远程应用全面优化的 IEEE 802.15.4 片上系统，建立在 CC2530 的基础上。CC2533 的最大发射功率可达到 4.5dBm，典型接收灵敏度可达 −97dBm。CC2533 集成了硬件 AES(高级加密标准)，可产生 128 位的密钥，从而保证了信息安全。CC2533 包括 1μA 睡眠模式的 4 种灵活电源模式，可实现最低的电流消耗。CC2533 采用 DSSS(直接序列扩频)调制技术，具有抗干扰性好、抗多径衰落能力强、环境噪声要求低和高度可靠的保密安全性等特点，适合于复杂环境条件下的应用。

4. 电源管理单元

在传感节点中，电源管理单元是一个关键的系统组件，体现在两方面：第一，它是存储能量，并为其他单元提供所需电压的稳压器件；第二，它能从外部环境中获取额外的能量。存储能量主要是通过电池实现的，还可以通过燃料电池、超级电容实现。在实际应用中，应根据环境及需求决定采用哪种储存能量的设备。

▶ 5.3.2 网关节点和汇聚节点的设计

网关节点和汇聚节点具备信息聚合、处理、选择、分发，以及子网网络管理等功能。

传感节点对其部署的区域进行监控，获取感知信息。网关节点和汇聚节点对其控制区域内的传感节点，实现任务调度、数据融合、网络维护等功能。传感节点获取的信息数据经过汇聚节点融合、处理及打包后，由网关节点聚合，根据不同的业务需求和接入网络环境，经由无线局域网接入点、互联网接入点、2GHz 公网接入点、3GHz 公网接入点、4GHz 公网接入点、中高速网络等多类型的异构网络，最终将信息数据传送给终端用户，实现针对 ZigBee 无线传感器

网络的远程监控。同样，终端用户也可以通过无线局域网接入点、互联网接入点、2GHz 公网接入点、3GHz 公网接入点、4GHz 公网接入点、中高速网络等接入网关节点，网关节点连接到相应的汇聚节点，再通过汇聚节点对传感节点进行数据查询、任务派发、业务扩展等操作，最终将 ZigBee 无线传感器网络与终端用户有机联系在一起。

在 ZigBee 无线传感器网络中，汇聚节点主要负责传感器网与外网的连接，可将其看作网关节点，因此下面主要介绍网关节点的设计。

1. 控制单元

控制单元主要考虑其计算能力、存储能力和接口。8 位和 16 位微处理器很难满足，一般选用高性能的 32 位微处理器作为网关节点的控制单元。

网关节点的功能如下。

(1) 网关节点具备信息融合、处理和分发功能。

(2) 网关节点能够同时支持 ZigBee 无线传感器网络协议栈和与终端交互的协议(如以太网协议、无线局域网协议等)。

(3) 网关节点能够维护区域 ZigBee 无线传感器网络，防止网络堵塞的发生。

(4) 网关节点能够处理监测区域内所有传感节点的突发数据传输，具有较高的数据吞吐量。

(5) 网关节点具有保存本地数据的功能，以免外部网络中断而丢失数据。

2. 无线收发单元

网关节点通信分为对上和对下两种。对上的无线收发单元主要面向 2GHz 公网、3GHz 公网、4GHz 公网、无线局域网、互联网、中高速网络等，满足接入各类骨干网络的需求，适应传感器网络的泛在特征。对下的无线收发单元主要用于与无线传感节点或汇聚节点通信。

3. 电源管理

网关节点的功耗远大于传感节点，应采用有线电源供电，其电源管理主要是为网关节点的各个器件提供合适的电压，而不考虑低功耗管理。

5.4 基于 ZigBee 的数据采集

▶ 5.4.1 节点应用选型

ZigBee 无线传感器网络节点大规模产业化，节点的性能受成本影响很大。针对节点应用，提出了“共性平台＋应用子集”的方案。

从系统层面的需求来看，传感节点存在的几类需求如下。

(1) 从目标探测方式来看，存在主动式探测和被动式探测两种需求。这两类设备在感知模式、用于对环境或指令等信息进行反馈的执行器结构和功能、感知信息在网络中的传输模式和流量特征、信息预处理(主动式探测设备要求可以针对特定任务进行有效的针对性处理，被动式探测设备必须支持不同环境下的感知信息的预处理，并尽量减小误报率和漏报率)，以及节点状态等方面存在较大差异，必须针对各个模块进行专门的模块级设计和实现。

(2) 从感知参数来看，存在单参感知和多参感知两种需求。这两类设备在感知量(单传感节点、多传感节点或同一类的不同参量导致了采样方式和预处理方式的不同)和信息处理(多参量需要基于物理相关性进行模态融合等)上有显著差别，进而导致设备结构存在较大差异，必须针对各个模块进行专门的模块级设计和实现。

(3) 从目标参数类型来看，存在标量感知信息和矢量感知信息两种需求。两类设备由于

传感信息类型的不同而在时间相关性、空间相关性、目标信息相关性、模态相关性、系统要求等方面存在差异，从而导致设备软/硬件资源配置、功能模块设计等方面会有较大差别，需要研制专门的设备种类。例如，矢量感知信息对同步、定位等存在较高要求。

(4) 从节点对感知信息的协同处理能力来看，对地震波、声波及大部分混合传感器信息需要本地高协同处理能力，以减少网络传输能耗损失，而对于一般家居控制等则仅需要简单的处理能力即可。针对高协同处理能力设备和低协同处理能力设备的开发在人力、技术、成本等因素上差别较大，有必要进行针对性研发。

(5) 从网络通信能力来看，对低功耗无线传输设备存在近距离(100m 以内)和中距离(1000m 以内)两类需求。近距离设备满足室内和室外传感器密集布设需要，而中距离设备将为野外传感节点的使用带来极大的便利。

▶ 5.4.2　无线节点

工业互联网平台采用物联网技术构建了异构传感网络，支持 ZigBee、Wi-Fi、LoRa、NB-IoT 等无线节点传感设备的数据接入和应用。所有传感器的数据通过智能网关接入数据中台的物联网云平台。基于云平台技术构建的传感网主要由无线节点、传感器、ZigBee 协调器等硬件构成。

1. 硬件框架

工业互联网经典型无线节点采用 CC2530 作为 MCU 主控，板载信号指示灯；电源电池、网络、数据，板载集成锂电池接口，集成电源管理芯片；板载 USB 串口，T 仿真器接口，ARM 仿真器接口；集成两路 RJ45 工业接口，提供主芯片 P0_0～P0_7 输出，硬件包含 I/O、ADC3.3V、ADCSV、UART、RS485 等功能。工业物联网配备了 5 个无线节点对应 8 个传感器，分别是空气质量无线节点(A)、电子横幅无线节点(A)、光照度(A)、人体红外(B)无线节点、噪声(A)、燃气(B)无线节点、大气压力(A)及火焰(B)无线节点。

2. 软件框架

远程无线节点就传输过程分为如下几部分：传感器、无线节点、ZigBee 协调器、云数据中心和客户端(Web)。通信流程图如图 5-6 所示。

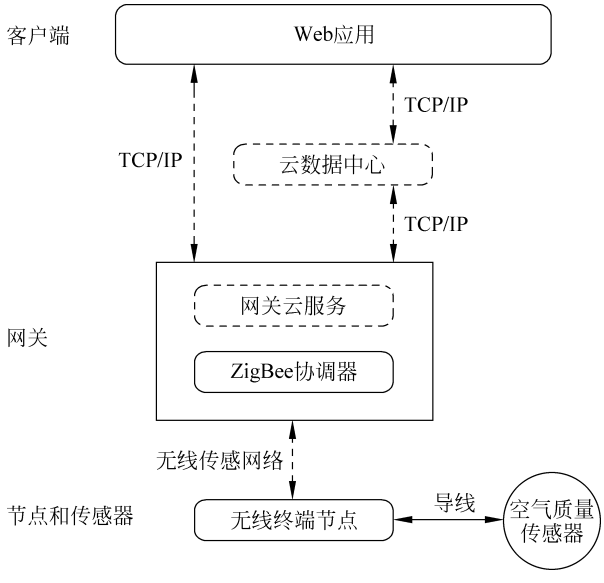

图 5-6　通信流程图

具体通信描述，如下所述。

(1) 无线终端/路由节点(以下简称无线节点)通过导线读取传感器的数据，无线节点通过ZigBee 协调器与网关上集成的汇集节点进行组网通信。

(2) 网关上的云服务，将集成在网关上的汇集节点收到的数据，推送到本地局域网和远程云数据中心。

(3) 客户端(Web)通过调用云数据接口，实现实时数据采集等功能，或对传感器进行控制。

▶ 5.4.3 节点软件设计

1. 空气质量传感器节点

空气质量传感器作为 TTL 串口采集类物联网传感器，能够定时采集当前环境下的空气质量数据并主动发送至节点。运用过程中通过对空气质量传感器的数据采集监控，实现智能地监测当前环境空气中 CO_2 浓度、温湿度以及 PM2.5 等数据，通过实时地将空气质量信息推送到云数据中心，在 Web 端获得这些数据后，用户就能够随时随地远程获取当前环境下的空气质量信息。这里使用商业空气质量传感器，通信接口为 TTL 串口，型号为 KHZL00IXTTL。空气质量传感器通过 RJ45 端口连接到 LiteB 节点的 A 端子连接。

空气质量传感器采用 TTL 串口通信协议，串口通信配置为波特率 9600、8 位数据位、无校验位、无硬件数据流控制和 1 位停止位。

本传感器的数据采集方法：利用 TTL 串口向该传感器发送读取数据命令，传感器收到命令后会返回 CO_2 浓度、温湿度以及 PM2.5 浓度值等。发送报文格式如表 5-5 所示。

表 5-5 空气质量传感器发送报文格式

发送内容	字节数	发送数据	备注
起始符	1	11H	数组第一个元素
长度	1	XXH	帧字节长度
命令号	1	XXH	指令号
数据	XX	XXXX…H	读取或写入的数据
校验和	1	XXH	数据累加和

从机返回的报文格式如表 5-6 所示。

表 5-6 空气质量传感器返回报文格式

接收内容	字节数	发送数据	备注
起始符	1	XXH	数组第一个元素
长度	1	XXH	帧字节长度
命令号	1	01H	主机读取测量结果时的指令号
CO_2 浓度	2	XXXXH	存储测得 CO_2 浓度的字节
甲醛浓度	2	XXXXH	存储测得甲醛浓度的字节
温度	2	XXXXH	存储测得温度的字节
湿度	2	XXXXH	存储测得湿度的字节
PM2.5 浓度	2	XXXXH	存储测得 PM2.5 浓度的字节
校验和	1	XXH	数据累加和

各项数据计算公式，如下所述。

(1) CO_2 浓度计算公式：CO_2 浓度＝十六进制数值(XXXX)→十进制数值(X)。

(2) 甲醛浓度计算公式：甲醛浓度＝十六进制数值(XXXX)→十进制数值(X)。

(3) 温度计算公式：温度＝十六进制数值(XXXX)×0.1→十进制数值(X)。

(4) 湿度计算公式：湿度＝(十六进制数值(XXXX)－500)×0.1→十进制数值(X)。

(5) PM2.5 浓度计算公式：PM2.5 浓度＝十六进制数值(XXXX)→十进制数值(X)。

2. 光照度无线节点

光照度传感器作为重要的 I2C 接口采集类物联网传感器，能够定时采集当前环境下光照强度数据并主动发送到节点。运用过程中通过对光照度传感器的数据采集控制，实现智能监测当前环境的光照强度，通过实时将光照强度信息推送到云数据中心，在 Web 端获得这些数据后，用户能实现随时随地远程获取光照强度等环境信息。这里使用商用光照度传感器，通信接口为 I2C，型号为 GZ001xIIC。光照度传感器通过 RJ45 端口连接到 LiteB 节点的 A 端子连接。

光照度传感器采用 I2C 通信协议，本传感器的数据采集方法：利用 I2C 总线向该传感器发送读取数据命令，传感器收到命令后会返回光照强度值。发送的报文格式如表 5-7 所示。

表 5-7 光照度传感器发送的报文格式

发送内容	字节数	发送数据	备注
从机地址	1	XXH	从机地址
功能码	1	03H	读取寄存器
起始寄存器地址	2	0000H	该寄存器中保存设备的父类型和子类型
读取寄存器数量	2	0001H	读取 1 个字(2B)
CRC 校验	2	XXXXH	前面所有数据的 CRC 码

从机返回的报文格式如表 5-8 所示。

表 5-8 光照度传感器返回的报文格式

接收内容	字节数	发送数据	备注
从机地址	1	XXH	从机地址
功能码	1	03H	读取寄存器应答
返回字节长度	1	02H	返回 2B
返回数据	2	XXXXH	2B 的光强信息
CRC 校验	2	XXXXH	前面所有数据的 CRC 码

光照度计算公式：光强＝十六进制数值(XXXX)→十进制数值(X)。

3. 人体红外无线节点

人体红外探测器传感器作为电平采集类物联网传感器，能够定时采集室内人体红外信号变化的信息，并主动发送至节点。运用过程中通过对人体红外探测器传感器采集的人体红外信号变化信息监控，实现智能监测当前室内的人员移动信息，通过实时将室内的人体红外信号变化信息推送到云数据中心，在凭借 Web 端获得这些数据后，用户能够实现随时随地远程获取室内的防盗安全信息。这里使用商业人体红外探测器传感器，通信方式为 I/O 电平通信，型号为 ZY-RTHW001xIO。人体红外探测器传感器通过 RJ45 端口连接到 LiteB 节点的 B 端

子连接。

在实际的使用过程中，人体红外探测器传感器通信接口默认电平位高，当人体红外探测器传感器监测到红外光信号发生变化时，人体红外探测器传感器会发出报警，同时通信接口电平被拉低，通知 LiteB 节点发现有人经过。

4. 噪声无线节点

噪声传感器作为 RS485 串口类物联网传感器，能够定时采集环境中的噪声信息并且主动发送至节点。运用过程中通过对噪声传感器采集的城市噪声的信息监控，实现智能地监测城市环境的噪声信息，通过实时地将噪声信息推送到云数据中心，在凭借 Web 端获得这些数据后，用户能够实现随时随地远程获取噪声信息。

这里使用噪声传感器，通信方式为 RS485 串口通信，型号为 ZY001x485。噪声传感器通过 RJ45 端口连接到 LiteB 节点的 A 端子连接。

传感器采用 RS485 串口通信协议，串口通信配置为波特率 9600、8 位数据位、偶校验位、无硬件数据流控制、1 位停止位。

本传感器的数据采集方法：利用 RS485 串口总线向该传感器发送读取数据命令，传感器收到命令后会返回噪声值。

5. 燃气无线节点

燃气探测器传感器作为电平采集类物联网传感器，能够定时采集室内燃气信号变化的信息，并主动发送至节点。运用过程中通过对燃气探测器传感器的采集，实现智能监测当前室内燃气的报警，通过实时将室内的燃气信号变化信息推送到云数据中心，在凭借 Web 端获得这些数据后，用户能够实现随时随地远程获取室内的燃气安全信息。这里使用商业燃气探测器传感器，通信方式为 I/O 电平通信。噪声探测器传感器通过 RJ45 端口连接到 LiteB 节点的 B 端子连接。

在实际的使用工程中，燃气探测器传感器通信接口默认电平位高，当燃气探测器传感器监测到燃气时，燃气探测器传感器会发出报警，同时通信接口电平被拉低，通过 LiteB 节点上传云平台并且报警。

6. 大气压力无线节点

大气压力传感器作为 RS485 串口类物联网传感器，能够定时采集环境中噪声信息并且主动发送至节点。运用过程中通过对大气压力传感器采集的城市大气压力的信息监控，实现智能地监测环境的大气压力信息，通过实时地将大气压力信息推送到云数据中心，在凭借 Web 端获得这些数据后，用户能够实现随时随地远程获取大气压力信息。

这里使用大气压力传感器，通信方式为 RS485 串口通信。噪声传感器通过 RJ45 端口连接到 LiteB 节点的 A 端子连接。大气压力传感器采用 RS485 串口通信协议，串口通信配置为波特率 9600、8 位数据位、无校验位、无硬件数据流控制、1 位停止位。

本传感器的数据采集方法：利用 RS485 串口向该传感器发送读取数据命令，传感器收到命令后会返回大气压力值。

7. 火焰无线节点

火焰探测器传感器作为电平采集类物联网传感器，能够定时采集室内火焰信号变化的信息并主动发送至节点。运用过程中通过对火焰探测器传感器的采集，实现智能监测当前室内火焰，通过实时将室内是否有火焰的信号变化信息推送到云数据中心，在凭借 Web 端获得这些数据后，用户能够实现随时随地远程获取室内的火焰安全信息。

这里使用商业燃气探测器传感器，通信方式为 I/O 电平通信。火焰探测器传感器通过 RJ45 端口连接到 LiteB 节点的 B 端子连接。

在实际的使用工程中，燃气探测器传感器通信接口默认电平位高，当火焰探测器传感器监测到火焰时，燃气探测器传感器会发出报警，同时通信接口电平被拉低，通过 LiteB 节点上传到云平台并且报警。

▶ 5.4.4 功能实现

1. 空气质量传感数据

传感器硬件初始化函数，代码如下。

```
/*****************************************************************************
 * 名称: sensorlnit()
 * 功能: 传感器硬件初始化
 * 参数: 无
 * 返回:
 * 修改:
 * 注释:
 *****************************************************************************/
void sensorlnit(void)
{
    //延时 50ms
    MicroWait(50000);
    //初始化传感器代码
    node_uart_init();
    //开启 VOC 测量
    sensorControl(D1);
    //启动定时器，触发传感器上报数据事件 MY_REPORT_EVT
    Osal_start_timerEx(sapi_TaskID,MY_REPORT_EVT,(uint16)((osal_rand()%10)* 1000));
    //启动定时器，触发传感器监测事件 MY_CHECK_EVT
    Osal_start_timerEx(sapi_TaskID,MY_CHECK_EVT,100);
}
```

处理主动上报的数据，代码如下。

```
/*****************************************************************************
 * 名称: sensorUpdate()
 * 功能: 处理主动上报的数据
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 *****************************************************************************/
void sensorUpdate(void)
{
    char pData[16];
    char *p = pData;
    ZXBeeBegin();                              //云数据帧格式包头
    //根据 D0 的位状态判定需要主动上报的数值
    if((D0&0x01) == 0x01){                     //若上报允许，则在 pData 的数据包中添加数据
        updateA0();
        sprintf(p,"%d",A0);
        ZXBeeAdd("A0",p);
    }
    if((D0&0x02) == 0x02){                     //若上报允许，则在 pData 的数据包中添加数据
        updateA0();
        sprintf(p, "%d",A0);
```

```
            ZXBeeAdd("A1", p);
        }
        if((D0&0x04) == 0x04){                    //若上报允许,则在 pData 的数据包中添加数据
            updateA0();
            sprintf(p," % .1f",A2);
            ZXBeeAdd("A2", p);
        }
        if((D0&0x08) == 0x08){                    //若上报允许,则在 pData 的数据包中添加数据
            updateA0();
            sprintf(p, " % .1f", A3);
            ZXBeeAdd("A3", p);
        }
        if((D0&0x10) == 0x10){                    //若上报允许,则在 pData 的数据包中添加数据
            updateA0();
            sprintf(p," % d", A4);
            ZXBeeAdd("A4", p);
        }
        sprintf(p," % u",D1);                     //上报控制编码
        ZXBeeAdd("D1", p);
        p = ZXBeeEnd();                           //云数据帧格式包尾
        if(p != NULL) {
            ZXBeelnfSend(p.strlen(p));
            //将需要上传的数据进行打包操作,并通过 zb_SendDataRequest()发送到协调器
        }
    }
```

解析收到的控制命令函数,代码如下。

```
    /*****************************************************************************
     * 名称: ZXBeeUserProcess()
     * 功能: 解析收到的控制命令
     * 参数: ptag -- 控制命令名称
     *       pval -- 控制命令参数
     * 返回: ret -- pout 字符串长度
     * 修改:
     * 注释:
     *****************************************************************************
    int ZXBeeUserProcess(char * ptag, char * pval)
    {
        int val;
        int ret = 0;
        char pData[16];
        char * p = pData;

        //将字符串变量 pval 解析转换为整型变量赋值
        val = atoi(pval);
        //控制命令解析
        if (0 == strcmp("CD0", ptag)){            //对 D0 的位进行操作,CD0 表示位清零操作
           D0 & = ~val;
        }
        if(0 == strcmp("OD0", ptag) ){            //对 D0 的位进行操作,OD0 表示位置 1 操作
           D0 | = val;
        }
        if(0 == strcmp("D0", ptag)){              //查询上报使能编码
           if(0 == strcmp("?", pval)){
              ret = sprintf(p, " % u", D0);
              ZXBeeAdd("D0", p);
           }
        }
        if (0 == strcmp("CD1", ptag)){            //对 D1 的位进行操作,CD1 表示位清零操作
```

```
        D1& = ~val;
        sensorControl(D1);                    //处理执行命令
    }
    if (0 == strcmp("OD1", ptag)){            //对 D1 的位进行操作,OD1 表示位置 1 操作
        D1 | = val;
        sensorControl(D1);                    //处理执行命令
    }
    if (0 == strcmp("Dl", ptag)){             //查询执行器命令编码
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%u", D1);
            ZXBeeAdd("D1", p);
        }
    }
    if (0 == strcmp("A0", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%d",A0);
            ZXBeeAdd("A0", p)
        }
    }
    if (0 == strcmp("A1", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%d", A1);
            ZXBeeAdd("A1", p);
        }
    }
    if (0 == strcmp("A2", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%.1f", A2);
            ZXBeeAdd("A2", p);
        }
    }
    if (0 == strcmp("A3", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%.f", A3);
            ZXBeeAdd("A3", p);
        }
    }
    if (0 == strcmp("A4", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%d", A4);
            ZXBeeAdd("A4", p);
        }
    }
    if (0 == strcmp("V0", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%u", V0);       //上报时间间隔
            ZXBeeAdd("V0", p);
        }else{
            updateV0(pval);
        }
    }
    return ret;
}
```

串口通信回调参数,代码如下。

```
/*****************************************************************************
 * 名称: node_uart_callback()
 * 功能: 节点串口通信回调参数
 * 参数: port -- 输入参数,接收端口
 *       event -- 输入参数,接收事件
 * 返回: 无
 * 修改:
 * 注释:
 *****************************************************************************/
static void node_uart_callback(uint8 port ,uint8 event)
{
    #define RBUFSIZE 32
    (void)event;
    uint8 ch;
    static uint8 szBuf[RBUFSIZE];
    static uint8 rlen = 0;
    static uint8 data_len = 0;
    static uint8 flag_ = 0x00;
    static uint8 data_mode = 0xff;

    //接收通过串口传来的数据
    while (Hal_UART_RxBufLen(port))
    {
        HalUARTRead (port, &ch, 1);
        if((ch == 0x16) &&(flag_ == 0x00))
        {
            flag_ = 0x01;
        }
        else if(flag_ == 0x01){
            data_len = ch;
            flag_ = 0x02;
        }
        else if(flag_ == 0x02)
        {
            data_mode = ch;
            flag_ = 0x03;
        }
        else if((flag_ == 0x03)){
            szBuf[rlen++] = ch;
            if(rlen >= data_len)
        {
            flag_ = 0x00;
            switch (data_mode){
                case 0x01:
                A0 = szBuf[0] * 256 + szBuf[1];
                A1 = szBuf[2] * 256 + szBuf[3];
                A2 = (szBuf[4] * 256 + szBuf[5]) * 0.1;
                A3 = (szBuf[6] * 256 + szBuf[7] - 500) * 0.1;
                A4 = szBuf[8] * 256 + szBuf[9];
                break;
            case 0x03:
                break;
            case 0x0c:
                if(szBuf[0] == 0x01)
                    D1 = 0;
                else
                    D1 = 1;
                    break;
                    default:
```

```
                        rlen = 0;
                        break;
                    }
                    rlen = 0;
                }
            }
        }
}
```

2. 电子横幅无线节点

初始化电子屏控制端口,代码如下。

```
/*********************************************************************************
 * 名称: phs_init()
 * 功能: 初始化电子屏控制端口
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 *********************************************************************************/
void lattice_init(void)
{
    osal_nv_item_init(LATTICE_CONTENT_ID, ASCII_MAX_BUF, latticeBuf);
    osal_nv_read(LATTICE_CONTENT_ID, 0, ASCII_MAX_BUF, latticeBuf);
    latticeBuf[ASCII_MAX_BUF - 1] = 0;
    if(strlen(latticeBuf) == ASCII_MAX_BUF - 1) {
    latticeBuf[0] = 0;
    }
    node_uart_init();
}
```

```
/*********************************************************************************
 * 名称: node_uart__callback()
 * 功能: 节点串口通信回调函数
 * 参数: port -- 输入参数,接收端口
 *       event -- 输入参数,接收事件
 * 返回: 无
 * 修改:
 * 注释:
 *********************************************************************************/
static void node_uart_callback ( uint8 port, uint8 event )
{
    #define RBUFSIZE 128
    (void)event;                                    //此处暂不定义参数
    uint8 ch;
    //static uint8 szBuf[RBUFSIZE];
    //static uint8 len = 0;
    //接收通过串口传来的数据
    while (Hal_UART_RxBufLen(port)){                //获取数据
    HalUARTRead (port, &ch, 1);                     //提取数据赋予 ch
    }
}
```

RS485 串口初始化,代码如下。

```
/*********************************************************************************
 * 名称: node_uart_init()
 * 功能: RS485 串口初始化
 * 参数: 无
 * 返回: 无
```

```
 * 修改:
 * 注释:
 ********************************************************************************
static void node_uart_init(void)
{
    halUARTCfg_t _UartConfigure;

    //UART 配置信息
    _UartConfigure.configured = TRUE;                    //启用串口
    _UartConfigure.baudRate = HAL_UART_BR_9600;          //波特率设置为 9600
    _UartConfigure.flowControl = FALSE;                  //关闭硬件数据流控制
    _UartConfigure.rx.maxBufSize = 128;                  //最大接收数组 128
    _UartConfigure.tx.maxBufSize = 128;                  //最大发送数据 128
    _UartConfigure.flowControlThreshold = (128/2);
    _UartConfigure.idleTimeout = 6;
    _UartConfigure.intEnable = TRUE;                     //使能串口
    _UartConfigure.callBackFunc = uart_callback_func;    //配置数据提取回调函数

    //启动 UART 功能
    HalUARTOpen (HAL_UART_PORT_0, &_UartConfigure);
}
```

3. 光照度与人体红外节点传感器初始化

传感器硬件初始化,代码如下。

```
/********************************************************************************
 * 名称: sensorInit()
 * 功能: 传感器硬件初始化
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 ********************************************************************************
void sensorInit(void)
{
    bh1750_init();                                       //光照度传感器初始化

    //初始化人体红外传感器代码
    P0SEL &= ~0x10;
    P0DIR &= ~0x10;
    //P0_4 上拉
    P0INP &= ~(1 << 4);
    P2INP &= ~(1 << 5);
    //启动定时器,触发传感器上报数据事件 MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand() % 10) * 1000));
    //启动定时器,触发传感器监测事件 MY_CHECK_EVT
    osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, 100);
}
```

传感器监测,代码如下。

```
/********************************************************************************
 * 名称: sensorCheck()
 * 功能: 传感器监测
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 ********************************************************************************
void sensorCheck(void)
```

```
{
    static uint8 last_A1 = 0, count = 0;
    updateA1();
    if(last_A1 != A1)
    {
        count++;
        if(count == 3) {
            count = 0;
            sensorUpdate();
            last_A1 = A1;
        }
    } else {
        count = 0;
    }
}
```

处理主动上报的数据,代码如下。

```
/*********************************************************************************
 * 名称: sensorUpdate()
 * 功能: 处理主动上报的数据
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 *********************************************************************************/
void sensorUpdate(void)
{
    char pData[40];
    char *p = pData;
    ZXBeeBegin();                       //智云数据帧格式包头
    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){           //若温度上报允许,则在 pData 的数据包中添加温度数据
        updateA0();
        sprintf(p, "%.1f", A0);
        ZXBeeAdd("A0", p);
    }
    if ((D0 & 0x02) == 0x02){           //若温度上报允许,则在 pData 的数据包中添加温度数据
        updateA1();
        sprintf(p, "%u", A1);
        ZXBeeAdd("A1", p);
    }
    p = ZXBeeEnd();                     //智云数据帧格式包尾
    if (p != NULL) {
        ZXBeeInfSend(p, strlen(p));
        //将需要上传的数据进行打包操作,并通过 zb_SendDataRequest()发送到协调器
    }
}
```

解析收到的控制命令,代码如下。

```
/*********************************************************************************
 * 名称: ZXBeeUserProcess()
 * 功能: 解析收到的控制命令
 * 参数: ptag -- 控制命令名称
 *       pval -- 控制命令参数
 * 返回: ret -- pout 字符串长度
 * 修改:
 * 注释:
 *********************************************************************************/
int ZXBeeUserProcess(char *ptag, char *pval)
```

```
{
    int val;
    int ret = 0;
    char pData[40];
    char *p = pData;
    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);
    //控制命令解析
    if (0 == strcmp("CD0", ptag)){                    //对 D0 的位进行操作,CD0 表示位清零操作
       D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)){                    //对 D0 的位进行操作,OD0 表示位置 1 操作
       D0 |= val;
    }
    if (0 == strcmp("D0", ptag)){                     //查询上报使能编码
       if (0 == strcmp("?", pval)){
          ret = sprintf(p, "%u", D0);
          ZXBeeAdd("D0", p);
       }
    }
    if (0 == strcmp("CD1", ptag)){                    //对 D1 的位进行操作,CD1 表示位清零操作
       D1 &= ~val;
       sensorControl(D1);                             //处理执行命令
    }
    if (0 == strcmp("OD1", ptag)){                    //对 D1 的位进行操作,OD1 表示位置 1 操作
       D1 |= val;
       sensorControl(D1);                             //处理执行命令
    }
    if (0 == strcmp("D1", ptag)){                     //查询执行器命令编码
       if (0 == strcmp("?", pval)){
          ret = sprintf(p, "%u", D1);
          ZXBeeAdd("D1", p);
       }
    }
    if (0 == strcmp("A0", ptag)){
       if (0 == strcmp("?", pval)){
          updateA0();
          ret = sprintf(p, "%.1f", A0);
          ZXBeeAdd("A0", p);
       }
    }
    if (0 == strcmp("A1", ptag)){
       if (0 == strcmp("?", pval)){
          updateA1();
          ret = sprintf(p, "%u", A1);
          ZXBeeAdd("A1", p);
       }
    }
    if (0 == strcmp("V0", ptag)){
       if (0 == strcmp("?", pval)){
          ret = sprintf(p, "%u", V0);                 //上报时间间隔
          ZXBeeAdd("V0", p);
       }else{
          updateV0(pval);
       }
    }
    return ret;
}
```

4. 噪声燃气无线节点

更新 A1 的值,代码如下。

```
/*****************************************************************************
 * 名称: updateA1()
 * 功能: 更新 A1 的值
 * 参数:
 * 返回:
 * 修改:
 * 注释:
 *****************************************************************************/
void updateA1(void)
{
    if (D_GAS_BET){                                    //判断管脚的电平状态值
        A1 = 1;                                        //检测为高电平时没有检测到燃气 A1 = 1
    }else{
        A1 = 0;                                        //否则检测到燃气 A1 = 0
    }
}
```

传感器硬件初始化,代码如下。

```
/*****************************************************************************
 * 名称: sensorInit()
 * 功能: 传感器硬件初始化
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 *****************************************************************************/
void sensorInit(void)
{
    //初始化传感器代码
    Noise_init();

    //初始化燃气传感器代码
    P0SEL &= ~0x10;
    P0DIR &= ~0x10;

    //P0_4 上拉
    P0INP &= ~(1 << 4);
    P2INP &= ~(1 << 5);

    //初始化继电器代码
    P0SEL &= ~0xC0;                                    //配置管脚为通用 IO 模式
    P0DIR |= 0xC0;                                     //配置控制管脚为输入模式
    sensorControl(D1);                                 //初始化继电器状态

    //启动定时器,触发传感器上报数据事件 MY_REPORT_EVT
    osal_start_timerEx(sapi_TaskID, MY_REPORT_EVT, (uint16)((osal_rand() % 10) * 1000));
    //启动定时器,触发传感器监测事件 MY_CHECK_EVT
    osal_start_timerEx(sapi_TaskID, MY_CHECK_EVT, 100);
}
```

处理主动上报的数据,代码如下。

```
/*********************************************************************************
 * 名称: sensorUpdate()
 * 功能: 处理主动上报的数据
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 *********************************************************************************/
void sensorUpdate(void)
{
    char pData[16];
    char *p = pData;
    ZXBeeBegin();                          //云数据帧格式包头
    //根据 D0 的位状态判定需要主动上报的数值
    if ((D0 & 0x01) == 0x01){              //若温度上报允许,则在 pData 的数据包中添加温度数据
        updateA0();
        sprintf(p, "%.1f", A0);
        ZXBeeAdd("A0", p);
    }
    if ((D0 & 0x02) == 0x02){
        updateA1();
        sprintf(p, "%u", A1);
        ZXBeeAdd("A1", p);
    }
    p = ZXBeeEnd();                        //云数据帧格式包尾
    if (p != NULL) {
        ZXBeeInfSend(p, strlen(p));
        //将需要上传的数据进行打包操作,并通过 zb_SendDataRequest()发送到协调器
    }
}
```

解析收到的控制命令,代码如下。

```
/*********************************************************************************
 * 名称: ZXBeeUserProcess()
 * 功能: 解析收到的控制命令
 * 参数: ptag -- 控制命令名称
 *       pval -- 控制命令参数
 * 返回: ret -- pout 字符串长度
 * 修改:
 * 注释:
 *********************************************************************************/
int ZXBeeUserProcess(char *ptag, char *pval)
{
    int val;
    int ret = 0;
    char pData[16];
    char *p = pData;
    //将字符串变量 pval 解析转换为整型变量赋值
    val = atoi(pval);
    //控制命令解析
    if (0 == strcmp("CD0", ptag)){                  //对 D0 的位进行操作,CD0 表示位清零操作
        D0 &= ~val;
    }
    if (0 == strcmp("OD0", ptag)){                  //对 D0 的位进行操作,OD0 表示位置 1 操作
        D0 |= val;
    }
    if (0 == strcmp("D0", ptag)){                   //查询上报使能编码
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%u", D0);
```

```
            ZXBeeAdd("D0", p);
        }
    }
    if (0 == strcmp("A0", ptag)){
        if (0 == strcmp("?", pval)){
            updateA0();
            ret = sprintf(p, "%.1f", A0);
            ZXBeeAdd("A0", p);
        }
    }
    if (0 == strcmp("A1", ptag)){
        if (0 == strcmp("?", pval)){
            updateA1();
            ret = sprintf(p, "%u", A1);
            ZXBeeAdd("A1", p);
        }
    }
    if (0 == strcmp("V0", ptag)){
        if (0 == strcmp("?", pval)){
            ret = sprintf(p, "%u", V0);            //上报时间间隔
            ZXBeeAdd("V0", p);
        }else{
            updateV0(pval);
        }
    }
    return ret;
    }
```

5. 大气压力与火焰无线节点

初始化传感器控制端口,代码如下。

```
/*****************************************************************************
 * 名称: phs_init()
 * 功能: 初始化传感器控制端口
 * 参数: 无
 * 返回: 无
 * 修改:
 * 注释:
 *****************************************************************************
void dqy_init(void)
{
    //初始化传感器代码
    node_uart_init();
    dqy_update();
    }
/*****************************************************************************
     * 名称: phs_update()
     * 功能: 更新一次数据,保存到全局静态变量(dqy_VAL)中
     * 参数: 无
     * 返回: 无
     * 修改:
     * 注释:
 *****************************************************************************/
    void dqy_update(void)
    {
    uart_485_write(f_szGetdqy, sizeof(f_szGetdqy));
    }
```

节点串口通信回调函数,代码如下。

```
/*********************************************************************************
 * 名称: node_uart__callback()
 * 功能: 节点串口通信回调函数
 * 参数: port -- 输入参数,接收端口
 *       event -- 输入参数,接收事件
 * 返回: 无
 * 修改:
 * 注释:
 *********************************************************************************/
static void node_uart_callback ( uint8 port, uint8 event )
{
#define RBUFSIZE 128
    (void)event;                                          //此处暂不定义参数
    uint8 ch;
    static uint8 szBuf[RBUFSIZE];
    static uint8 len = 0;
    //接收通过串口传来的数据
    while (Hal_UART_RxBufLen(port)){                      //获取数据
        HalUARTRead (port, &ch, 1);                       //提取数据赋予 ch
        if (len > 0){                                     //如果长度大于 0
            szBuf[len++] = ch;                            //将 ch 值赋予 szBuf 并将地址加 1
            if (len == 7){                                //如果长度为 7,数据接收完毕
                uint16 crc;
                crc = calc_crc(szBuf, 5);                 //循环冗余检验,检验最后两字节
                if (crc == ((szBuf[6]<<8) | szBuf[5])){   //如果正确
                   dqy_VAL = (float)((szBuf[3]<<8) | szBuf[4]);//提取传感器采集光照强度值
                }
                len = 0;                                  //数据提取成功,len 置零
            }
            }else{
                if (len == 0 && ch == 0x01){              //当检测到数据 0x01 且 len 为 0
                   szBuf[len++] = ch;                     //将 ch 值赋予 szBuf 并将地址加 1
                }else{
                   len = 0;                               //否则数据错误,len 置零
                }
            }
       }
}
```

▶ 5.4.5 操作步骤

1. 硬件部署

(1) 实验平台接上 220V 电源线,确保电源总开关已开启,然后按下按键群组的所有电源开关,确保实验平台的各个工作单元正常上电。

(2) 这里涉及 5 个节点与 8 个传感器,正确将传感器接入对应的 LiteB 节点,并确认 LiteB 节点的跳线是否正确。(默认出厂设备已经连接好。)

2. 程序调试

(1) 程序烧录。

① 协议栈安装。确认已安装 ZStack 的安装包。如果没有安装,打开光盘提供的安装包,路径为“DISK-IwsPlat\02-软件资料\02-无线节点 ZStack.zip”,解压即可。ZStack 的安装包路径如图 5-7 所示。

② 将 DISK-IwsPlat\02-软件资料\02-无线节点文件复制到 ZStack 的 Projects\zstack\Samples 文件夹下,Samples 文件夹如图 5-8 所示。

图 5-7　ZStack 的安装包路径

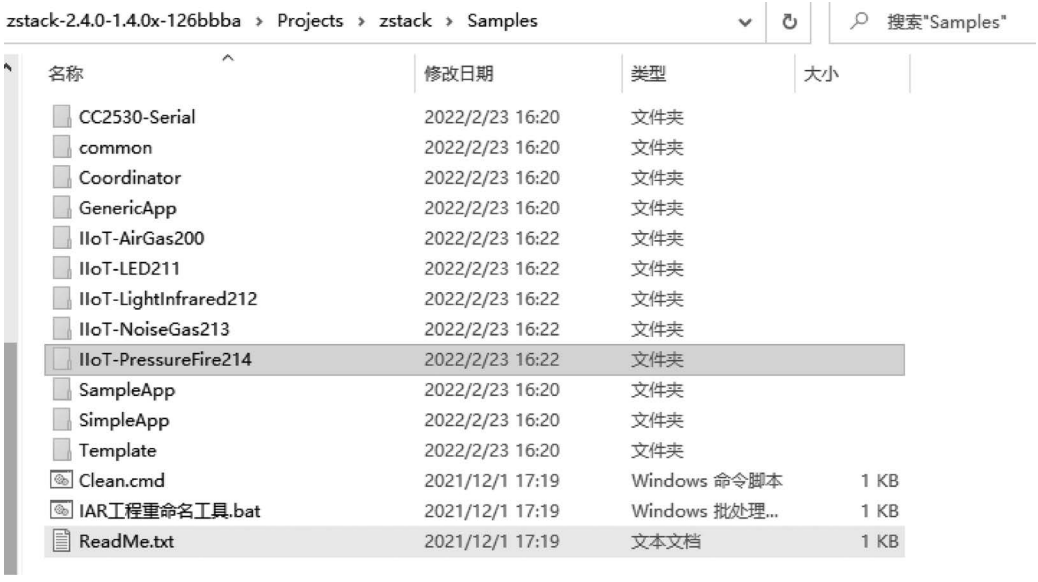

图 5-8　ZStack 的 Samples 文件夹

双击打开复制的文件夹内 CC2530DB 文件夹中的 .eww 格式的文件，打开对应的传感器工程。程序对应节点为

IIoT-AirGas200：空气质量无线节点(A)。

IIoT-LED211：电子横幅无线节点(A)。

IIoT-LightInfrared212：光照度(A)、人体红外(B)无线节点。

IIoT-NoiseGas213：噪声(A)、燃气(B)无线节点。

IIoT-PressureFire214：大气压力(A)、火焰(B)无线节点。

③ 下面以烧录空气质量为例。双击打开 zstack-2.4.0-1.4.0x-126bbba\Projects\zstack\Samples\IIoT-AirGas200\CC2530DB 文件夹中的 IIoT-AirGas200.eww。

④ 配置 PANID。根据实际需求分别设置温湿度传感器、光照度传感器、空气质量传感器、电子横幅工程及透传程序 PANID，编译工程(当有多组操作同时进行时，为了避免冲突，需要根据实际硬件平台修改节点 PANID(如用学号后 4 位表示，范围为 0x0001～0x3FFF)，工程文件为 Tools ->f8wConfig.cfg)。PANID 配置如图 5-9 所示。

⑤ 烧写程序。分别单击刚打开的 IAR 工程界面上方的图标依次编译工程。当工程全部编译通过以后将 SmartRF04EB 仿真器 USB 线一端连接到计算机，调试端连接到当前工程对应的 CC2530 调试接口上。单击 IAR 编译器界面上方的图标，等待程序烧写完成。在烧

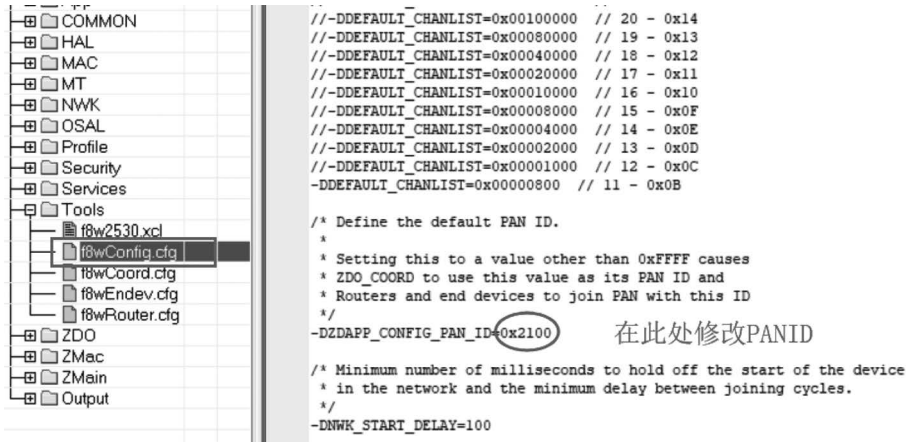

图 5-9　PANID 更改页面展示

写完成后单击工程上方的✕图标。退出调试模式并更换 CC2530 节点，重复烧写程序操作将剩余节点对应的程序全部烧写完成。

（2）工业网通板卡汇集节点网络修改。

方式一：在工业网通板卡界面上修改。

① 使用操作界面修改时，首先选择“网关设置”，然后选择下方的 ZIGBEE 栏目，单击“设置”按钮，弹出修改界面，如图 5-10 所示。

图 5-10　在工业网通板卡修改界面

② 在弹出的界面中修改 PANID 和 CHANNEL 参数，修改完成之后单击“确定”按钮，如图 5-11 所示。

③ 修改完成之后界面提示是否重启，重启之后修改的参数才能生效，单击“重启”按钮系统立即重启，如图 5-12 所示。

④ 重启完成之后，此时再次查看 ZigBee 网络参数，已修改成功，如图 5-13 所示。

方式二：通过串口终端修改。

① 使用串口终端修改时，首先使用 MiniUSB 线连接计算机和工业网通板卡，连接方式如图 5-14 所示。

图 5-11　单击“确定”按钮

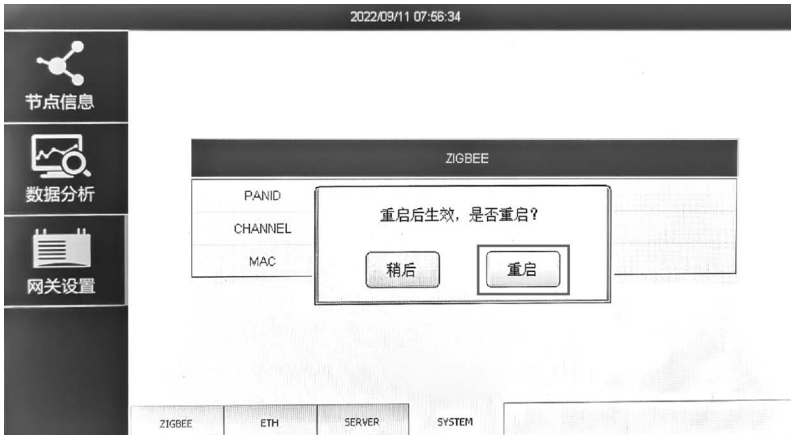

图 5-12　单击“重启”按钮

图 5-13　修改成功界面

图 5-14　计算机和工业网通板卡连接方式

② 打开 MobaXterm 软件(路径：DISK-Packages\51-常用编程工具\MobaXterm)，单击 Session 新建会话，在弹出的界面中选择 Serial 串口会话，如图 5-15 所示。

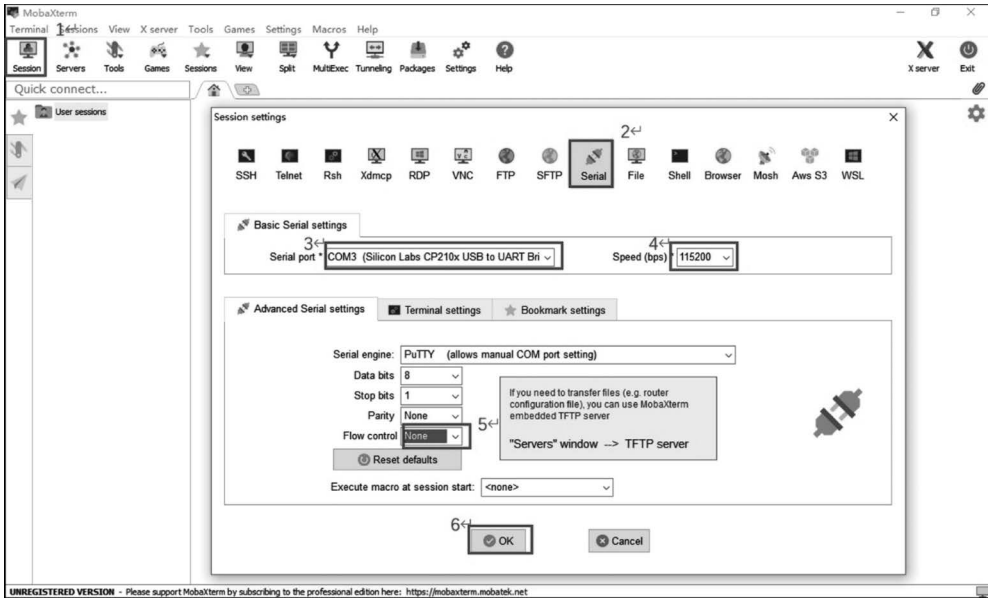

图 5-15　Serial 串口会话界面

在 Serial 串口会话界面中选择对应的串口号，波特率选择 115200，流控选择 None 选项，配置完成之后单击 OK 按钮，即可进入终端界面，按 Enter 键即可进入 Shell 界面，如图 5-16 所示。

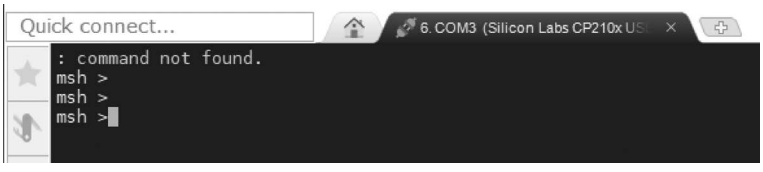

图 5-16　Shell 界面

③ 在终端输入 zigbee 命令，查询 ZigBee 网络参数，如图 5-17 所示。

图 5-17　输入 zigbee 命令

输入：zigbee panid [数值]，修改 PANID 为此处的数值。输入：zigbee channel [数值]，修改 CHANNEL 为此处的数值，如图 5-18 所示。

备注：设备组网时 LiteB 节点和汇集节点的网络参数必须相同，并且不同实验台的网络参数不能相同，否则在组网时会互相冲突。

(3) 数据上云。

① 确保工业网通板卡已经通过网线连接交换机，并且交换机能够连接外网，使得工业网

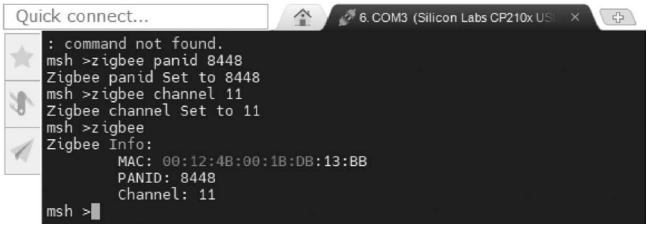

图 5-18 输入 zigbee panid [数值] 命令

通板卡收到无线传感网数据能够推送到云平台(默认情况下已完成配置)。

② 使用 MobaXterm 软件连接工业网通板卡,通过终端命令查询或设置工业互联网云台认证的 ID 和 Key 参数。终端输入:gateway idk,查询 IDKey 参数。输入:gateway idk [id 值] [key 值],即可设置 IDKey 参数。具体操作如图 5-19 所示。

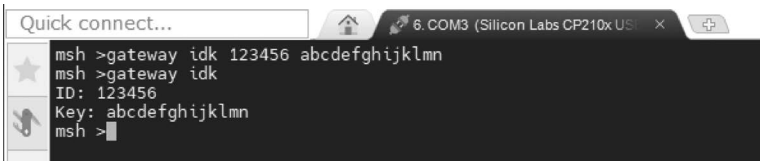

图 5-19 输入 gateway idk 命令

备注:此处需要使用经过授权的 ID KEY 才能将数据接入云平台(可寻求厂商获取授权)。

③ 输入 ID KEY 后,工业网通会提示,是否重启设备,重启生效,选择重启,如图 5-20 所示。

图 5-20 选择重启

④ 重启后,MobaXterm 工具串口提示“eth:zhiyun connect!”,即代表服务器连接成功,如图 5-21 所示。

3. 效果演示

解压 ZCloudWebTools(路径:DISK-IwsPlat\02-软件资料\06-测试工具\ZCloudWebTools.zip),使用 Chrome 浏览器打开 index.html,在“网络拓扑”栏的账号和密钥处分别填写工业网通板卡的 ID 和 Key 信息,填写完成之后,单击“连接”按钮连接服务器,等待数据上线之后,即可查看网络拓扑结构,空气质量、光强、人体红外、噪声、燃气、大气压力、火焰节点,以及电子横幅节点上线,显示数据已推送至云平台,如图 5-22 所示。

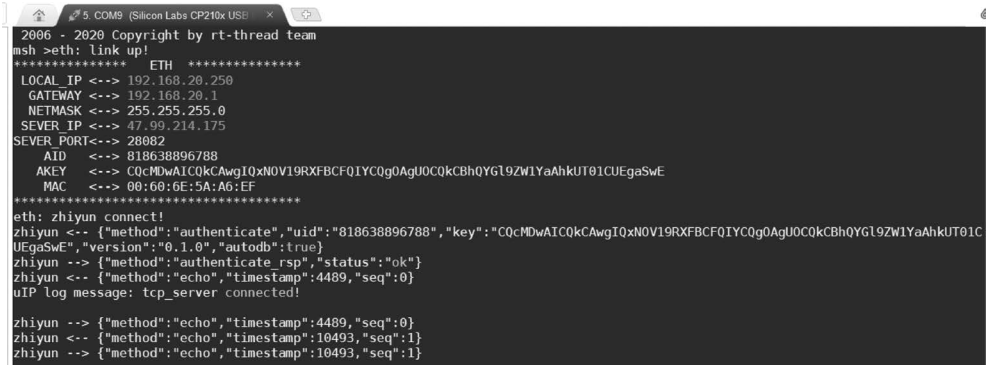

图 5-21　服务器连接成功

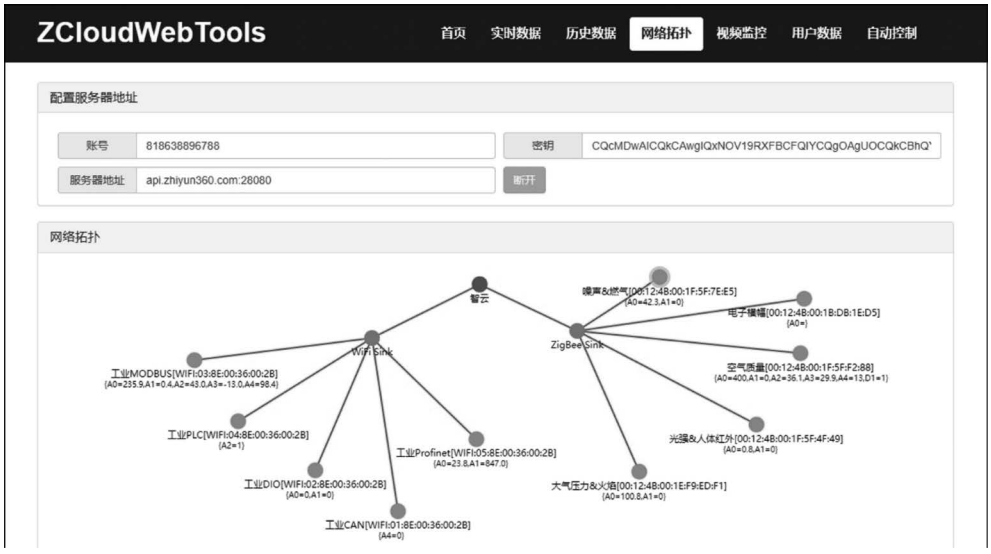

图 5-22　数据推送至云平台

习　　题

简答题

1. ZigBee 技术的主要特点是什么？
2. 为什么选择 ZigBee 技术来实现无线数据采集？
3. ZigBee 网络的通信协议栈包括哪些层次？

习题 5