基于 ZigBee 的远程老人监护系统

1 前言

1.1 问题背景

步入 21 世纪以来，我国的人口老龄化日益严重，独居老人问题日渐突出。据相关权威部门的资料显示，截止到 2014 年，全国老龄人口已达近 2 亿，为全国总人口比重的 15%。专家分析，到 2030 年该比率将继续增大，且独居老人家庭的比例可能会高达 95%，更多相关资料表明，届时我国将会有超过 3 亿的独居老人[1]。独居老人是指那些独自居住、没有亲人和伴侣陪伴的老年人群体，是一个弱势群体。近几年，独居老人意外离世的新闻屡见不鲜，引起社会的诸多关注。尽可能地为独居老人提供安全、健康、快乐的生活条件，为老人创造一个富有爱心的家园，便成为每一个人都应当思考的问题。 老年人视力减退、腿脚不够灵活、反应迟缓，容易出现一些意外，比如：上下楼梯时摔倒、忘记关燃气阀门、开水烫伤等事故。独居的老人更是因为独自居住、身边缺少人员看护和照顾，经常意外发生多日不能得到发现与治疗，甚至还会导致各种悲剧的发生。此外，当独居老人的住所环境发生变化但是没有及时发现，也可能带来难以预料的后果，比如突发疾病、有害气体泄漏、入室抢劫等。因此，对于独居老人的居住环境进行及时有效的监测就显得尤为重要。

1.2 研究现状

早在 1960 年，在美国海军资助下美国相关医疗院所就开展了第一代可穿戴设备的研究并用于远程医疗监护，最初研发的目的是为了战场的士兵生理状况监测，可以检测人体体温，心率，脉搏，呼吸等。后来，美国一些医疗保健公司将此技术引入民用，开发出一种称之为“生命衫”的设备。使用这个设备可以实现血压，血氧浓度，体温等生理参数的测量，然后通过 PDA 等手持设备传输至互联网，经数据处理分析中心的处理，将监测结果返回给医生和用户。在此期间，无线通信技术的发展极大推动了这类监控系统的适用范围，实现了移动性。第三次通信技术的发展，极大推动了各类监控系统的传输能力[2]。而在国内，此类个人监测系统的发展轨迹同步于智能家居

的发展，智能家居近十年来在国内的发展，极大促进了相关的采集技术，无线网络传输技术，以及各类自动控制技术。而智能家居的发展趋势有如下特点：从有线走向无线，从模拟信号转向数字信号。而其中很大一个理念就是万物联网，而组网又分为有线网和无线网，传统的有线网传输可靠但是扩展麻烦不具备移动性，因而智能家居选用的是无线组网技术。传统的技术有红外、蓝牙、WIFI 技术。而近些年新崛起的 ZIGBEE 技术，拥有功耗低，组网功能强大，成为了智能家居无线网络的重要解决方案[3]。

1.3 研究意义

针对以上空巢老人的现状，本文设计了一种基于物联网技术的远程老人监护系统。这是一个将传感器技术、无线通信技术和 Linux 系统开发相结合的环境监测系统。系统的功能如下：基于 Exynos4412 芯片，利用温湿度传感器、烟雾有害气体传感器、人体红外感应模块检测老人周围环境的温度与湿度和所处环境中各种燃气浓度信息，判断周围环境是否对老人有隐患、快速识别环境中的烟雾指数以判断是否有火情；检查是否有坏人入室；如果检测出指标异常则立即通知其家属，联系相关人员前往处理救治，并在必要条件下发出警报。同时取得的环境信息通过 ZigBee 无线网络存入嵌入式服务器 SQLite，以供用户随时访问和后期分析。按照功能系统被划分为三个模块：数据采集模块，无线传输模块，数据分析处理模块。随着物联网技术、移动通信技术和嵌入式操作系统技术的不断发展，本文希望能够借助这些先进技术构建一套适用于独居老人的常居环境监护系统。这套系统可以在不需要太多人工操作的情况下，对老人周围环境进行实时智能监控，为老人的健康保驾护航，同时也可以减轻子女的担心和照顾压力，对于整个老人团体将有巨大的意义。

1.4 本文的研究内容及主要工作

1.4.1 系统介绍

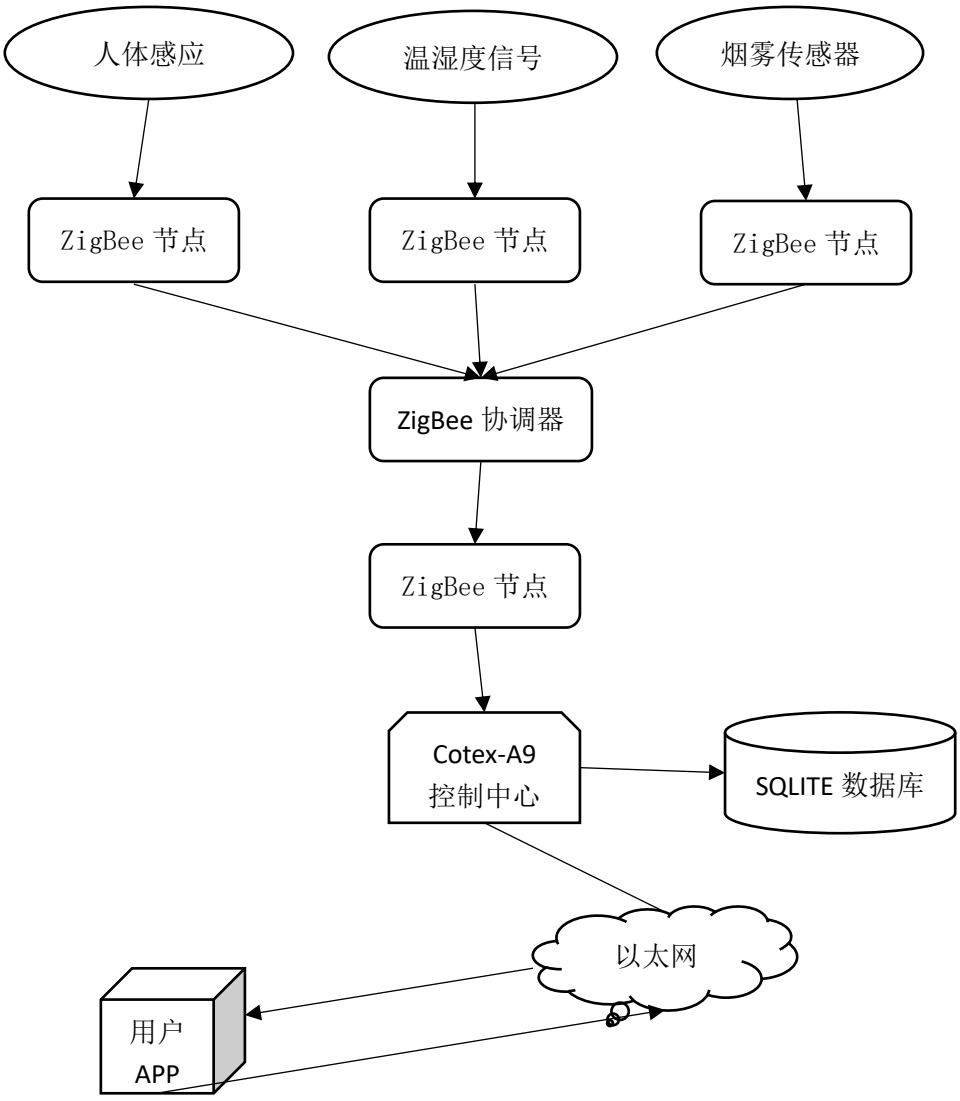


图 1　系统总拓扑图

本系统采用 CC2530 芯片进行数据采集，搭建基于 ZigBee 协议的无线传输网络，传输采集数据至智能网关，并在 Contex-A9 主控芯片植入 Linux 系统，进而搭建嵌入式服务器 SQLite，对采集数据进行实时处理和分析，实现信息采集、无线传输、智能处理、上传云端后期分析、以及紧急情况的预警。老人与其家人均可以通过手机 APP 访问数据库获取当前环境信息。系统主要实现的功能如下：在采集模块，控制中心 Exynos4412 模块启动采集数据进程，进而向每个采集结点的 CC2530 发送对应指

令，CC2530 驱动相应传感器工作，采集的数据返回到 CC2530。一旦 CC2530 拿到了数据，通过 ZigBee 无线网络传输至 Exynos4412 模块中（在 Exynos4412 模块中已搭建好嵌入式服务器 SQLite）；在分析模块,在 Exynos4412 模块中运行裁剪后的 Linux 系统，启动子进程对采集数据进行分析，如果有危险，进入控制逻辑 1；否则进入控制逻辑 2；在控制模块，在 Exynos4412 模块上运行的 Linux 系统之上，编程并结合硬件外围电路，实现两类控制逻辑；控制逻辑 1：通过 ARM 启动报警逻辑并显示在 APP 上。控制逻辑 2：从 Exynos4412 模块中已部署的服务器中拿到刚检测的数据进行展示，在 APP 中绘制各类指标的曲线图。监护人后期可随时提取这些历史数据进行分析。

1.4.2 本文研究的主要内容

第一章为前言，介绍了老人监护系统出现的背景及现实意义，并且综合国内外此前已取得的研究成果，并基于此，提出了本文的基于 ZigBee 的远程监护系统。

第二章介绍系统的第一个模块，即数据采集模块。目前包括三个采集结点：温湿度采集节点、室内有害气体采集节点、人体感应采集节点。这三个节点均由 CC2530 采集数据，并将采集的数据经过处理后，借助基于 ZigBee 协议搭建的无线局域网传输至主控中心，即 Cotex-A9 处理器，作进一步的数据分析和反馈。

第三章介绍系统的第二个模块，即无线传输模块。采用的是基于 ZigBee 协议的 802.0.0.1 的无线传输网络。ZigBee 具有高效率、低功耗以及传输距离适中的诸多优点，非常适合在各类局域网中承担数据传输的任务。考虑到本系统后期的可扩展性，采用基于 Zstack 组网的方式，采取星形拓扑结构，各个模块可以非常方便地加入局域网。这对于提升系统的可扩展性和鲁棒性具有重大意义。

第四章介绍系统最为核心的一个主要模块，即主控模块。首先，基于 Cotex-A9 的处理器平台，移植裁剪后的 Linux-3.10 系统；然后，基于刚移植的 LINUX 系统搭建嵌入式服务器 SQLite；再然后，基于上述平台编写程序使 Cotex-A9 通过 UART 串口与协调器进行数据通信，进而编写控制程序对传感器数据进行实时采集、反馈、报警功能；在最后，编写安卓 APP 实现数据的实时查看与手机控制采集过程的两个功能。

第五章，是基于前三章的技术进行系统实现和测试。所涉及的传感器和硬件如下：DHT11 温湿度传感器、MQ-2 模块烟雾气敏传感器模块、人体红外传感器模块

HC-SR501、CC2530 核心板、Exynos4412 核心板、LED、蜂鸣器相关电路模块。依次构建采集系统、无线传输网络、主控系统、APP 以及上位机访问例程。并将过程中的采集的数据、传输过程的抓包演示、以及 APP 对传感器网络进行反馈的结果，均以截图的方式予以展示。

第六章，是对本系统的优缺点进行总结，对已取得的成果予以记录，对系统存在的问题进行记录和反思，并针对系统存在的不足进行展望，期待在后期的学习生涯进行进一步的改善，希望借此对整个本科四年的计算机基础课程有个完整的梳理。

2 采集模块

2.1 CC2530 处理器

CC2530 基于 IEEE 802.15.4、ZIGBEE 以及 RF4CE，是一款高效的 SOC 解决方案。它具有高性能低成本的特点。CC2530 内部集成了非常优秀的 RF 收发器，性能卓越的 8051MCU，结合了德州仪器的 ZIGBEE 协议栈 ZSTACK，使用 CC2530 能够方便地使用 ZIGBEE 协议。CC2530 根据内部 FLASH 容量分类，即：CC2530Fx(x 分别为 32、64/128/256) [3]，CC2530 的主要特点如下表 1。

表 1　CC2530 特点

优势与特点
完全符合 ZIGBEE 协议栈
小体积 SMD 表封装
IEEE802.15.4 标物理层和 MAC 层
单指令周期高性能 8051 微控制器内核
UART、SPI 和调试接口
板载 32.768kHZ 实时时钟（RTC），4 个定时器
高性能直接序列扩频（DSSS）射频收发器

CC2530 芯片内部集成了 USART 串口通信接口和众多寄存器，非常方便我们与其他微控制器的串口通信。比如，USART0 和 USART1 是两个串口通信接口，他们能够分别运行于异步 UART 模式或者同步 SPI 模式。两个 USARTx 具备同样的功能，可以分别设置在单独的 I/O 引脚。UART 操作由 USART 控制和状态寄存器 UxCSR 以及 UART 控制寄存器 UxUCR 来控制。当 UxCSR 位 MODE 设置为 1 时，就选择了 UART 模式。对于每个 USART 接口，有 5 个寄存器[3]。此外，CC2530 芯片的内部结构如下图 2。

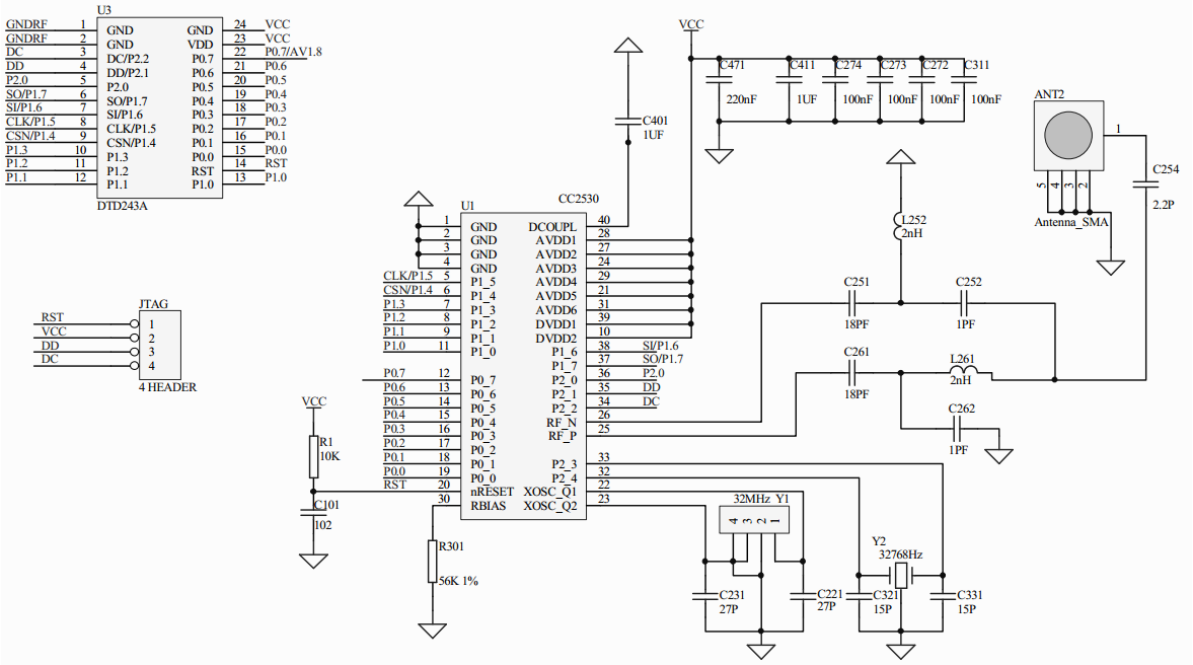


图 2　CC2530 内部结构图

表 2　USART 寄存器

寄存器名称	寄存器功能
UxCSR	USARTx 控制和状态
UxUCR	USARTxUSAR 控制
UxGCR	USARTx 通用寄存器
UxDBUF	USARTx 数据缓冲器
UxBAUD	USARTx 波特率控制

2.2 温度采集节点

2.2.1 DHT11 芯片原理简介

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保传感器的稳定性和精确性。传感器包括一个电阻式感湿元件和一个 NTC 测温元件，并与一个高性能 8 位单片机相连接。每个 DHT11 传感器都在极为精确的湿度校验室中进行校准，典型应用电路图如下图 3[3]。

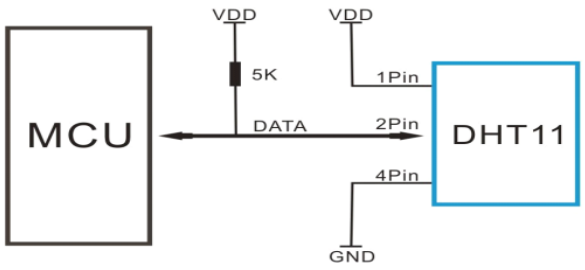


图 3　DHT11 典型应用电路

首先左边接一个微处理器 MCU，通过数据线与 DHT11 的 P2 口相连，P1 接电源 VCC，P4 接地 GND。DHT11 不会主动采集数据，必须等到 MCU 发送采集命令方可启动。

2.2.2 DHT11 芯片引脚功能描述

关于串行接口 DATA 的说明：DATA 用于微处理器与 DHT11 之间的通讯和同步，采用单总线数据格式，一次通讯时间 4ms 左右，数据分小数部分和整数部分。当传感器发送命令时，DATA 在 SCK 上升沿有效且在 SCK 高电平时必须保持稳定。DATA 在 SCK 变成下降沿之后改变[4]，DHT11 芯片引脚分布图以及内部结构如下表 3。

表 3　DHT11 芯片引脚分布

Pin	名称	注释
1	VDD	供电 3-5.5VDC
2	DATA	串行数据，单总线
3	NC	空脚，悬空
4	GND	接地，电源负极

2.2.3 MCU 与 DHT11 通讯

(1) 启动传感器：首先，选择供电电压后让传感器上电，通电后传感器需要 11ms 进入休眠状态，在此之前不允许对传感器发送任何的命令。

(2) 发送测量命令：通过一组时序来完成数据传输的初始化。它的过程如图 4。后续命令包括 3 个地址位、5 个命令位。

(3) 温度测量：发送一组测量命令后，当 DATA 跳变低电平时，DHT11 进入空闲模式，测量结束。进入读取时序后，所有数据从 MSB 开始，右值有效，CPU 依次测量每一个比特位，直至 LSB。保存测量数据并结束通讯，进入休眠模式。

(4) 数据计算：DHT11 内部已集成了计算公式，测量的数据可通过串口助手查看。

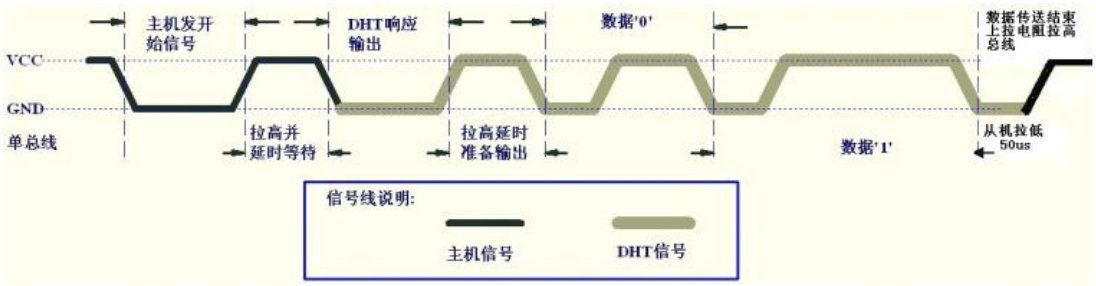


图 4 传输时序图

首先，总线处于空闲状态即高电平，主机 MCU 把总线拉低至低电平作为采集开始的启动信号，然后等待至少 18ms，DHT11 检测到启动信号后，将总线拉至高电平，作为对主机 MCU 的响应信号，持续至少 80uS。等待主机 MCU 检测到来自 DHT11 的响应，可进入 Input 模式，由上拉电阻将总线拉高至高电平[4]。

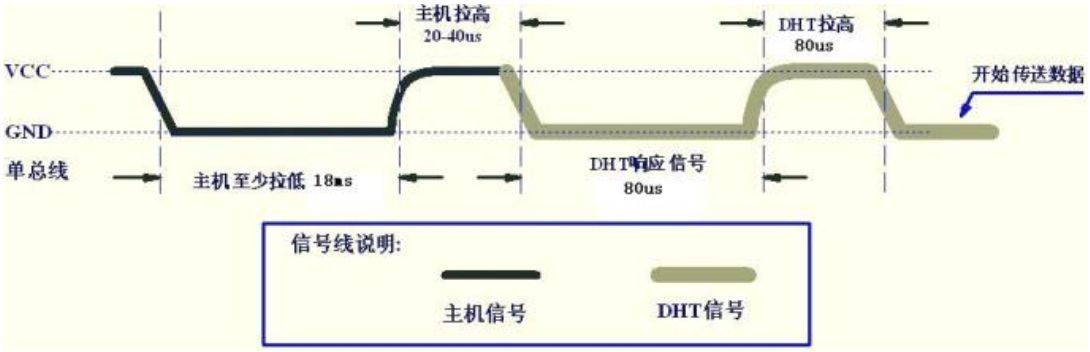


图 5 传输时序图

然后，当 DHT11 拉高总线 80us 后准备发送数据，每一个比特的开始时刻都是在 50us 的末尾，根据后面高电平持续时间的长度分为 0 和 1，详见下图 5。当最后一个比特传输完毕，DHT11 将总线拉低持续 50us，随后使之进入高电平即空闲模式[4]。

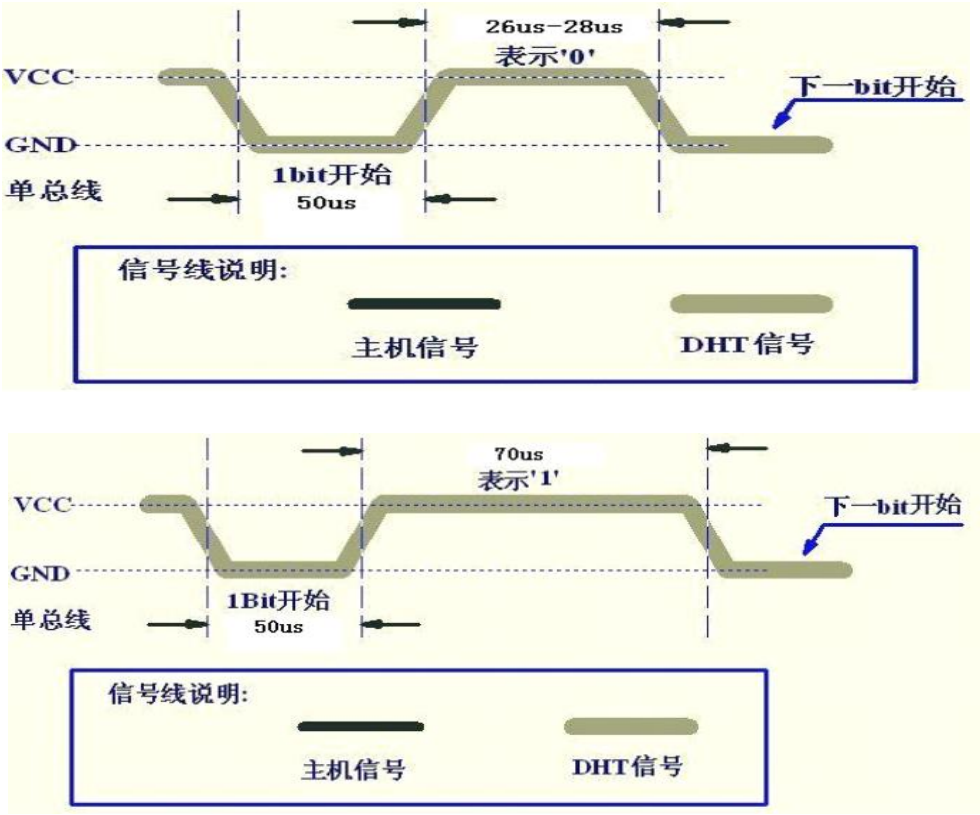


图 6　数据中 0 和 1 的表示

如上图 6，在 DHT 信号 1 比特的末尾，它由低电平跳变至高电平并持续 26~28us 时，表示此刻的数据信号为 1，直至电平变至低，开始下一个比特的传输。如果 DHT 信号线中间持续的时间为 70us，那么表明此刻的数据信号为 0，等待电平变低重新开始下一比特的传输。

2.3 烟雾报警节点

2.3.1 MQ-2 传感器原理简介

MQ-2 型可燃气体/烟雾传感器属于表面离子式 N 型半导体。烟雾浓度越大，电导率越大输出电阻越低。 使用简单的电路即可将电导率的变化转换为与该气体浓度相对应的输出信号。可检测多种可燃性气体，是一款适合多种应用的低成本传感器[5]。

2.3.2 MQ-2 传感器引脚描述

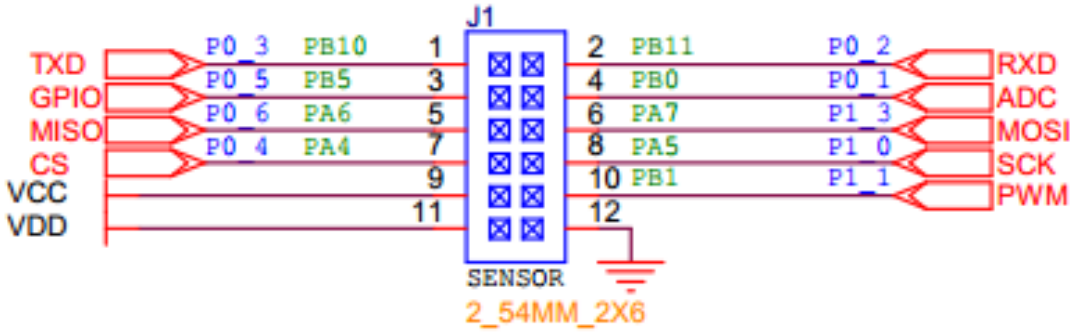


图 7　MQ-2 引脚图

首先，GND 接电源负极，VCC 接+5V 电源，DOUT 接 TTL 电平输出端，AOUT 接模拟电压输出端。此外，传感器模块上还集成了 TTL 输出灵敏度调节。

2.3.3 烟雾传感器电路原理

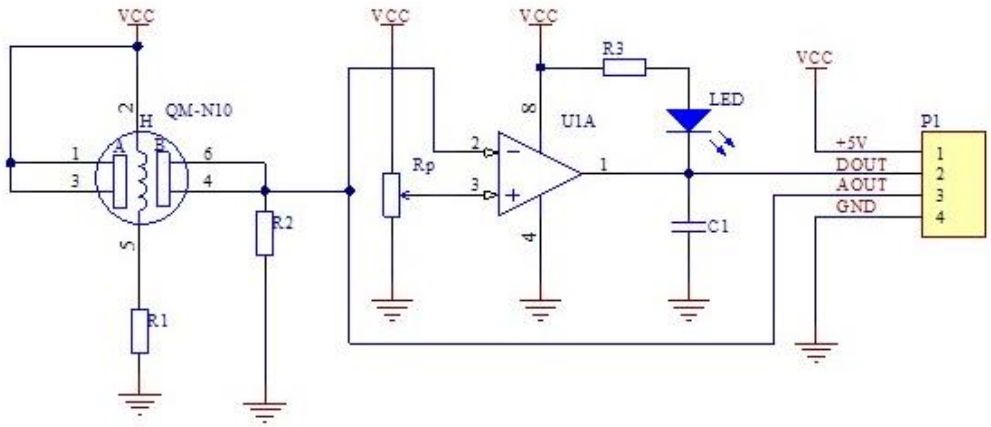


图 8　MQ-2 电路原理图

首先，当检测到可燃气体，MQ-2 气敏型元件 4 脚输出信号会发生变化，进而引起 U1A 的 2 脚的电平发生变化。当 2 脚的电平高于门限电平即 3 脚电平时，比较器 U1A 的 1 脚输出电平为 0V 即低电平，LED 点亮报警。否则，1 脚输出 VCC 即高电平，LED 不亮。同时 1 脚输出电平经过 DOUT 以及 QM-N10 的 4 脚输出电平经由 AOUT，发送到 MCU。本系统的 MCU 即 CC2530 芯片通过 ADC 来读取可燃气体/烟雾传感器的输出的值[6]。

2.4 人体感应采集节点

2.4.1 HC-SR501 模块原理简介

本系统的人体感应模块采用的 HC-SR501 基于红外技术，工作电压为直流 4.5v-20V，全自动感应，根据无人/有人自动输出为 0/3.3V 的高低电平。可通过设置光敏控制和温敏控制来指定特定工作环境，本系统设置为夜间开启。该传感器具有高灵敏性、低功耗、使用简单等优点。正常人体的体温恒为 37 摄氏度，会发射恒定波长为 10UM 的红外线。HC-SR501 模块装载有被动式红外探头，一旦模块检测到到 10UM 的红外线，经由模块上集成的菲涅尔滤光片加强后聚集到红外感应源元件，引发元件的热释电效应，会向外释放电荷，经过处理电路进行捕捉放大，即可检测到人体。从而引发进一步的报警功能[6]。

2.4.2 HC-SR501 引脚描述

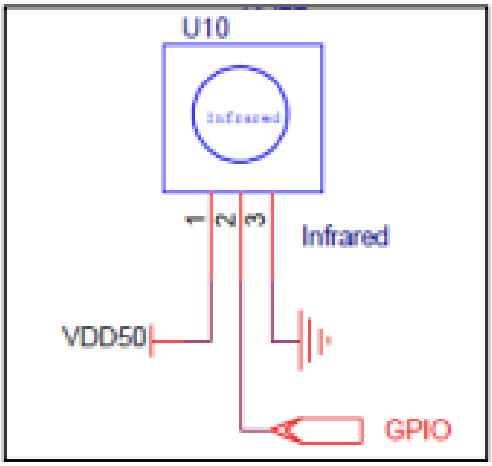


图 9　红外传感器原理图

如上图 9，HC-SR501 的 3 脚输出口通过 GPIO 连接到 CC2530 的 P0x 端口，VDD50 接电源，GND 接地。

2.4.3 检测原理与流程

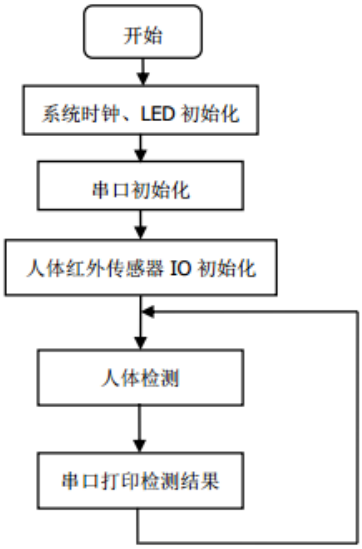


图 10　检测流程

如上图 10，当 HC-SR501 人体感应模块检测到有人体活动时，其输出的 IO 值发生变化。当传感器模块检测到有人入侵时，会返回一个高电平信号， 无人入侵时，返回一个低电平信号，通过读取 I/O 口的状态判断是否有人体活动。本系统采用 SHT1x 作为采集传感器。开启一个循环不断查询端口的电平值，结果由串口输出至 MCU-CC2530。

3 无线传输模块

3.1 无线传输技术对比

现代社会的移动通信产业的迅猛发展，为物联网行业万物相联的夙愿变成了可能，而目前的物联网行业广泛使用的无线网络技术有三种：蓝牙、WIFI、ZIGBEE。蓝牙是一种支持设备短距离通信的点对多点的无线数据传输技术。蓝牙技术的加入简化了各类智能设备与互联网的连接，通常意义下的蓝牙技术使用 2.4GHZ 的 ISM 频段，引用跳频技术（FHSS），传输范围 10m 之内，最大可加强至 100m，具备效率高，成本低，操作简单的特点。目前蓝牙技术主要应用于手机和计算机的连接、数据共享、办公，Internet 接入、无线车载免提、各种家庭设备的遥控和家电设备的联网[6]。WIFI 技术和蓝牙技术一样，属于适用于短距离的通信解决方案，使用 2.4GHZ 频段。WIFI 是由接入点 AP 和无线网卡组成的无线网络，结构简单，可以实现快速组网，成本低于有效效率却接近有线。WIFI 无线局域网又分为：接入点模式和无线接入点模式。并且由于 WIFI 所使用的频段是免费频段，故而其应用极其广泛，目前大致分布在：无线 POS 机、医疗监控、工厂监测系统、交通监控、港口码头类的物流行业等等[6]。

ZIGBEE 技术是一种基于 IEEE 802.15.4 的无线传输网络，创建初始就具备低功耗、低成本、低开发难度的优点，自 ZIGBEE 联盟成立，已经广泛应用于智能家居、工业监控、传感器网络、和其他电信领域。无线传感器网络综合了微电子技术、嵌入式计算技术、现代网络及无线技术、分布式信息处理技术等技术。考虑到本系统的传输距离要求、稳定性，以及功耗方面的要求，将采用 ZIGBEE 作为组网方案[6]。

此外，ZIGBEE 通信方式有三种：点对点，组播，广播。考虑到系统的可扩充性，本系统采用组播的方式。建立一个局域网，各个采集结点加入此局域网进行数据传输。

3.2 ZigBee 技术简介

3.2.1 ZigBee 协议的体系结构

ZigBee 基于 IEEE 802.15.4 无线标准，作为一种用于通讯的组网协议，ZigBee 的协议栈位于 IEEE 802.15.4 物理层及数据链路层的规范之上，ZigBee 的协议栈示意如下表 4[7]。

表 4　ZIGBEE 协议栈简介

用户应用程序
ZIGBEE 设备对象
应用层
应用支持子层
网络层
数据链路层
物理层

ZigBee 协议的最低两层是由 IEEE 802.15.4 标准定义，其他各层均是 ZigBee 相关组织制定，分别为：网络层、应用层以及其他。其中应用层提供应用支持子层 APS 和 ZigBee 设备对象 ZDO 等服务[7]。

(1) 物理层：定义了 MAC 层和信道之间的接口，在硬件驱动的基础上，实现物理信道、数据传输的管理，提供物理层管理及数据服务。其主要职责是：物理信道的能量检测、数据的发送和接收、空闲信道评估等等[7]。

(2) 数据链路层：MAC 层和 NET 层之间的接口，提供 MAC 层的数据及管理服务。其主要职责包括：采用 CSMA/CA 机制来控制访问；协调器对网络的维护和组建；通讯同步使两 MAC 实体之间开展可靠的数据传输，支持安全机制等[7]。

(3) 网络层：定义了 NET 层与 APP 层的接口，提供 Network level 和 Application level 的管理与数据服务。它的主要功能：提供设备间连接和断开网络的通用机制；数据传输过程中的安全机制；各类设备的路由、维护以及转交；当首次创建新的网络时为新的设备分配短地址，等等[7]。

(4) 应用层：APP 层定义了 AF 和 NWL 之间的接口。主要由 Application Support level、ZigBee device level 和用户应用程序组成。提供了应用层管理服务和应用层数据服务[7]。

3.2.2 ZIGBEE 协议设备类型

ZigBee 网络协议定义了三种 ZigBeeDevice Type：

(1) Coordinator: Coordinator 负责开启整个网络。它是网络中的首个并且也是最复杂、最难控制的一个设备。它主要用来建立新的网络、发送网络标识、管理网络节点、

存储网络各个节点的信息。协调器也可以用来协助建立网络中的 SL 和 AL 的绑定[8]。

(2) Router: Router 的功能主要是用来扩展网络的规模，以及物理距离。它允许更多节点加入网络，并且必要时提供网络流量控制和监视功能[8]。

(3) End Device : End Device 没有特定的网络结构的责任，它属于用户端的设备，为了保证低耗可进入睡眠模式，或者被设备唤醒，属于整个网络的边缘设备[8]。

3.2.3 ZSTACK 简介

(1) ZSTACK 的结构。

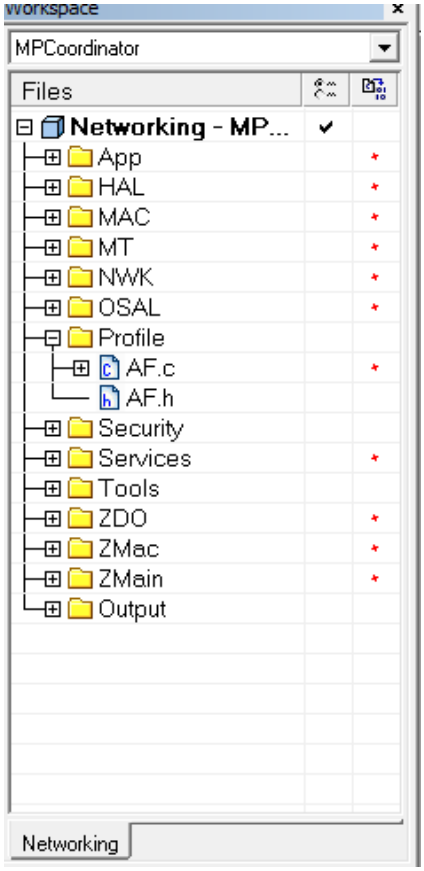


图 11　ZSTACK 文件目录

表 5　ZSTACK 目录功能介绍

名称	功能
APP	应用层，系统任务
HAL	硬件层，硬件配置信息、驱动
MAC	MAC 层配置信息和 LIB 库接口函数
MT	通过串口联系各层
NWK	网络层，配置文件、库函数接口文件
OSAL	协议栈的操作系统
Profile	应用架构层，包含 AF 层库函数
Security	安全层，包含安全层库函数
Services	地址处理库函数
Tools	工程配置文件等信息
ZDO	ZIGBEE 设备对象层，管理 ZIGBEE 中的设备
ZMAC	MAC 层目录，含 MAC 层配置参数和 LIB 库

上表 5 是基于 ZIGBEE 协议的 ZSTACK 协议栈，提供了一套库函数用于 ZIGBEE 应用程序开发。

(2) 运行原理：ZSTACK 是一个基于查询式是类操作系统。总体而言，系统的运行过程为一个大的循环。

(3) 系统初始化的主要工作如下图 12。

(4) 启动操作系统所做的工作：在完成初始化 OS 后，开始执行操作系统入口函数：OSal_start_system();整个系统开始进入查询模式：不断查询每个任务是否发生了事件需要处理。如果检测到 OK，进入处理子函数，否则继续循环地查询[8]。

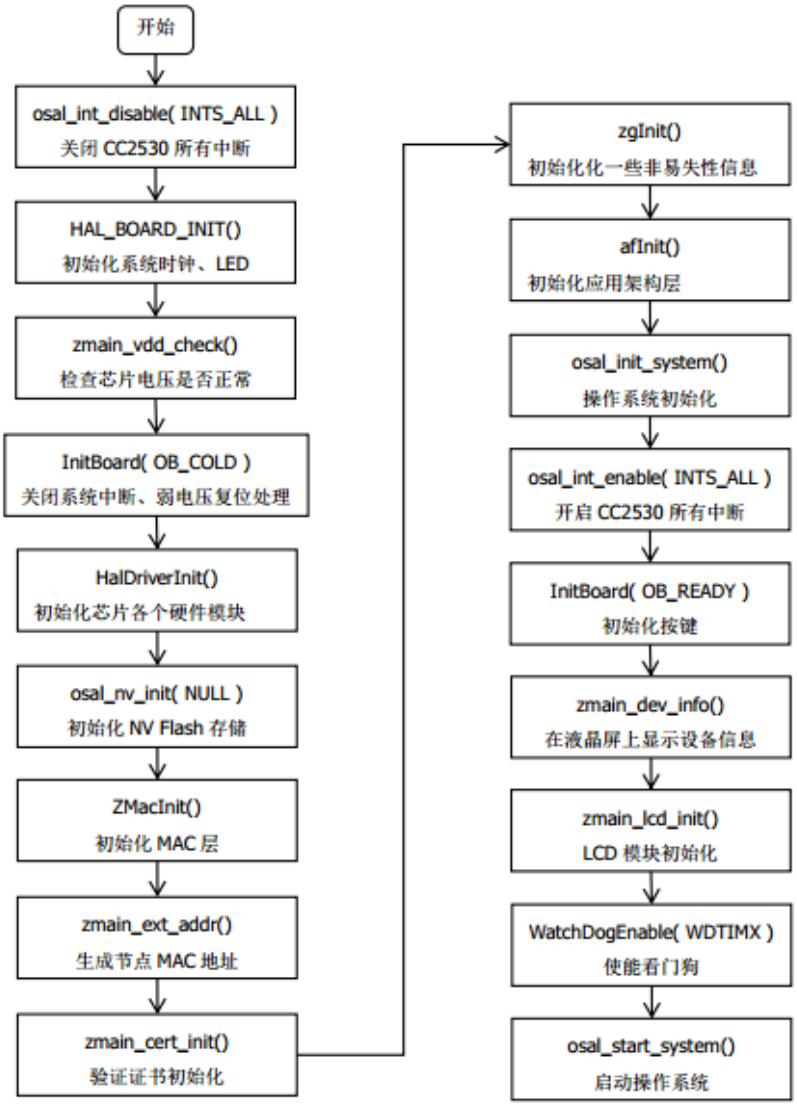


图 12　ZStack 系统初始化详解

3.2.4 基于 ZigBee 的组网

目前 ZigBee 组网模式有三种：网状、星形、簇状。它的拓扑图如下，其中核心的是协调器和路由节点，外围的是终端设备[8]。本系统基于 TI 的 Z-Stack 协议栈，构建一个无线局域网的流程图如下图 13。

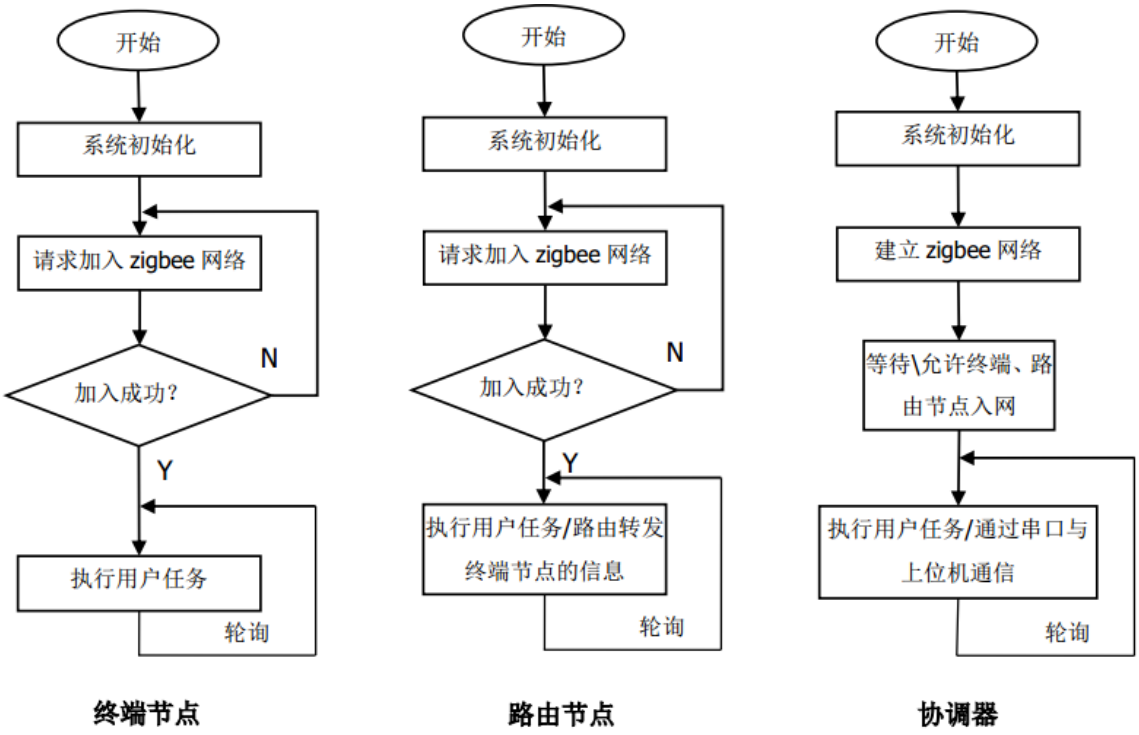


图 13　ZigBee 组网流程图

首先，由协调器建立 ZigBee 网络，启动轮询服务，等待节点加入。终端节点或者路由节点搜索附近网络，若发现则发起入网请求，协调器收到请求根据情况发出响应，若允许则终端节点（或路由节点）加入成功，开始进入执行用户任务阶段，而协调器往下继续执行轮询或者其他用户任务[8]。

3.3 系统通信协议设计

本系统涉及到传感器与协调器、协调器与智能网关、智能网关与远程访问设备之间的数据通信，为保证其可靠性，必须设计好通信协议。通信协议包括三部分：语义、语序、语法。语义规定协议实现的功能和数据内容，语序规定了通信双方的顺序，语法规定了数据格式和编码，本系统分三部分设计协议。

3.3.1 传感器与协调器

考虑到协议的设计要兼顾效率和简单，本模块的数据格式为：Data-Header + Data-Type+ Data-Length + Data + Reserve(保留位) + Check(检验位)，数据编码采用二进制。如下表 6 所示。

表 6 传感器与协调器之前的数据格式

Header	Type	Length	Data	Reserve	Check
2 Byte	1 Byte	1 Byte	4 Byte	2 Byte	1 Byte

如上表 6，Header 为固定部分，0xFF 和 0xFD；Data-Type 的值有三种，0x00 表示温湿度传感器数据，0x01 表示烟雾传感器数据，0x02 表示人体红外传感器数据；Length 为固定值 4；Data 有 4 字节，为采集的数据；Reserve 是保留位，以备系统后期扩展使用，默认为 FFFF；Check 是校验位。例如，数据为 0xFFFD010412345678FFFFHH，它的含义就是：把烟雾传感器数据 0x12345678 传至协调器，校验位为 HH。

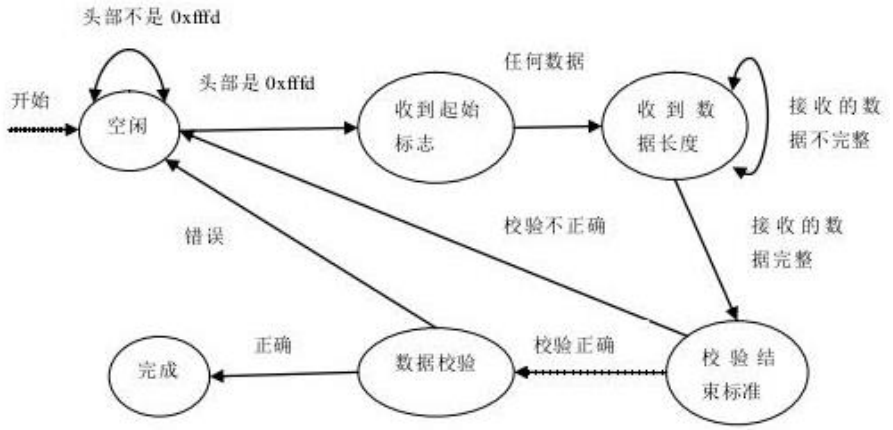


图 14　传感器与协调器通信时序图

3.3.2 协调器与智能网关

协调器与智能网关之间的通信属于双向通信，故而需要分上行和下行两种方向。上行方向，即协调器发送给智能网关的数据包完全沿用表 3.3 的格式，直接使用；而下行方向，即智能网关发送给协调器的数据格式定义如下表 7。

表 7 智能网关向协调器发送信息的数据格式

Header	Device_Id	Data	Reserve
2 Byte	1 Byte	1 Byte	4 Byte

如上表 7 所示，首部为固定两字节，0xFAFB；Device_Id 指定目标设备编号，比如 0x00 表示温湿度传感器，0x01 表示烟雾传感器，0x02 表示人体红外传感器；数据内容部

分为 1 字节即一条命令；以及最后面的 4 字节的保留位，暂时默认为 0，以供后期扩展。例如，0xFAFB020100000000 表示，向人体红外传感器发送“开始采集”的命令。

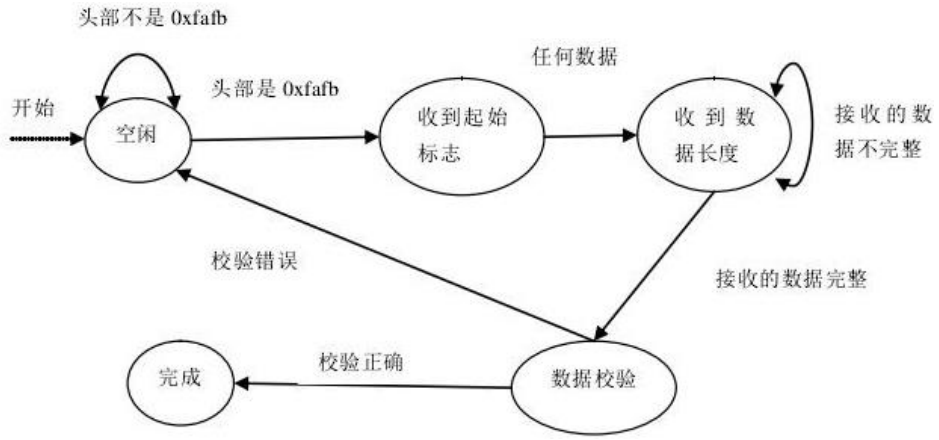


图 15　协调器与智能网关通信数据时序图(下行方向)

3.3.3 智能网关与远程访问设备

本系统的智能网关和远程访问设备（APP）均基于 Android 平台，故通信协议为 HTTP 协议，而主流通讯的数据格式可用 XML、JSON 等等。兼顾效率和与开发难度，采取 JSON 作为两者交互的数据格式。比如，APP 向协调器发送一条命令：{‘object’,‘fog’;‘action’,‘get’}，智能网关返回一条信息：{‘fog’,‘20’;‘result’,‘ok’}。

4 控制模块

4.1 智能网关简介

4.1.1 ARM 特点

当下物联网行业方兴未艾，其中物联网中最为核心的部分是 MicroCPU：源自英国的 ARM 微处理器采用 RISC 架构，而支撑 ARM 占据微处理器半壁河山的重大原因在于它的一系列特点，简介如下表 8。

表 8　ARM 微处理器特点

特点
指令均为固定长度，大量使用流水技术，大量采用单周期指令
大量使用寄存器，除了 Load/Store 指令访存，大部分数据操作在寄存器
寻址方式灵活，执行效率高
高性能、低成本、体积小、功耗低
支持 ARM(32bit)/Thunb(16bit)双指令集，向下兼容 16/8 位部件
采取指令执行结果预判机制，提高后续指令执行效率

4.1.2 ARM 体系结构简介

(1) 工作状态：ARM 通常有两种工作模式：对于 Thumb 模式，处理器执行 16bit 半字对齐的 Thumb 指令；对于 ARM 模式，处理器执行 32bit 字对齐的 ARM 指令；ARM 处理器可以在程序执行的任何时候随意切换在两种模式之中，而切换操作的激发主要由操作数寄存器的状态位决定，当这一个 Bit 为 0 进入 ARM 状态，为 1 进入 Thumb 状态[9]。

(2) 运行模式：目前的 ARM MicroCPU 具备 7 种运行模式，如下表 9。运行模式的切换既可以是软件实现，也可以是中断或异常导致。除了用户模式之外的其他模式，常常用于处理中断、异常以及某些需要保护的系统资源。ARM 处理器正常执行过程与模式切换可见下图 16[9]。

(3) 存储格式：ARM-MicroCPU 的字长为 32bit，从 0 字节开始。0,1,2,3 存放第一个字；4,5,6,7 存放第二个字，诸如此类。它最大的地址空间为 2^{32}=4GB。ARM 内部有大端模式和小端模式这两种数据存放方式。大端模式的特点是：“数据的高部分放在低址空间”，小端模式恰相反[9]。

(4) 寄存器组织：ARM-MicroCPU 具有 31 个通用寄存器和 6 个状态寄存器，不能同时被访问。而某时刻某寄存器是否被编程使用取决于当时的 CPU 运行模式以及运行状态，每一种模式下都有一组寄存器与之对应。任何时候，状态寄存器、程序计数器 PC、通用寄存器 R0~R4 都是可以访问的，不受模式的限制[9]。

(5) 指令系统：除 Load/Store 用于访存，其余指令的执行均在寄存器中完成。本系统仅讨论 ARM 类指令。分类：根据操作数分为定点数和浮点数；条件域：任何一条指令包含 4bit 的条件码，均要按照 CPSR 中条件码的状态和指令自身的条件域有选择的执行；寻址方式：依据操作数所处位置从大类上可分为：立即数、寄存器类、内存类。

表 9　ARM 处理器的 Runting Mode

MicroCPU 运行模式	Description
系统模式（System）	运行 VIP 特权的系统任务
未定义的终止模式（Undefined）	Support 硬件协处理器的软件 Simulation
数据访问终止模式（Abort）	虚拟存储和虚拟保护
管理模式（Supervisor）	系统的保护模式
外部中断模式（IRQ）	通用的中断处理
快速中断模式（FIQ）	高数的数据处理/通道处理
用户模式（User）	一般的正常状态

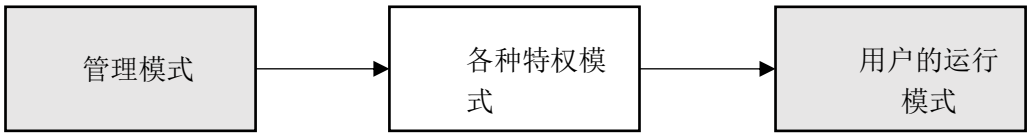


图 16　模式切换

4.1.3 开发平台简介

Exynos4412 这款处理器采用 RISC 结构，采用了 ARM920T 的内核，具有计算能力强功耗低的特点。具有以下的优秀特征：所有的存储器 bank 都是可编程的操作周期；支持各类型号的 ROM 引导(FLASH/EEPROM 等)；支持 NAMD FLASH 存储器直接启动并在启动后 NAMD 存储器直接被当作外存使用；具有更强大的 Cache 存储器；支持普通、DMA 和中断模式的数据传输模式；兼容支持 SD 卡协议（version 1.0）以及 MMC 多媒体卡协议（version 2.11）[9]。本系统将在 Exynos4412 的基础上进行

Linux 系统移植和嵌入式服务的搭建，以及进一步的实现监控功能。本系统采用的是 GA-S214xA90 实验箱作为参考研究平台，该开发板基于 Exynos4412 微处理器，具备完整的软硬件资源，它的简介如下表 10。

表 10 实验箱的软硬件资源

(1)CPU 处理器：Samsung Exyos4412A，主频 400MHZ，可适当超频
(2)SDRAM：在板 64M SDRAM，32bit 数据总线，SDRAM 的时钟的频率为 100MHZ
(3)FLASH 存储: 在板 64M Nand Flash,掉电易失，BIOS 已安装
(4)接口资源：1 个 RJ45 接口；3 个串行口；一个 SD 卡存储接口；以及其他接口
(5)系统时钟：12MHZ 无缘晶振
(6)扩展接口：34pin GPIO 接口和 44pin 2.0mm 系统总线接口，各一个
(7)LCD 显示屏
(8)操作系统支持：Linux2.6.13

4.1.4 远程客户端

考虑到监护系统用户的年龄以及移动互联时代的特点，本系统采用基于 Android 平台的安卓 APP 作为远程访问的客户端。用户可以使用手机上的 APP 对采集系统发起采集命令，也可以访问智能网关上的嵌入式服务器 SQLITE，进而实时获得采集的数据，而且第三方用户也可以通过 APP 远程访问智能网关，对被监护人进行远程监护。

4.2 Linux 系统及移植到 Exynos4412

4.2.1 Linux 移植简介

嵌入式 OS 的发展经历了至少 4 个阶段：首先是无操作系统的嵌入算法阶段，之后是基于芯片的可编程控制器阶段，再之后是基于嵌入式 MicroCPU 的简单操作系统阶段，再到现在的以嵌入式 OS 为核心的嵌入式开发阶段。32 位的 ARM-MicroCPU 与 Linux 的结合也是一大高效搭配。本系统基于 GA-S214xA90 平台，相继移植了与 Exynos4412A 相匹配的 BootLoder、Linux-kernel、以及相关配置文件。并为本开发板的各传感器编写了配套的驱动程序。

4.2.2 移植过程

(1) 建立交叉开发环境

1) 开发主机和目标主机的连接：本系统采用串口通信的方式，使用 Windows 超级终端进行文件传输。
2) 安装 Linux 操作系统：首先在 PC 上安装 VMware 虚拟机，然后在其中安装 Ubuntu 13.04 操作系统，将其作为开发主机，在其上安装开发工具、编辑和编译各类程序，包括 BootLoder 和 Kernel 以及 File System。然后通过传输工具传输至目标平台调试运行。
3) 交叉编译工具：考虑到 Intel X86 架构下的编译文件无法在 ARM 结构下正常运行，这里必须使用交叉编译工具链来对程序进行编译。下载好 ARM-Linux 交叉编译链工具包后安装在上一步创建好的 Ubuntu 操作系统中，并完成环境变量及其相关配置[10]。
4) 主机交叉环境配置：接下来还需要配置 Windows 超级终端的配置、主机的 TFTP7 配置、NFS 服务。

(2) BootLoder 及其移植

1) 介绍：BootLoder 先于 Kernel 被访问，是系统启动的第一个程序段，负责对硬件的初始化、以及后续启动 Kernel。考虑到 BootLoder 对于不同处理器以及设备配置的敏感性，本系统综合实验箱硬件配置和各类 BootLoder 特点，本系统将采用 U-BootLoder 作为模板进行定制。
2) 操作模式：目前的 BootLoder 支持本地和远程两种模式，第一类（Boot-Loding）从目标板的固定存储区将 OS 加载至内存；第二类（Downloading）将借助目标板上的串口或者 RJ45 网络接口从远程下载系统文件，被载入内存，之后再存储至固态存储区[10]。
3) 启动方式：目前 BootLoder 引导方式有三种：闪存 FLASH、HardDisk 硬盘、Internet 网络远程启动。每次硬件上电后，CPU 从 reset 地址开始加载，然后转自 BootLoder，进而转向 Kernel，启动整个软硬件环境[10]。
4) 启动过程：BootLoder 的任务分为两个阶段。首先是设备初始化，这部分的代码采用汇编语言（与 CPU 体系结构关系密切，故而使用汇编），这一部分的主要工作如下表 11 所示[10]。

表 11　BootLoder 的第一阶段

任务	描述
硬件设备初始化	包括设置 CPU 时钟、中断屏蔽、RAM 初始化、初始化测试灯、数据缓存、关 CPU 内部指令
为下阶段载入 BootLoder 准备 RAM 空间	代码加载至 RAM 可获得更快的执行速度
Copy 下阶段的代码到 RAM 空间	Copy 之前必须知道 RAM 的起始地址
设置堆栈	设置堆栈的生长方向（即 SP 的值）
程序跳转到第二阶段的入口处	执行 JMP 指令

其次是初始化更多硬件设备以及加载 Kernel，这部分的代码使用 C 语言编写，兼顾代码可维护性和高效性。其主要任务如下表 12 所示。

表 12　BootLoder 的第二阶段

任务	描述
初始化本阶段的所需硬件	比如初始化 Timer/串口
检测 System 的 Mermory Map	确保有足够的系统 RAM 空间可用
Load 根文件系统和 Kernel 的映像文件至 RAM 空间	
给 Kernel 设置好 Boot 参数	诸如 CORE、MEM、RAMDISK 等
Call Kernel	直接 JMP 到 Kernel 的第一条指令位置

5) 编译与移植：首先，下载好 U-Boot 做编译 Test，下载最新版 U-Boot，找到 Makefile 文件，修改与 Exynos4412A 相关的配置文件。以及在/board 目录下建立实验板的目录，并修改配置文件，测试编译。如果通过，才开始下一步的定制；接下来，依据 Exynos4412 处理器的特点修改 U-Boot：各项修改如下表 13。

表 13　U-Bootz 修改配置信息

修改位置	详细描述
/cpu/arm920t/start.S	根据 2440A 的 datasheet 修改配置
board/friendlyarm/qq2440	加入对 NAND 的 read()
board/friendlyarm/qq2440/Makefile	OBJS:=qq2440.onand_read.oflash.o
include/configs/qq2440.h	添加 NANDFLASHJFFS2USB 启动支持
board/friendlyarm/qq2440/lowlevel_init.S	根据 datasheet 修改
/board/friendlyarm/qq2440/qq2440.c	对 GPIO、PLL 的修改配置
CONFIG_Exynos4412	对 2440 的一些配置

最后，执行编译命令 make clean, make Exynos4412_config，即可生成 u-boot.bin 文件，即为下一步写入开发板的文件[10]。

6) 安装 BootLoder 到实验板：直接使用 JTAG 板直接写入开发板的 FLASH 中。

(3) Linux-Kernel 及 Root-File-System 移植

1) Linux2.6-Kernel 新添加的特性：首先，使用了更优秀的调度算法，更适应高负载和多处理器的应用场景；其次，启用了内核可抢占的 patch，减少了用户干预，减少了某些程序的调度 Delay；最后，创建了统一的设备模型，通过数据结构定义的方式支持更多类型的设备，改进了原本较弱的 DeviceManage[10]。

2) Linux-Kernel 的代码结构如下表 14。

表 14 Linux-Kernel 文件结构

目录	描述
/arch	存放与 CPU 体系结构相关的文件
/drivers	各类设备驱动文件
/Documents	Kernel 文档
/FS	各种文件系统相关
/include	系统的库函数、头文件
/init	系统初始化相关
/ipc	各进程之间进行通信
/kernel	系统内核相关
/lib	核心的库函数
/mm	内存管理
/net	网络协议等网络相关文件
/script	各种脚本文件

3) Linux-Kernel 启动流程：主要流程见下图 17。

4) Kernel 配置系统：Linux-Kernel 内部集成了一个配置 System，可以方便的开启图形界面进行有关 ARM 平台相关的配置。配置系统包含了三部分：Makefile、Cconfig-Tool、KCconfig[10]。

5) Makefile 的组织结构：Linux-Kernel 的 Makefile 较为复杂，可分为五个方面：Kernel_config 相关、/Arch/Makefile、根目录下的 Makefile、内核编译级别下的 Makefile、/Scripts/Makefile 等。进行 Make 编译时，从根目录的 Makefile 开始自顶向下，根据各层的 config 文件逐层编译[10]。

6) 配置编译内核：首先，选择参考板：考虑到移植原则，要求处理器、驱动程序及外围电路的相似性，本系统参考了试验箱自带的 smdk2440 的 BSP 代码。其次，经编译测试和代码分析后，本系统采用 Kernel 自带的配置文件对内核进行编译。并进入配置菜单，根据需要对某些选项进行定制。

7) Root File System：首先，目录结构：Linux 根文件系统包含必要的库函数以及用户级软件，目录结构如下表 15。

表 15 Linux 文件系统目录

目录	描述
/dev	设备文件
/root	系统根目录
/usr	一般存放程序和命令文件
/home	用户的家目录
/proc	系统内存的映射文件
/bin	存放可执行文件
/etc	存放系统的配置文件
/boot	与启动相关文件
/lib	程序的标准库函数
/mnt	设备挂载目录

其次，嵌入式文件系统：目前的嵌入式文件系统架构如下图 18 所示。考虑本试验箱的存储介质为 FLASH 闪存，故本系统采用 Yaffs-2 文件系统；最后，FileSystem img 制作：下载好 mkyaffsimage 工具，解压。并将所有系统相关文件集中在一个文件夹 D 中，执行编译命令 mkyaffsimage 命令，之后就生成了镜像文件[10]。

8) 安装 Linux-Kernel 及其 Root-File-System：设置开发板的 Boot-Mode 为 FLASH，连接串口，打开 Windows 超级终端，平台接电并选择 Kernel-Download 模式，进行一些必要的配置，即可开始下载。RFS 的下载安装过程同 Linux-Kernel，在此不再赘述。

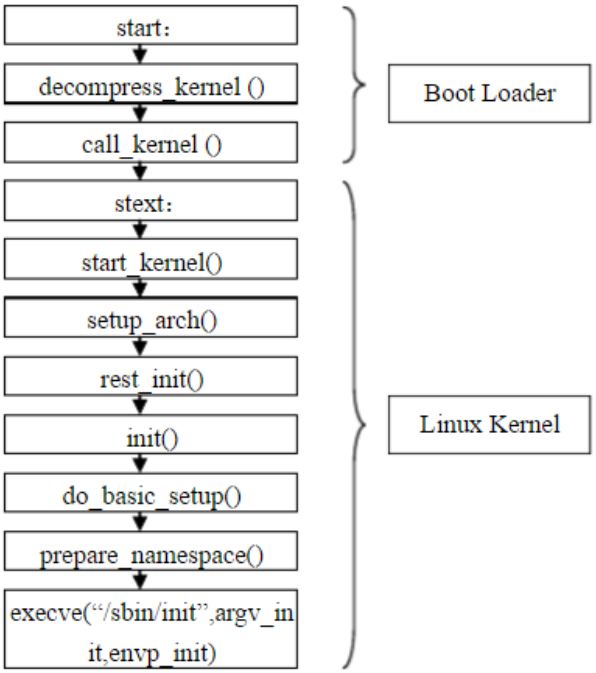


图 17　Linux 内核启动流程图

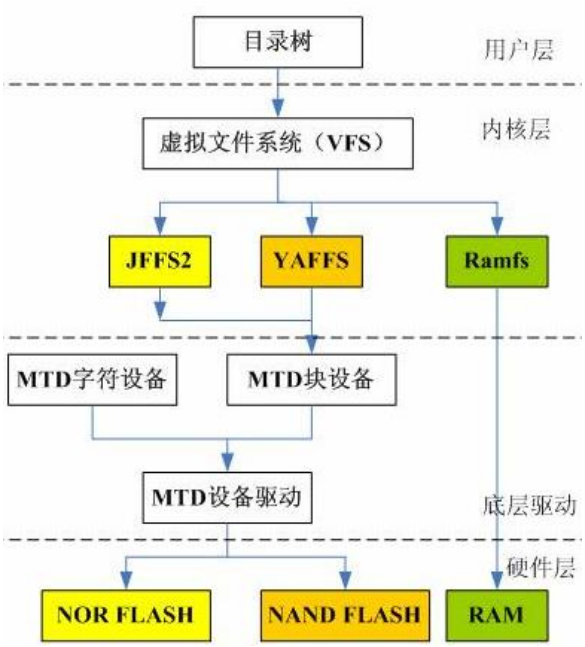


图 18　嵌入式文件系统结构

4.3 嵌入式服务器 SQLITE 移植到 Exynos4412

4.3.1 SQLITE 简介

(1) 目前基于 Linux 的嵌入式数据库种类繁多， 其中 SQLITE 支持大部分的 SQL 标准，速度快，体积小，源码开放，考虑到容量和效率，本系统采用 SQLITE。

(2) SQLITE 由以下几个组件组成：内核、SQL 编译器、后端以及附件，SQLITE 的体系结构图如下图 19。

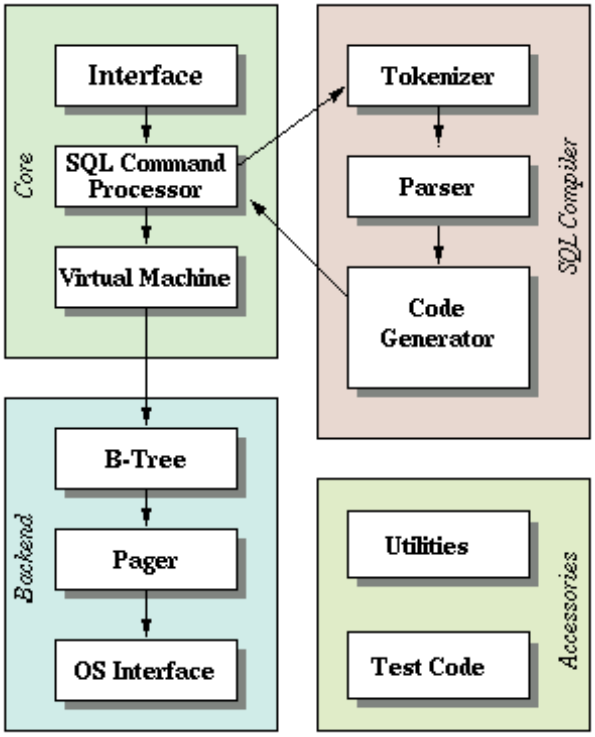


图 19　SQLITE 体系结构

4.3.2 移植过程

(1) 下载 SQLITE 3 源码。

(2) 解压缩，并修改配置文件，主要是指定 host(交叉编译工具)、prefix(安装目录)。

(3) 编译，接下来使用交叉编译工具 strip 清除调试信息，并重新压缩。

(4) 参照之前下载内核的方式，将修改后的源码压缩包下载至 ARM 实验板。

(5) 在开发板上执行 SQLITE 命令，测试 SQLITE。

5 系统测试与实现

5.1 电路设计及平台环境搭建

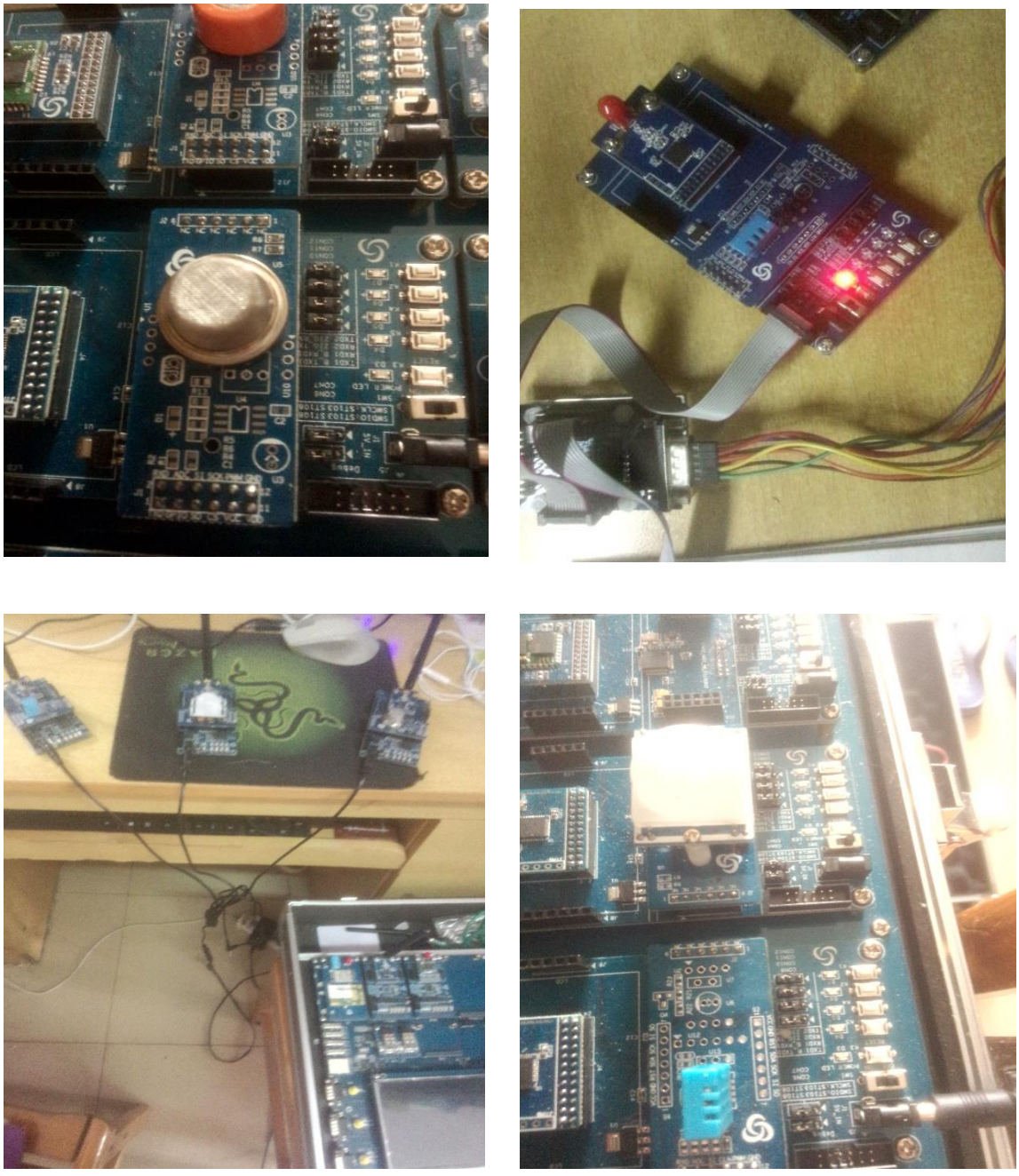


图 20　实验硬件环境搭建

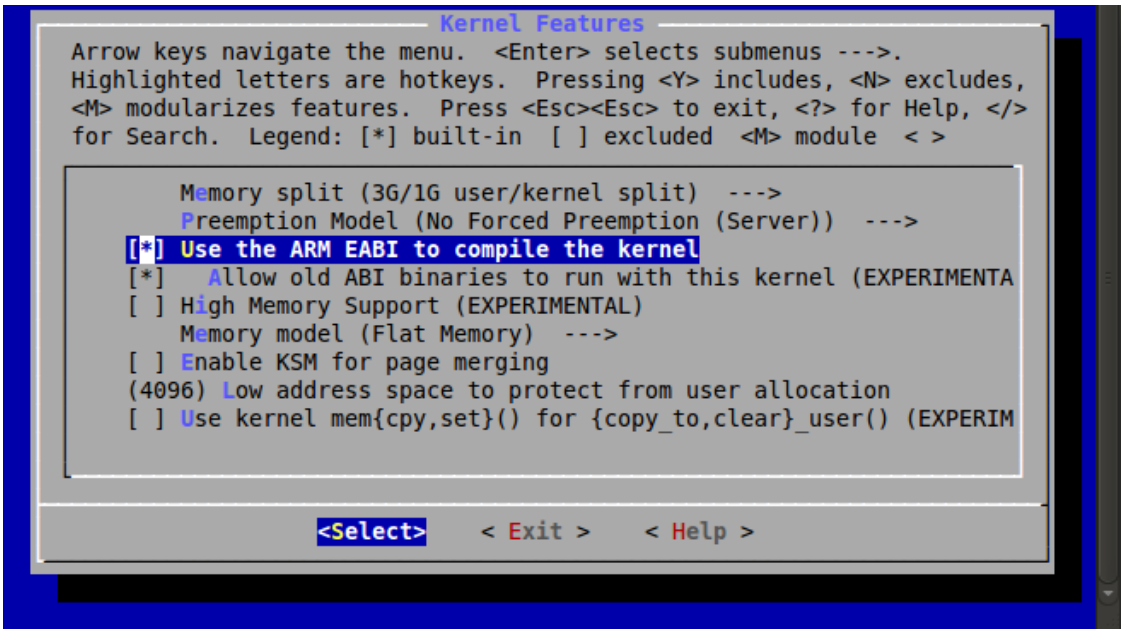


图 21　LINUX 移植到 Exynos4412

5.2 程序设计

5.2.1 智能网关程序设计

首先，智能网关至少要提供两个服务：与协调器进行串口通信，接收并存储来自协调器的数据包并解析；其次要能支持和远程客户端的 HTPP 连接和网页服务。故而本系统的智能网关的内部结构如下图 22。

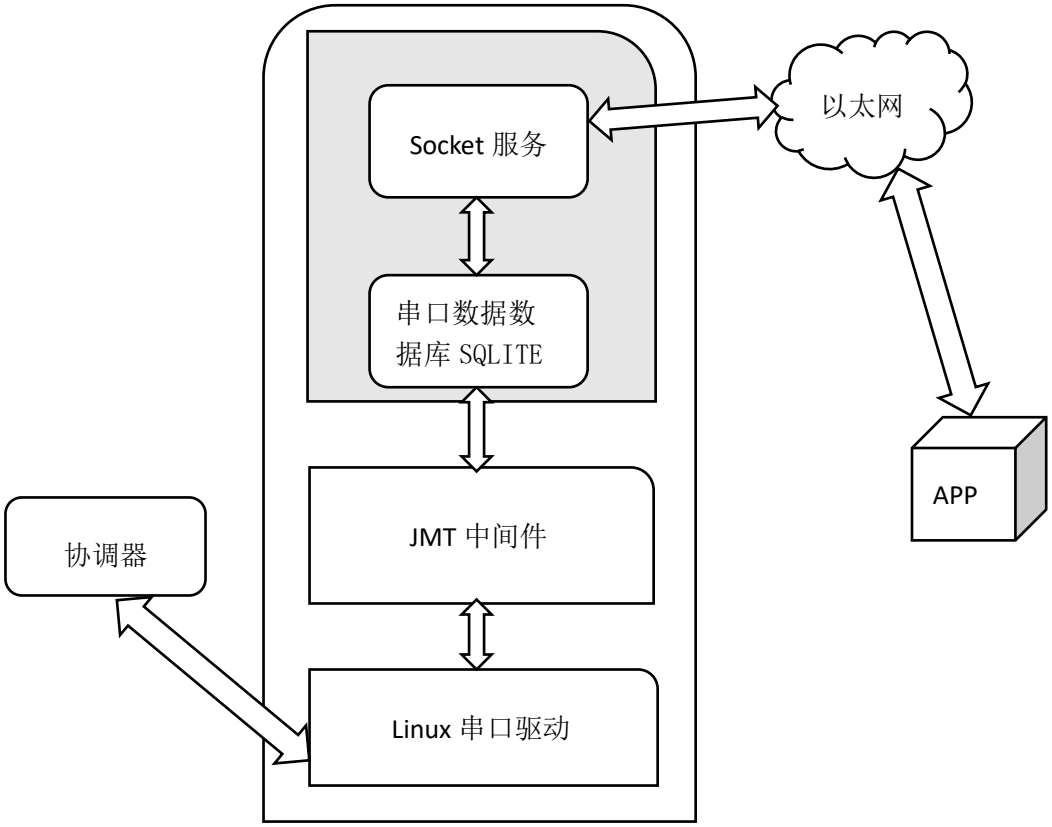


图 22　智能网关内部结构

网关实现的功能有两项：首先是通过串口实时监听并接收来自协调器的数据包并解析之后存放在 SQLITE 数据库；提供 Socket 服务，提供实时的网页文件输出，以供远程 APP 实时查看传感器数据；其次是反馈控制功能。实现方式是通过在 APP 层编写 ZIGBEEMain.java 文件，创建两个进程：一个是 UARTServer，一个是 SocketServer，具体核心代码如下：

```
//主类，创建两个子进程 UARTServer，SocketServer
public void onCreate() {
super.onCreate();
// 启动 UARTServer 线程
UARTServer.start();
// 启动 SocketServer 线程
mSocketServerThread = new SocketServerThread();
mSocketServerThread.setDaemon(true);
mSocketServerThread.start();
}

//子进程 UARTServer
class ZBSerial extends Thread {
boolean mRunning = false;
SerialPort mSerialPort;
public void start() {
    mRunning = true;
    super.start();
}

public void exit() {
    mRunning = false;
    if (mSerialPort != null) {
    mSerialPort.close();
    } try {
    this.join();
    } catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();}
}

// 读取串口数据，在指定缓冲区的某起始位置读取 N 个字节的数据
int serial_read(byte[] buffer, int off, int cnt) throws IOException {
    int len;
    while ((len = mSerialPort.getInputStream().available()) == 0) {
    // this.wait(10);
    }
    if (len < 0)    return -1;
    if (len > cnt) len = cnt;
    return mSerialPort.getInputStream().read(buffer, off, len);
}
```

```
// 线程运行主体
public void run() {
    byte[] buffer = new byte[256];
    Log.d(TAG, "serial thread start.....");
    while (mRunning) {
    if (mSerialPort == null)
    { // 打开串口
        try {
            mSerialPort = new SerialPort(new File("/dev/s3c2410_serial3"), 38400);
        } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();}
    } else { //串口已经打开
            int ret;
            try {
                ret = serial_read(buffer, 0, 1);
                //读取失败
                if (ret < 0) {
                    mSerialPort = null;
                    continue;
                }
                if ((buffer[0] & 0xff) != 0xFE)      continue;
                ret = serial_read(buffer, 1, 1);
                if (ret < 0) {
                    mSerialPort = null;
                    continue;
                }
                int off = 2, pkglen = 1 + 1 + 2 + (buffer[1] & 0xff) + 1;
                while (off < pkglen) {
                ret = serial_read(buffer, off, pkglen - off);
                if (ret < 0) {
                    mSerialPort = null;
                    continue;
                }
                off += ret;
                }
                //将串口接收到的数据进行处理
                onSerialPackage(buffer, pkglen);
            }
```

首先主进程启动,接下来初始化串口、启动网络串口，分别创建两个子进程 UARTServer 和 SocketServer。UARTServer 主要工作是开启串口、取出缓冲区数据，

接下来调用数据解析函数 OnSerialPackage()对取得数据进行解析并上传至 SQLite 数据库；SocketServer 实现的功能有开启 TCP 连接、准备好数据，等待客户端访问。上述两项功能的程序流程图如下图 23。

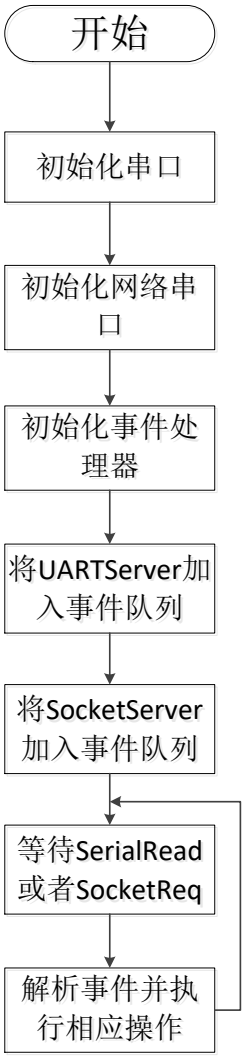


图 23　智能网关程序流程图

5.2.2 ZIGBEE 程序设计

考虑到通信距离不太远，目前暂时忽略路由器节点，只关注采集结点和协调器。传感器节点负责采集三种信号，因此本部分的程序设计包括传感器的程序设计、协调节点的程序设计。此外，本系统采用了 TI 公司的 Z-STACK 协议栈开发，ZSTACK 封装了一款 mini 系统 OSAL 以及一系列库函数，使用 ZSTACK 的框架可以方便地搭建一个基于 ZIGBEE 协议的应用程序。

(1) 传感器的程序设计

传感器部分要实现的功能细分为：入网、采集数据、发送数据到协调器。因为本系统底层基于 ZSTACK 协议栈，故涉及网络部分可根据 ZSTACK 的事件机制，与传感器无关，可重用。需要为每个传感器单独处理采集程序。总体的程序流程图如下图 24。

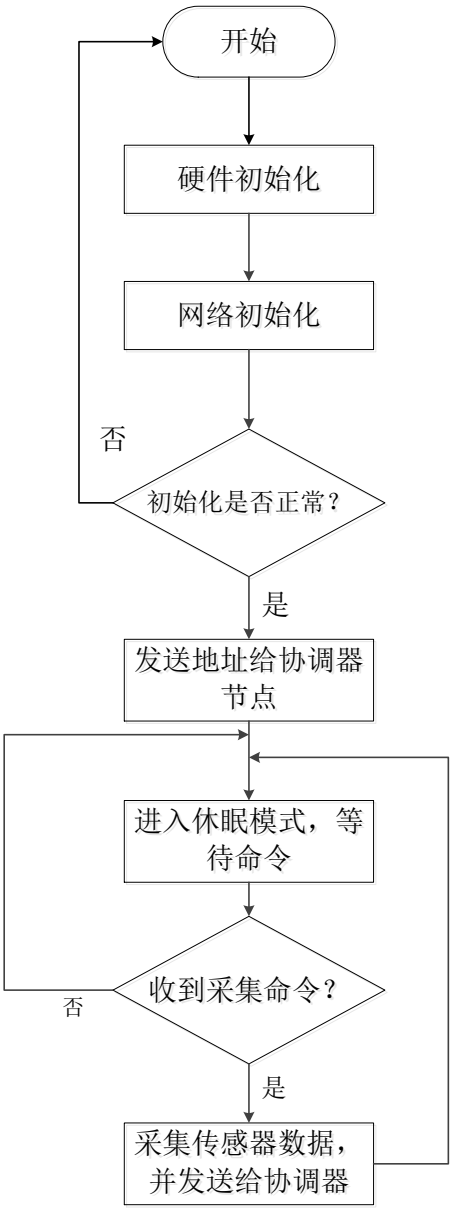


图 24　传感器程序流程图

具体到每个传感器的采集部分的原理已在第 2 章详细描述，在此仅展示采集程序的部分核心代码：

1) 温湿度采集

```
///*  初始化温度传感器  */
void dht11_io_init(void)
{
    P0SEL &= ~0x20; //P1  为普通  I/O  口
    COM_OUT;
    COM_SET;
}

/*  更新数值  */
void dht11_update(void)
{
    int flag = 1;
    unsigned char dat1, dat2, dat3, dat4, dat5, ck;
    //主机拉低  18ms
    COM_CLR;
    halWait(18);
    COM_SET;
    flag = 0;
    while (COM_R && ++flag);
    if (flag == 0) return;
    //总线由上拉电阻拉高  主机延时  20us
    //主机设为输入  判断从机响应信号
    flag = 0;while (!COM_R && ++flag);
    if (flag == 0) return;
    flag = 0;
    while (COM_R && ++flag);
    if (flag == 0) return;
    dat1 = dht11_read_byte();
    dat2 = dht11_read_byte();
    dat3 = dht11_read_byte();
    dat4 = dht11_read_byte();
    dat5 = dht11_read_byte();
    ck = dat1 + dat2 + dat3 + dat4;
    if (ck == dat5)
    {
        sTemp = dat3;
        sHumidity = dat1;
    }
    //printf("湿度: %u%%  温度: %u℃ \r\n", dat1, dat3);
```

```
}
```

温湿度 DHT11 部分采集过程大概为：初始化 I/O 口、拉低总线电平启动采集、延时等待数据、采集结束拉高总线，详细原理见第二章。

2) 有害气体采集

```
//void CombustibleGas_Test(void)
{
    char StrAdc[10];
    int AdcValue;
    AdcValue = getADC(); //得到 ADC 转换的值
    sprintf(StrAdc,"%d\r\n",AdcValue);
    Uart_Send_String(StrAdc); //串口发送数据
    halWait(250); //延时
    D7=!D7; //标志发送状态
    halWait(250);
    halWait(250);
}
/*得到 ADC 值----------------------------------------------------*/
int getADC(void)
{
unsigned int value;
    P0SEL |= 0x02;
    ADCCON3 = (0xB1); //选择 AVDD5 为参考电压； 12 分辨率； P0_1 ADC
    ADCCON1 |= 0x30; //选择 ADC 的启动模式为手动
    ADCCON1 |= 0x40; //启动 AD 转化
    while(!(ADCCON1 & 0x80)); //等待 ADC 转化结束
    value = ADCL >> 2;
    value |= (ADCH << 6); //取得最终转化结果，存入 value 中
    return ((value) >> 2);
}
```

有害气体检测模块主要过程为：设置端口、选择参考电压和启动模式、读取数据、进行 AD 转换、启动串口发送数据至 CC2530，详细原理见第二章。

3) 人体红外检测

```
// /*主函数-------------------------------------*/
void main(void)
{
    xtal_init(); //初始化系统时钟
    led_init(); //初始化 LED
```

```
    uart0_init(0x00, 0x00); //初始化串口
    P0SEL &= ~0x20; //P0_5 为普通 io 口
    P0DIR &= ~0x20; //P0_5 输入
    while(1)
    {
        Infrared_Test(); //人体红外检测
    }
}
/*Infrared_Test 函数----------------------------*/
void Infrared_Test(void)
{
    char Str[10];- 106 -
    int Value;
    Value = P0_5; //P0_5 与 GPIO 相连
    sprintf(Str,"%d\r\n",Value);
    Uart_Send_String(Str); //串口发送数据
    halWait(250); //延时
    D7=!D7; //标志发送状态
    halWait(250);
    halWait(250);

}
```

人体红外 HC-SR501 模块的采集过程大致为：初始化时钟和 LED、初始化串口、设置好 Px 口与 GPIO 相连、读取数据、经串口传输至 CC2530，更多原理详见第二章。

4) 网络相关的部分核心代码（各传感器可复用）

```
void zb_StartConfirm( uint8 status )
    {
        if ( status == ZB_SUCCESS ) //zigbee 协议栈启动成功
        {    myAppState = APP_START;
            HalLedSet( HAL_LED_2, HAL_LED_MODE_ON );
            // 设置定时器事件来触发自定义的 MY_REPORT_EVT 事件
        osal_start_timerEx( sapi_TaskID, MY_REPORT_EVT, REPORT_DELAY );
        else //zigbee 协议栈启动失败重新启动
        {osal_start_timerEx( sapi_TaskID, MY_START_EVT, myStartRetryDelay );}
```

```
    }
void zb_HandleOsalEvent( uint16 event )
{ if (event & ZB_ENTRY_EVENT) { //zigbee 入网事件}
        if ( event & MY_START_EVT )
       { //启动 ZStack 协议栈事件 zb_StartRequest()}
        if (event & MY_REPORT_EVT)
        { // MY_REPORT_EVT 事件触发处理
            myReportData();osal_start_timerEx( sapi_TaskID, MY_REPORT_EVT,
            REPORT_DELAY );}
}
static void myReportData(void)
{
        byte dat[6];
        uint16 sAddr = NLME_GetShortAddr(); //读取本地的网络短地址
        uint16 pAddr = NLME_GetCoordShortAddr(); //读取协调器的网络短地址
        HalLedSet( HAL_LED_1, HAL_LED_MODE_OFF );
        HalLedSet( HAL_LED_1, HAL_LED_MODE_BLINK );
        //数据封装
        dat[1] = (sAddr>>8) & 0xff; //本地网络短地址
        dat[2] = sAddr & 0xff;
        dat[3] = (pAddr>>8) & 0xff; //父节点短地址（协调器短地址）
        dat[4] = pAddr & 0xff;
        dat[5] = MYDEVID; //设备 ID 号
        //将数据包发送给协调器（协调器的地址为 0x0000）
    zb_SendDataRequest(0, ID_CMD_REPORT, 6, dat, 0, AF_ACK_REQUEST, 0 );
}
```

传感器入网过程可分为两个阶段：首先，当 ZSTACK 协议栈启动成功，触发了 zb_StartConfirm()函数，触发本传感器的 zb_HandleOsalEvent()，本函数进而触发了 ZigBee 入网事件、启动 ZSTACK 请求事件、myReportData()事件。本终端节

点顺利入网；其次，在 myReportData()事件中，终端节点获取自己的网络地址、PANID、信道名称以及某些采集数据，把上述数据根据事项规定好的数据帧格式进行封装，启动 zb_SendDataReques()库函数进行发送至协调器，完成一次入网及同协调器的通信。

(2) 协调器的程序设计

协调器节点需要完成的功能如下：创建网络、接收传感器节点的数据并传输至智能网关、接收智能网关的指令对采集结点发送命令。本系统基于 ZSTACK 的库函数，先进行硬件初始化，包括中断的配置、I/O 设置、能量检测、外部设备初始化等[8]。

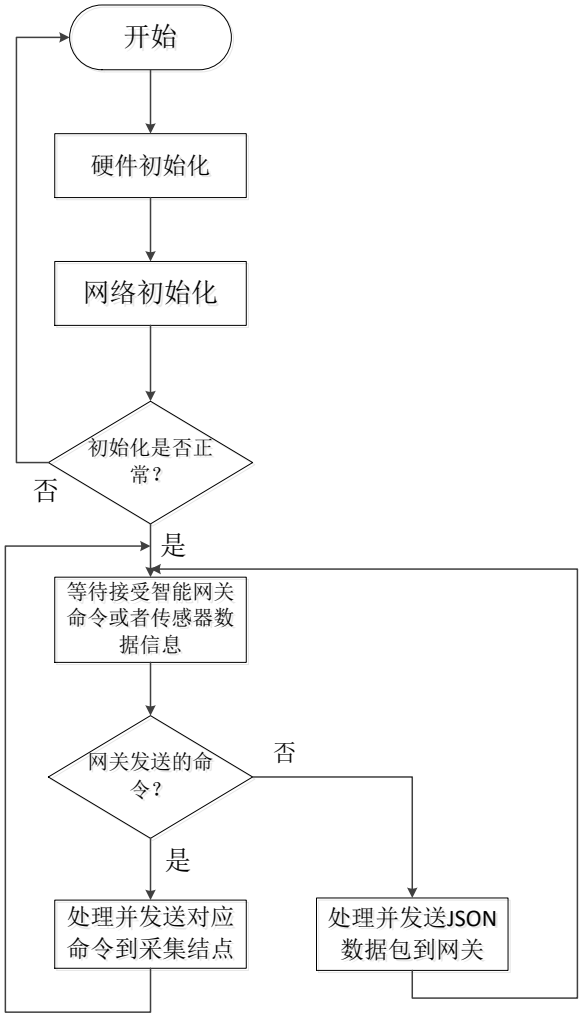


图 25 协调器的程序流程图

接下来进行网络初始化，初始化结束后，就创建了一个新的局域网，接下来协调器节点进入监听状态。监听的内容有两种：一是监听来自智能网关发送的命令，二是来自

传感器节点采集到的数据信息。基于 Z-Stack 协议栈的库函数，只要协调器监听到有新的数据包来到，OSAL 层会产生一个响应事件，所以仅需要在协调器程序中添加对该事件的响应函数即可。参考第三章设计的协议，本系统使用 DeviceID 对协调器接收的无线消息进行区分，普通传感器和智能网关都有各自的 DeviceID。其中对于 AF_INCOMING_MSG_CMD 事件的响应函数如下：

```
//协调器用户事件解析
void zb_HandleOsalEvent( uint16 event )
{     uint8 startOptions; uint8 logicalType;
     if (event & ZB_ENTRY_EVENT)
     { //处理 zigbee 入口事件
       void zb_ReadConfiguration( ZCD_NV_LOGICAL_TYPE,sizeof
       (uint8),&logicalType );
       if ( logicalType != ZG_DEVICETYPE_COORDINATOR )
           { //设置节点类型为协调器
                logicalType = ZG_DEVICETYPE_COORDINATOR;//将节点类型写 NV
      zb_WriteConfiguration(ZCD_NV_LOGICAL_TYPE, sizeof(uint8), &logicalType);}
      zb_ReadConfiguration( ZCD_NV_STARTUP_OPTION, &startOptions );
      if (startOptions != ZCD_STARTOPT_AUTO_START)
      {    startOptions = ZCD_STARTOPT_AUTO_START;
           zb_WriteConfiguration(ZCD_NV_STARTUP_OPTION,sizeof(uint8)    );
        }
     //入口事件一直在触发，则表明 zigbee 网络正在建立，就闪烁 LED 灯
      HalLedSet( HAL_LED_2, HAL_LED_MODE_OFF );
      HalLedSet( HAL_LED_2, HAL_LED_MODE_FLASH );
     }
}
//接收传感器发送的环境数据
void AF_INCOMING_MSG_CMD _Action(af Incoming MSGPacket_t * pkt)
{
     switch ( pkt->cluster Id )
     {
        case Sensor_Info:
             get Sensor Info(); //
             Hal UARTWrite(0,buffer,5);//  把数据通过串口发送给智能网关
             break;
```

```
        case Node_Info:
            get Node Info();//获取终端节点信息
            break;
    }
}
```

协调器首先在入口事件中将自身类型设置为协调器，进而进行一系列的网络配置，并且开始建立 ZigBee 网络，建立成功后将发送信号给其他节点，以供其他节点入网。此外，响应函数使得协调器监听串口缓冲区，一旦有数据更新，取出传感器发送过来的数据包，并开启解析事件，解析数据并二次封装成数据包通过网络发送给智能网关。同时还启动一个守护进程监听来自智能网关的数据包，解析来自 APP 的数据包是否包含命令。区分的依据是数据帧的格式，其定义详见第三章。

5.2.3 客户端 APP 程序设计

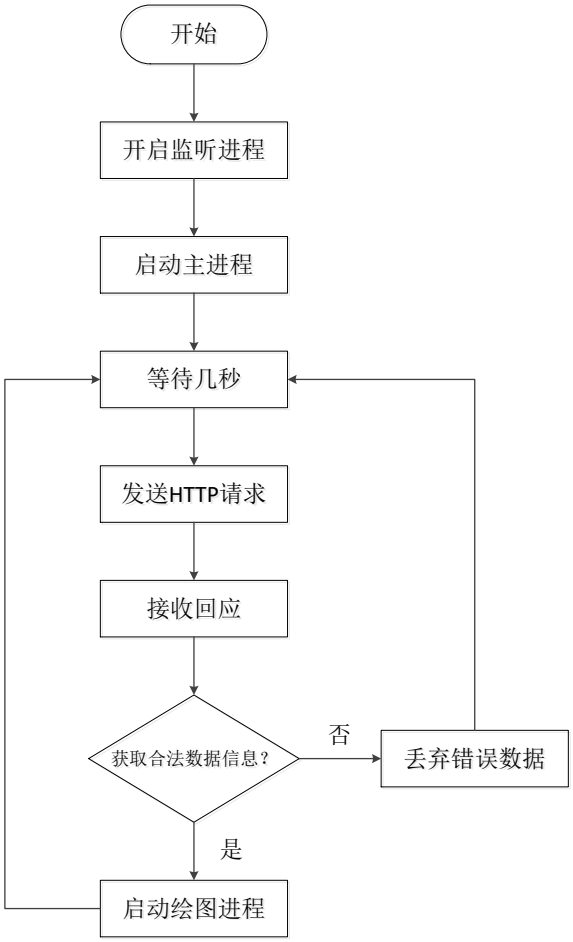


图 26　客户端接收网关的传感器数据并绘图

客户端要完成的功能分两种：首先是发起 HTTP 请求获取传感器采集数据并实时绘图，其次是发送命令对传感器进行控制。第一个功能的实现过程大概是：客户端建立一个进程 T 负责接收来自网关的数据包。首先创建一个线程 A 发送 HTTP 请求，网关的监听进程返回数据包，客户端对获取的数据包进行解析，若合法则开启一个线程 B 进行实时绘图，部分代码展示如下：

```
//连接服务器
private void connect2server()
{
    int port = 8320; //端口号
    String x[] = mZigBeeGetWay.split(":");
    if (x.length==2)
    {
        port = Integer.parseInt(x[1]); //类型转换
        if (port == 0) port = 8320;
    }
    mConnectStatus = 1; //连接状态标志位
    mZbThread.requestConnect(x[0], port); //响应连接
    setTitle("正在连接到 ZigBee 网关 -- " + this.mZigBeeGetWay);
    setProgressBarIndeterminateVisibility(true);
}
//解析网关发送的传感器数据信息
rivate void onResponseMSG_GET_APP_MSG(byte[]dat)
{
    if (dat == null) return;
    Log.d(TAG, "APP MSG :"+Tool.byte2string(dat));
    if (dat == null || dat.length<=4)
    { //数据包发送错误
        Log.d(TAG, "APP MSG timeout or package error.");
        return;
    }
    int addr = Tool.builduInt(dat[0], dat[1]); //地址
    int cmd = Tool.builduInt(dat[2], dat[3]); //命令
    byte[] data = new byte[dat.length-4];
    for (int i=0; i<data.length; i++) data[i] = dat[4+i];
    synchronized(mLock)
    { //异步信号量
        if (mNodePresent != null )
        {
            mNodePresent.procAppMsgData(addr, cmd, data); //调用函数处理数据
```

```
        }
    }
}
```

首先连接网关的 SQLite 服务器，接下来取出数据库的数据进行解析，按照第三章约定的数据帧格式取得数据包中的地址、设备 ID、Data 部分，并将取得的数据域送至下一个处理函数模块，作进一步的处理。

第二个功能，客户端会创建一个进程用于发送命令和接受返回值。首先把命令按照第 3 章约定的协议格式封装成 HTTP 请求，发送给网关，并等待返回结果。流程图如下图 27。

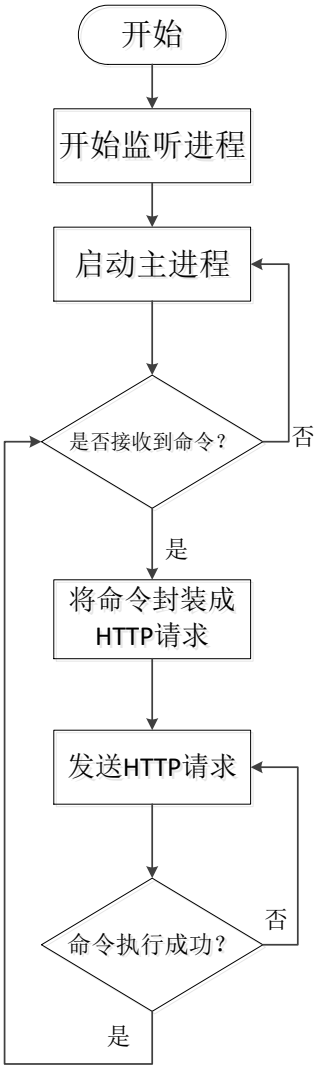


图 27　客户端发送命令给智能网关

部分代码展示：

```
//封装命令，（准备发给给智能网关）
```

```
void procAppMsgData(int addr, int cmd, byte[] dat) {
    int pid;
    int i = -1;
    Log.d(TAG, Tool.byte2string(dat));
    if (cmd == 0x8001 && dat[0] == 0)
    { i = 1; //读参数响应}
    if (cmd == 0x0003)
    { //主动上报传感器的值 i = 0;}
    if (i < 0) return;
    while (i < dat.length)
    {
        pid = Tool.builduInt(dat[i], dat[i + 1]);
        if (pid == 0x4102)
        { //获取网络的状态
            if (dat[i + 2] == 0) {
            mLightImageView1.setImageBitmap(Resource.imageLightOff);
            mLightImageView2.setImageBitmap(Resource.imageLightOff);
            mBtnOn2.setChecked(false);
        } else if (dat[i + 2] == 1)
            {
            mLightImageView1.setImageBitmap(Resource.imageLightOn);
            mLightImageView2.setImageBitmap(Resource.imageLightOff);
            } else if (dat[i + 2] == 2)
                {//设置网络某一节点的信息
                    mLightImageView1.setImageBitmap(Resource.imageLightOff);- 292
-                   mLightImageView2.setImageBitmap(Resource.imageLightOn);
                    mBtnOn2.setChecked(true);
                }
                else
                {
                    mLightImageView1.setImageBitmap(Resource.imageLightOn);
                    mLightImageView2.setImageBitmap(Resource.imageLightOn);
                    mBtnOn1.setChecked(true);
                    mBtnOn2.setChecked(true);
                }
            i += 3;//递归地设置各个节点的网络信息
        }
        else
        {
            return;
        }
    }
```

}

如上例，首先 APP 的主进程启动，封装一个网络设置命令，格式化成 HTTP 请求，通过以太网传输至智能网关，智能网关第一次解析数据包，分析目的地和数据域，进而再次封装数据帧发送至协调器，协调器接受到命令，根据数据帧格式进行二次解析，取出数据域的命令，尝试执行，然后将返回执行的结果封装成数据帧传输至网关，网关经过二次封装成 HTTP 请求返回至用户 APP。中间涉及了复杂的数据帧的格式转换和拆分与合并，在此仅介绍了大致的流程。

5.3 系统运行结果截图

图 28　实物连接

如上图 28，有三个采集节点和一个协调器，智能网关直接使用实验箱的 Exynos4412 处理器。当系统上电，协调器开始组建网络，三个采集节点开始搜索网络并加入 ZigBee 网络，进而开始数据通讯。

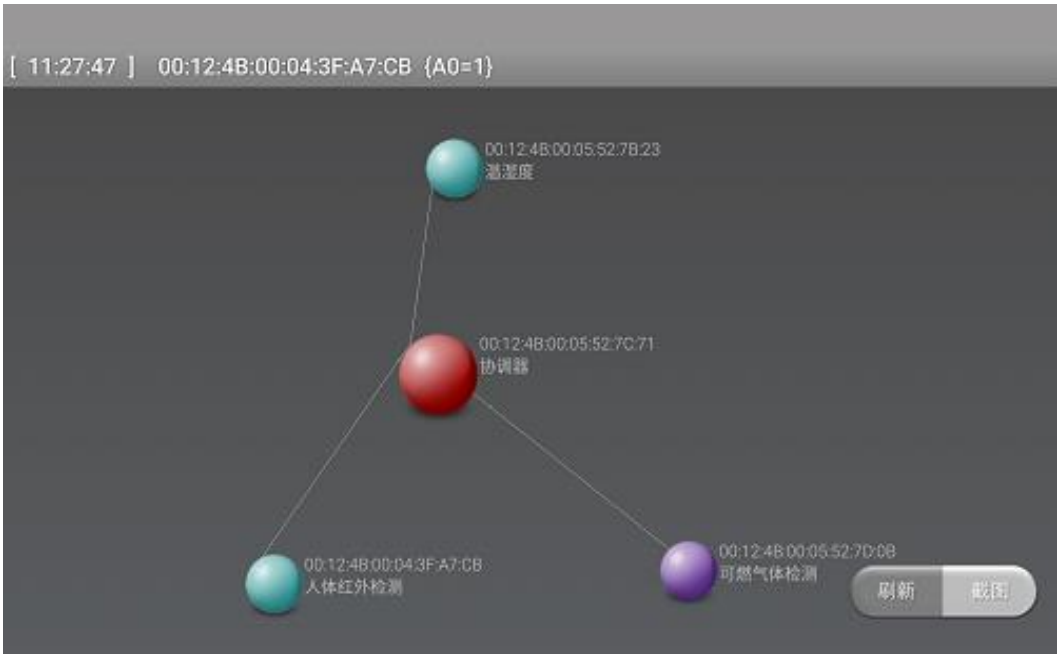


图 29　两次不同测试环境中的系统的网络拓扑图

如上图 29，红色代表协调器，其余分别为终端节点和路由器，本系统为 3 个采集节点。当协调器建立好网络，终端节点搜索网络并加入，用户 APP 通过递归查询的方式向各节点发送消息，根据返回信息建立了系统的拓扑图。点击其中一个节点，可了解更多关于此节点的详细信息。

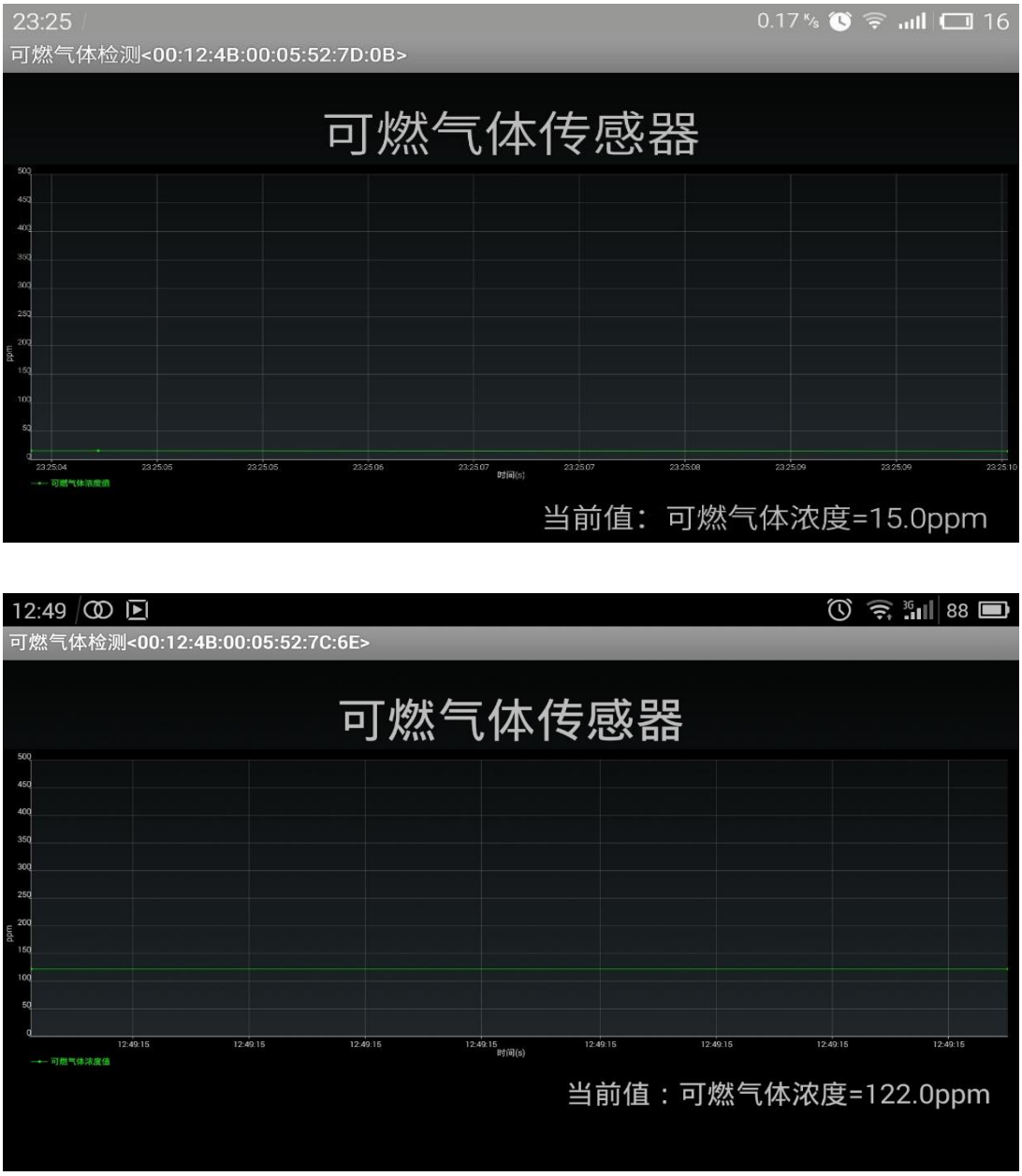


图 30　两次不同环境中采集有害气体指标

本系统在不同环境不同时间下，进行了两次对空气中有害气体的测量实验。第二次相比第一次的数据明显有增大，测试结果表明第二次的环境中的有害气体浓度远高于第一次，说明第二次的测试环境存在一定的安全隐患。这个测试结果与实际情况完全符合，一定程度上证明了本系统在有害气体检测方面的正确性和有效性。

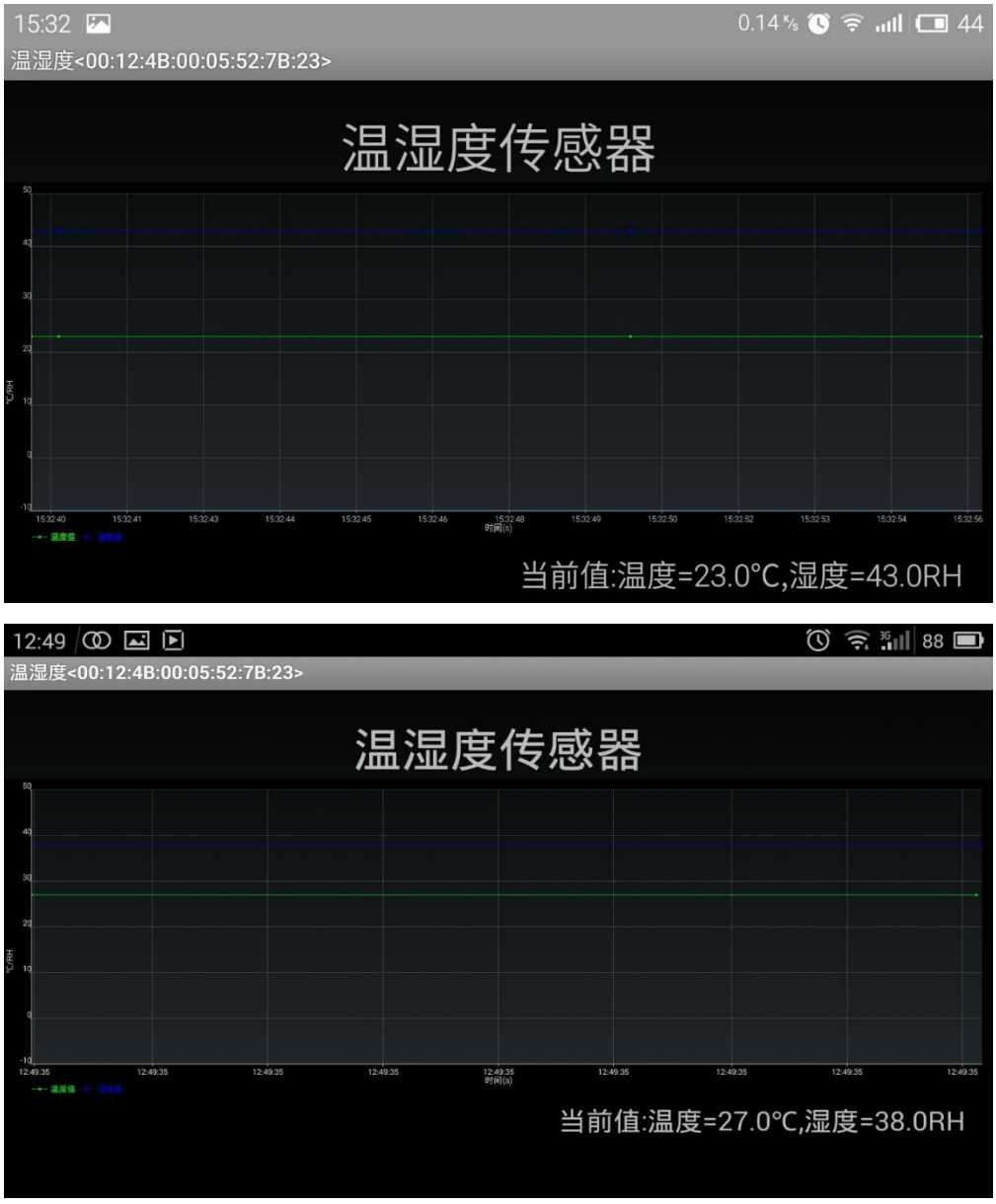
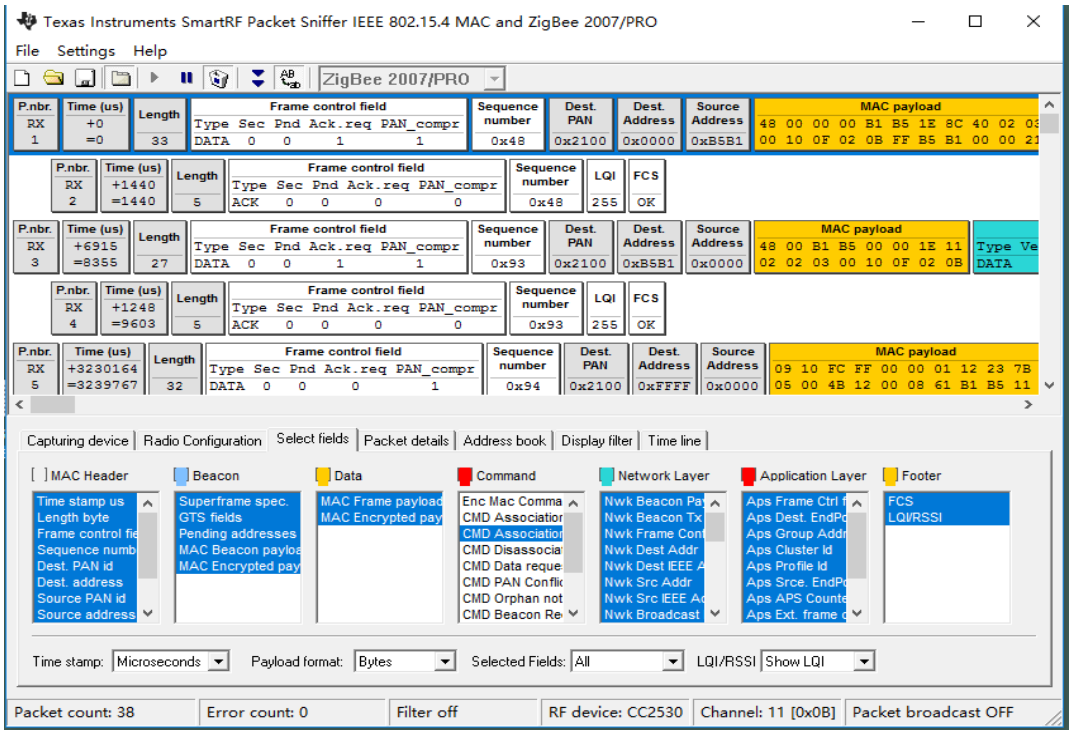


图 31　两次不同环境中采集温湿度指标

同有害气体检测实验，在两次不同环境和时间中，本系统进行了两次对于空气中温湿度信号的测量。结果显示两次差异不大，这与实际结果相符合。从而证明了本系统在空气温湿度测量方面的有效性。老人的某些疾病可能与环境中的温湿度有密切联系，故而检测空气中温湿度参数对于老人居住环境的也具有一定程度上的意义。

图 32　人体红外报警节点的两次采集结果

老人独居在家，晚上容易发生盗贼入室事件。故而在夜晚需要开启人体红外感应模块，用来监控不明外来人员。通过在采集节点编程设置 HC-SR501 模块为夜间开启模式，如果发生陌生人破门而入就发生警报事件。对于老人安全监护具有极其重要的意义。实验的测试结果也证明了本系统的有效性。



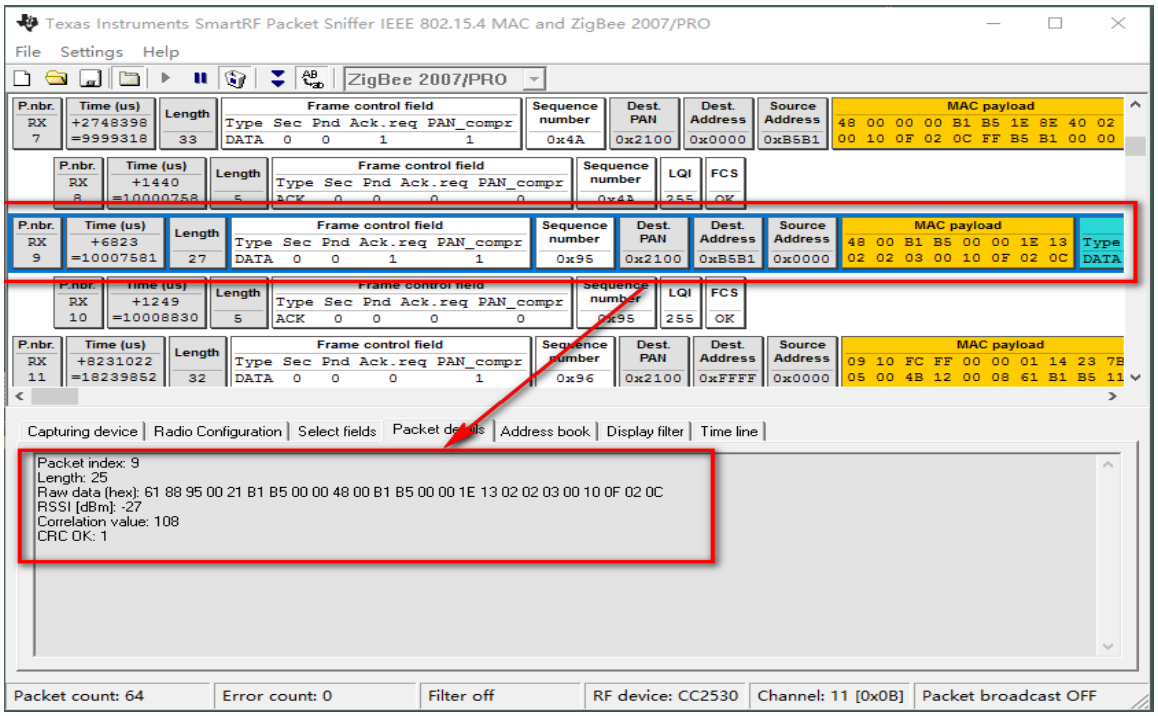


图 33　对采集数据进行抓包分析

数据传输的稳定性和正确性可以通过对传输过程中数据帧进行抓包分析得以验证。本系统在运行过程的某一时刻，通过截获温湿度传感器向协调器发送的某一个数据包，并使用 TI 的数据帧分析工具 SmartRF-Pakage 对此进行分析，第一张图是若干个数据帧的格式解析，大致分为首部、数据域、命令位、地址信息、校验位等。第二张图是对其中一个数据帧的每一部分的详细解析。对于 ZigBee 网络稳定性的测试可借助 TI 提供的更高级的工具软件进行验证，在此不做过多叙述。实验结果表明，数据帧格式和第三章定义的传感器与协调器的格式完全符合，这同时也证明了传输过程的稳定性以及本系统设计的正确性。

图 34　向协调器发送控制命令

通过 APP 向整个 ZigBee 局域网发送命令设置网络参数，比如，修改 PANID、信道名称、节点名称及 ADDR。此功能可方便地随时加入一个网络节点，随时调整网络状态以防止某些意外情况，对于系统的可扩展性及鲁棒性意义重大。实验结果表明，本系统可正确完成 APP 对于 ZigBee 网络结构的调整。

6 系统总结与展望

从半年前开始选题，之后就是查阅文献，和老师同学讨论，逐步确定整个系统要实现的功能，分模块设计实施方案、查找相应的指导资料、确定硬件平台的选择、开展系统涉及技术的学习，并进行软件设计的初步实践。这半年来逐步掌握了嵌入式开发的基本流程，熟悉了 ARM 平台下 Cotex-A9 处理器的体系结构及上层的软件编程，并且在 Exynos4412 的平台下移植了嵌入式 Linux 系统，以及嵌入式服务器 SQLITE，并之后基于以上环境，熟悉了嵌入式 Linux 环境下的驱动程序开发；其次是熟悉了 ZIGBEE 协议，加深了对无线传输网络及协议栈编程的学习，同时初步掌握了基于 ZSTACK 环境的 ZIGBEE 无线网络的搭建与基于 ZSTACK 协议栈的程序设计。再然后逐步掌握了基于安卓平台的手机客户端 APP 的程序设计。基于上述基础，最后综合三大模块，构建了一个无人值守的老人居住环境监护系统。本系统基本实现了：室内环境各项安防参数检测，包括：室内温湿度、有害气体检测、异常人员监测；以及构建了 ZIGBEE 无线网络将各项参数传输至智能网关；之后通过智能网关将数据发送至云端，以供多用户通过手机 APP 实时访问，并通过智能网关对室内环境作出反馈。并且，本系统的可扩展性很高，只需要更换采集模块的传感器，即可轻松实现其他各类监护类系统。比如，仓库监控系统、物流平台监控系统、远程医疗监护系统等等。此外本系统分无线传输模块的鲁棒性也较好，可以在发生网络事故的时候，实时通过手机 APP 设置网络拓扑，以保证本系统在极端情况下依旧可以正常运行。

最后的系统测试表明，本系统基本监护功能已经实现，但不足之处也依然存在。比如老人监护系统最好加入对老人本身的生理健康数据的监测，但由于时间和成本原因，这方面的传感器还没有加入系统，所以在下阶段的学习过程中，希望能加入心率脉搏传感器等增强型传感器；其次，本系统的智能反馈还做得不够，比如只有目前只能通过 APP 修改一些 ZIGBEE 网络参数以及对传感器的基本调节，没有加入继电器之类的器件进行更高级的反馈；这些问题都是下阶段的学习所要解决的。同时也希望可以指导下阶段研究生的学习以及后面的职业发展。

参考文献

[1]刘小钦. 基于 ZigBee 的老年健康监护系统的研究[D].安徽理工大学, 2012.
[2]张瑞. 基于 ZigBee 的人体健康监测系统设计[D].曲阜师范大学, 2014.
[3]钟恬恬. 基于 ZigBee 的生理参数监测系统设计与实现[D].北京理工大学, 2014.
[4]钱显毅. 传感器原理与应用[M].南京:东南大学出版社, 2008.
[5]杨宝清. 现代传感器技术基础[M].北京：北京铁道出版社, 2002
[6]汤申生. 第三代移动通信系统无线传输技术的发展[J]. 电信科学, 1998(10):12-15.
[7]郭青. 基于 ZigBee 技术的独居老人远程监护系统的设计与实现[D].河北工程大学, 2016.
[8]宋菲. 基于 ZigBee 和 OpenCV 的老人智能监护系统的设计与实现[D].上海交通大学, 2015.
[9]秦蔚. ARM 平台下 Linux 内核移植技术的分析研究与应用[D].昆明理工大学, 2004.
[10]杨星. 基于 S3C2440 平台的 Linux 系统移植[D].北京交通大学, 2011.

致谢

自开题到即将结题的这半年里，我经历了许多。本来因为考研而耽误了完成毕业设计的时间，所幸，我的毕业设计指导导师，胡杰老师，对于我的时间安排从没有做强制要求，一直以来对我总是谆谆善诱，所以让我有足够的时间进行构思和自定方案。同时在我遇到技术难题的时候，给予我许多好的建议。同时，提供我毕业设计所需所有的软件和硬件设备，让我可以全身心投入到系统设计中来，节省了宝贵的时间。此外，他对于我的毕业设计的要求也比较严格，从他身上我不仅学到了专业知识，更能学习到一种严谨的做事和科研的态度。同时也感谢这半年来陪伴我的几个同学，和他们的交流让我更高效地完成了毕业设计，同时也是他们的加油和鼓励让我一直坚持把系统做到自己能达到的最优。

最后感谢长江大学所有曾在学业和生活中帮助过我的老师们，是你们一直的呵护和培养，使我从一个无知的孩子慢慢变成一个具备初步科研基础的合格的本科毕业生，让我面对未知的未来具有了勇气和智慧，是你们的辛勤付出让我得以成长。在这里向所有帮助过我的老师和同学们，致以我最真挚的敬意，谢谢你们。

章浩

2017 年 5 月

附录

(1) 本系统涉及到的传感器

图 35　HQ-2 有害气体检测模块

图 36　DHT11 温湿度传感器模块

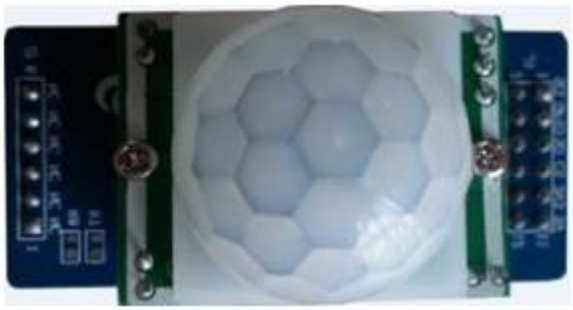


图 37　HC-SR501 人体红外感应模块

(2) 本系统的 ZigBee 模块及 Exynos4412

本系统所需要的核心硬件包括 ZigBee 开发套件（终端节点可做采集器，以及协调器）、Exynos4412 核心板，如下图：

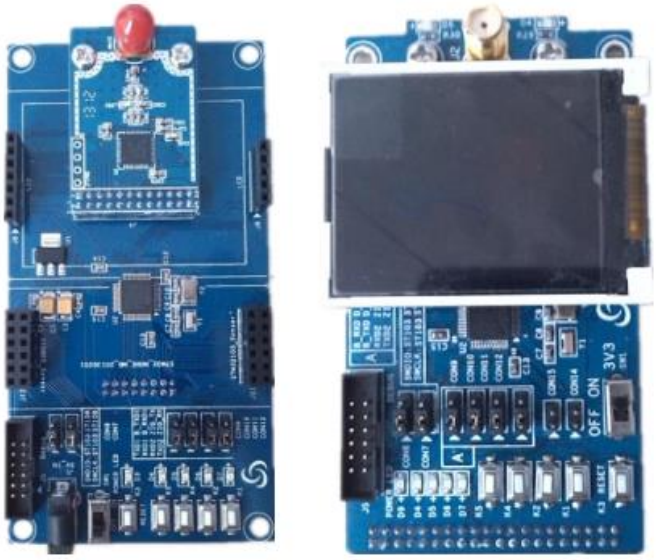


图 38　ZigBee 开发套件

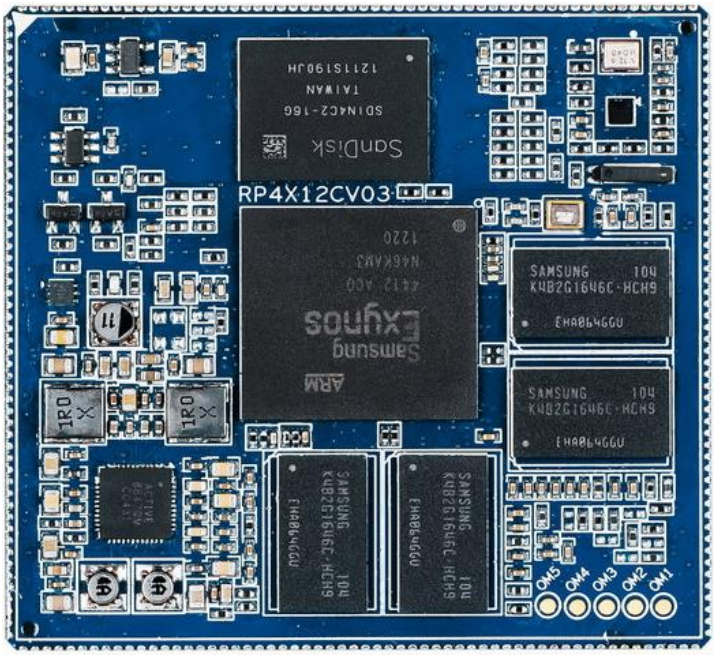


图 39　Exynos4412 核心板