

GAINZ 软件用户使用手册

宁波中科集成电路有限公司为您提供无线传感器网路方面全方位的技术支持,包括自主开发的 GAINS 系列节点和各种配套的后台软件以及 TestBed 无线网络测试平台,希望我们的产品为您的学习和研究带来方便。

公司地址: 浙江省宁波市科技园区创业大厦 6 层 WSN 事业部

邮编: 315040

产品主页: <http://www.wsn.net.cn/wsnb/wsn.htm>

客户服务热线: 0574-87910141

E_mail: jp@nbicc.com

目录

引言	- 1 -
提醒	- 1 -
特性	- 1 -
注意	- 1 -
免责声明.....	- 1 -
GAINZ 节点硬件介绍	- 2 -
软件协议栈对节点资源占用	- 3 -
软件协议栈对节点 I/O 资源的占用要求.....	- 3 -
软件协议栈对节点定时资源的占用要求.....	- 3 -
软件协议栈对节点其余资源占用要求.....	- 3 -
安装源文件.....	- 4 -
源文件构成.....	- 4 -
应用程序演示.....	- 4 -
演示程序的功能.....	- 4 -
编译演示应用程序的正确步骤.....	- 6 -
将演示应用程序烧写到器件中.....	- 6 -
演示应用程序.....	- 6 -
SNAMP_Z 使用说明.....	- 7 -
软件协议栈结构.....	- 10 -
物理层函数说明.....	- 11 -
MAC 层函数说明.....	- 18 -
软件协议栈编程注意.....	- 38 -
配套 Makefile 重点参数使用指南.....	- 40 -

引言

ZigBee™是一种短距离、低速率无线网络技术，它在工业控制，交通监控，仓储物流，环境和构筑物监测、抢险救灾以及军事等多个方面有着广泛的应用前景。ZigBee的基础是IEEE802.15.4无线个人域网标准。

802.15.4包括用于低速无线个人域网（LR-WRAN）的物理层和媒体接入控制层两个规范。它能支持消耗功率最少，一般在个人活动空间（10m直径或更小）工作的简单器件，支持两种网络拓扑，即单跳星状或当通信线路超过100m时的多跳对等拓扑，对等拓扑的逻辑结构由网络层定义。LR-WRAN中的器件既可以使用64位IEEE地址，也可以使用在关联过程中指配的16位短地址。一个802.15.4网可以容纳最多65536个器件。

本文旨在引导用户使用GAINZ 2.4Ghz开发套件，使您对套件的软件机构和硬件系统有更深入的了解。手册提供对物理层和MAC层协议栈函数库中所有接口函数的详细的说明，使得用户可以方便的调用并且进行网络层等更高层次应用的开发。

提醒

协议栈使用C语言编程，本文中对函数的解释均以C语言的形式出现。本文使用了大量的ZigBee和IEEE802.15.4规范的术语，但只给出了规范的简单介绍，用户可以参阅ZigBee和IEEE802.15.4规范文档来阅读手册。

特性

GAINZ软件协议栈的设计考虑了多方面的因素，在发布该文档时，协议栈具有以下特性：

- ◆完全符合IEEE802.15.4协议，按照原语开发
- ◆使用Chipcon CC2420 RF收发器支持2.4GHz频带
- ◆支持简化功能设备（Reduced Function Device, RFD）和协调器（Coordinator）
- ◆支持信标帧广播和超帧结构
- ◆可以在大多数ATmega系列单片机之间进行移植
- ◆采用带时槽的CSMA-CA算法实现信道的访问
- ◆模块化设计使得协议栈的结构十分清晰，易于删除和修改
- ◆支持AVR-GCC 3.4.3编译器

注意

在使用GAINZ协议栈时请注意一些局限：

- ◆本协议栈不支持GTS时槽分配，不支持安全功能。
- ◆本协议栈仅提供802.15.4PHY/MAC层支持，对于ZigBee网络层以上由于商业版权问题，本协议栈不与提供。
- ◆本协议栈仅提供802.15.4相应的库文件，不提供协议源代码，但是对所有的接口函数做了详细的说明。

免责声明

（1）本公司提供该版本协议栈仅用于各科研机构学术研究用途，不确保商业用途性能稳定，用户使用本协议栈做商业用途产生任何后果，本公司不负任何相关责任。

GAINZ 节点硬件介绍

GAINZ 节点的硬件由传感器模块、处理模块、无线通信模块和能量供应模块四个部分组成，如图一所示。

传感器模块负责监测区域内信息的采集和数据转换，处理模块负责控制整个传感器节点的处理操作、存储和处理本身采集的数据和其它节点发来的数据，包括了数据安全、通信协议、同步定位、功耗管理、任务管理等等；无线通信模块负责与其它传感器节点进行无线通信，交换控制消息和收发采集数据；电源供应模块为传感器节点提供运行所需的所有电源。

处理模块:处理核心选用 8 位低功耗微控制器 ATMEGA128，相对于其他通用的 8 位微控制器来说，它具有非常丰富的资源，具有片内 128K 字节的程序存储器（Flash），4K 字节的数据存储器（SRAM，可外扩到 64K）和 4K 字节的 E2PROM。

无线通信模块:无线通信模块核心采用工作在 2.4GHZ 的单芯片低电压 CC2420 收发器。CC2420 片上集成了 802.15.4 的物理层功能,并硬件支持部分 MAC 层功能的实现,具有工作电压低（2.1v~3.6v 均可工作）、能耗低、体积小等非常适合于集成的特点。它采用 QPSK 调制方式，最大收发波特率 250kbps，外部采用 SPI 的接口，可以和微控制器直接接口。

传感器模块:考虑到各种不同的应用场合中需要采集的模拟量千差万别,我们选用了一些当今应用最为常见的传感器，设计了拥有通用接口的传感器子模块.传感器电路部分设计采用 power gating 技术在无数据采集任务时降低功耗。

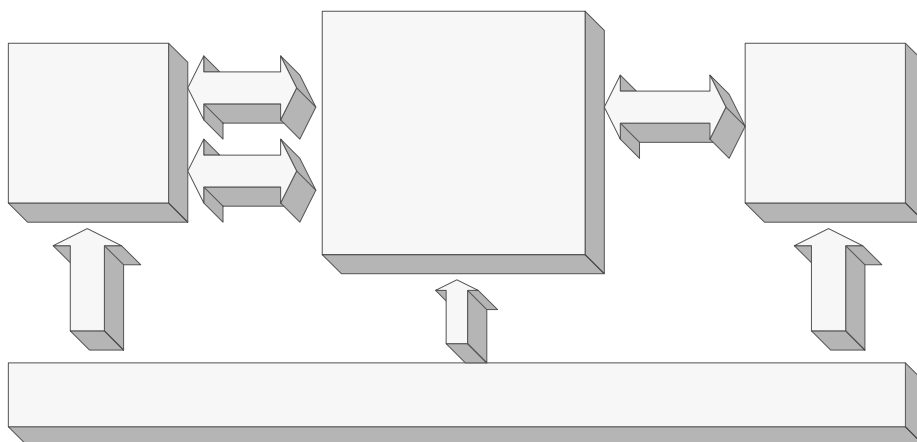


图 1: GAINZ 节点硬件结构图

软件协议栈对节点资源占用

软件协议栈对节点其余资源占用要求

软件协议栈对节点I/O资源的占用要求

软件对节点硬件的占用主要是用于与射频芯片之间的通信。

单片机与 RF 芯片的接口

ATMEGA128引脚	CC2420引脚
PA6 (输出)	CC2420: RESETn
PA7 (输出)	CC2420: VREGEN
PB0 (输出)	CC2420: CSn
PB1 (输出)	CC2420: SCLK
PB2 (输出)	CC2420: SI
PB3 (输入)	CC2420: SO
PB7 (输入)	CC2420: FIFO
PD4 (输入)	CC2420: SFD
PD6 (输入)	CC2420: CCA
PE6 (输入)	CC2420: FIFOP

ATMEGA128	用途
串口0	调试, 输出
SPI	驱动CC2420芯片

软件协议栈对节点定时资源的占用要求

软件对定时器资源的占用主要是由于支持超帧结构以及对 802.15.4 规范中时间敏感的传输过程的支持

ATMEGA128 定时器	用途
定时器0	演示占用时钟
定时器1	系统时钟
定时器3	协议栈私用

安装源文件

可从光盘中的802.15.4文件夹中取得源代码。源代码以一个Windows® 安装文件形式发（setup.exe）。执行下列步骤完成安装：

1. 执行setup.exe 文件； Windows安装向导将指导您完成安装过程。
2. 在继续安装之前，必须通过单击I Accept（我接受）接受软件许可协议。
3. 在完成安装过程之后，完整的源代码将被复制到您的计算机的根驱动器中的program files\GAINZ 目录中。

源文件构成

GAINZ协议栈由许多源文件组成，包括以c代码形式提供的文件与库文件，此外还提供演示应用程序用的源文件。所有的源文件都位于GAINZ文件夹中。

表：源文件结构

目录名称	内容
Mib_beacon_coordinator_save	Coordinator演示用的源文件与库
Mib_beacon_RFD_save	Rfd演示用的源文件与库

注：

1 协议栈根据Coordinator 与 Rfd不同的应用特点，分别优化生成了函数库，函数库分别位于相应的目录中，[建议用户开发coordinator用户应用程序时，在Mib_beacon_coordinator_save的演示程序的基础上编写和修改代码。](#)建议用户开发Rfd用户应用程序时，[在Mib_beacon_RFD_save的演示程序基础上编写和修改代码。](#)

2 用户可以在原有的文件基础上添加新的文件，这需要在 Makefile 文件中做修改，Makefile 的详细信息请看（Makefile 使用）

应用程序演示

GAINZ 协议栈包含两个应用程序：

- 1 Coordinator 应用程序：演示符合 802.15.4 规范的 Coordinator 设备的应用程序。
- 2 Rfd 应用程序：演示符合 802.15.4 规范的 Rfd 设备的应用程序

演示程序的功能

1.0 版的演示程序提供了如下功能：

- 旨在与 GAINZ 硬件一起使用
- 使用系统休眠与基于事件驱动机制演示低功耗处理
- 使用串口工具演示 802.15.4 网络建立过程
- 使用 SNAMP_Z 后台软件演示传感器数据采集和发送
- 支持 WINAVR 与 GCC 编译器 3.4.3

演示程序文件详细介绍

演示程序中 Coordinator 的文件列表

源文件	文件用途
Os.c	操作系统源码
Router.c	演示用 router 代码
Systime.c	提供系统时间
Kertime.c	提供系统静态定时器
Time_queue.c	基于定时器 3 的时间队列
Uart0.c	串口 0 驱动 (printf 缺省输出流)
Main.c	主文件
libPhy.a/ Libmac.a	802.15.4PHY 与 MAC 层库函数
Generic.h	系统参数设置
Macs.c	802.15.4 调用上层函数接口
Led.c	Led 驱动
Interrupter.c	中断处理

演示程序中 Rfd 的文件列表

源文件	文件用途
Os.c	操作系统源码
Router.c	演示用 router 代码
Systime.c	提供系统时间
Kertime.c	提供系统静态定时器
Time_queue.c	基于定时器 3 的时间队列
Uart0.c	串口 0 驱动 (printf 缺省输出流)
Main.c	主文件
libPhy.a/ Libmac.a	802.15.4PHY 与 MAC 层库函数
Generic.h	系统参数设置
Macs.c	802.15.4 调用上层函数接口
Led.c	Led 驱动
Interrupter.c	中断处理

编译演示应用程序的正确步骤

此处假定您已能熟悉使用，若情况并非如此，请参阅 WINAVR 的用户手册来创建、打开、编译项目

1. 确认安装了GAINZ 协议栈的源文件。如果尚未安装，请参阅“安装源文件”。
2. 启动WINAVR 并创建相应的项目文件：项目文件的确切名称取决于您选择的编译器。
3. 使用WINAVR 菜单命令来编译项目。注意仅当源文件位于硬盘驱动器根目录/program files的GAINZ中的相关目录中时，创建的演示应用程序项目才能正确工作。

如果您已将源文件移动到了另一个位置，则必须修改现有的Makefile文件才能编译项目。4. 编译过程应该成功完成。如果未成功编译，请确认正确设置了WINAVR。

将演示应用程序烧写到器件中

要使用两个演示应用程序之一对目标板进行编程，必须要有AVR编程器。下面的步骤假定您将使用我们提供的AVR JTAG编程器作为编程器。如果使用其他编程器，请参阅具体编程器的说明。

1. 将AVR JTAG 连接到GAINZ 演示板或您的目标板。
2. 对目标板通电。
3. 启动AVRStudio。
4. 点击TOOLS。
5. 选择STK500作为编程器。

6. 如果您希望使用以前编译生成的hex 文件，只要导入相应的hex文件即可。

7. 如果正在重新编译hex 文件，请打开相应的演示项目文件并遵循编译步骤来生成应用程序hex 文件。

8. 单击**Program**（编程）菜单选项，开始对目标板编程。

9. 数秒之后，应该看到“Programming successful”（编程成功）等相关消息。如果未看到这条消息，请仔细检查板和WINAVR的连接和串口是否被占用。

10. 除去板的电源并断开AVR JTAG 电缆与目标板的连接。

11. 重新为该板加上电源。

演示应用程序

1. 将演示程序Coordinato与Rfd编译成的hex文件，分别烧写进相应的节点中。
2. 将串口线分别连至目标板
3. 用户打开打开配套的SNAMP_Z软件，可以看到Rfd传送给Coordinator的传感数据的动态图表（关于SNAMP_Z使用请见SNAMP_Z使用说明）
4. 首先给Coordinator节点通电
5. 接着给Rfd节点通电
6. 观察Coordinator与Rfd串口数据

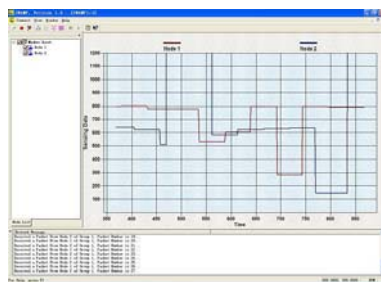


图 2 SNAMP_Z后台获取Rfd传感值并动态显示

SNAMP_Z 使用说明

SNAMP_Z 后台是用来可视化 GAINZ 网路运行过程的一个后台软件。SNAMP_Z 后台运行于与 Coordinator 节点相连的主机上面，通过串口（通过 usb 口的暂时不提供）SNAMP_Z 可以读取 Coordinator 节点收到的数据，并且对这些数据进行分析，最后显示出网络运行时的动态效果。我们提供的 MAC 版的 Demo 实现的是对光强数据信息的定时采集，时间设定为每 1 秒钟将采集的数据发送一次。节点上面有 3 个指示灯，为红，绿和黄三种颜色，对于 Coordinator 节点，每收到一个数据包红灯就闪烁一次。对于 Rfd 节点，不管在何种情况下，每发一次数据红灯就要闪烁一次。下面是 SNAMP_Z 后台软件的使用说明。

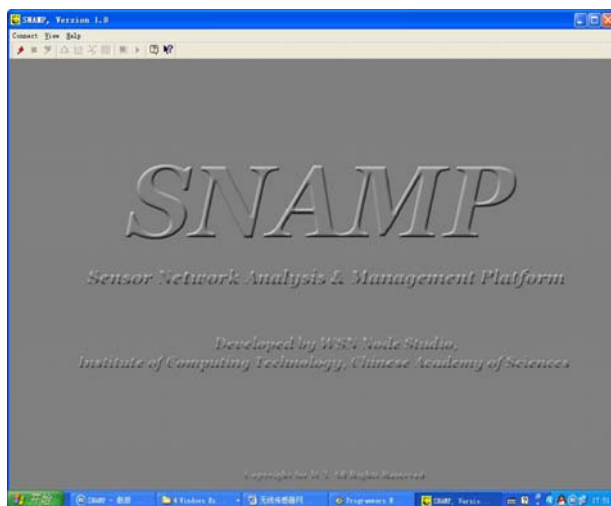



图 3 SNAMP_Z 后台软件

打开 SNAMP_Z 后台软件后会出现上图的界面，选择 Connect/Connecting 或点击按钮  就可以进入 Connect Network....界面，在该界面中可以选择协议类型，串口号和节点组号。我们提供的协议是 MAC 协议，需要在 Node Program 中选择 XPMAC。

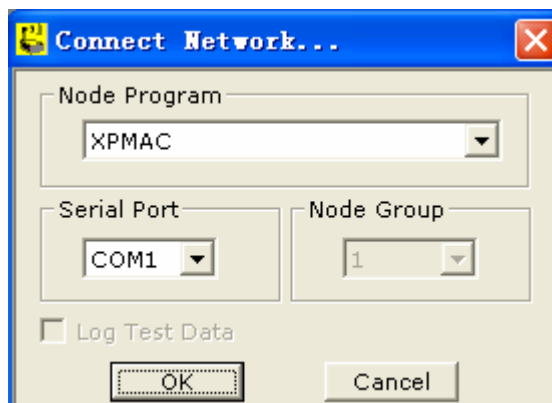


图 4 SNAMP_Z 后台类型选择

节点程序中设定节点的组号是 1，因此在 Node Group 中应该选择 1 至于串口号 (Serial Port)，那就需要看情况了，SNAMP_Z 后台一共支持 3 种选项，COM 1 代表串口 1，COM 2 代表串口 2，一般的主机就只有 2 个串口，SNAMP_Z 的第三种选择是对应 usb 接口的，它是和 GIANTSU 配套使用的，一般情况下是看不到 COM 3 选项的，只有在接上 usb 设备的时候才可以被看到和被选则。需要注意的是，我们发布的免费的 SNAMP_Z 后台是 MAC 专用的，有些和 MAC 协议的分析无关的功能已经屏蔽掉了，有些选项已经固定了，不能更改。

图 5 是 SNAMP_Z 后台运行时拓扑效果图：

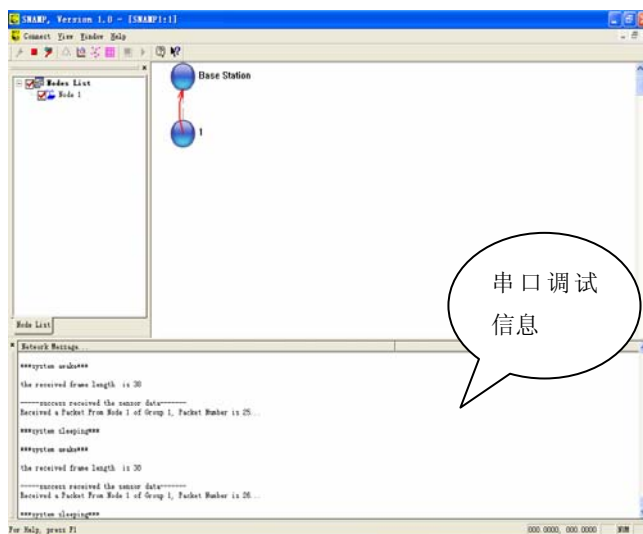


图 5 SNAMP_Z 效果图

图 6 是显示网络数据的动态曲线图，是根据节点发来的数据包，通过解析获得节点所在位置的光强信息，然后用动态曲线描绘出来，不同节点的曲线用不同的颜色标示。



图 6 动态数据效果图

软件协议栈结构

根据 IEEE 802.15.4 的协议，协议栈包括 PHY 层和数据链路层的 MAC 子层。PHY 层由射频收发器驱动以及服务原语构成，MAC 子层为高层访问物理

信道提供点到点通信的服务接口。MAC 层以上的层次，可由用户自行定义，方便进行二次开发。

表 IEEE 802.15.4 规范各层所定义的功能

	物理层	MAC 层
需要实现的功能	<ul style="list-style-type: none"> ● 打开或关闭射频收发器 ● 检测当前信道的能量 ● 对收到的包的质量评估 ● 检测当前信道是否被占用 ● 选择信道频率 ● 接受数据和发送数据 	<ul style="list-style-type: none"> ● 为协调者产生信标帧，普通设备可以同步到信标帧 ● 支持关联和取消关联 ● 支持无线通信安全 ● 使用CAMA-CA算法访问信道 ● 管理和分配GTS时槽 ● 在两个MAC实体间提供一个可信任的信道

根据规范定义的功能的要求，参考 IEEE 802.15.4 规范，协议栈使用如图 3 所示的架构：各层定义独立的数据访问接口与管理访问接口，层与层之间通过接口互连。其中数据访问接口提供高层发送和接收数据的能力，管理访问接口提供高层管理和设置物理层属性的能力。

物理层定义了物理无线信道和 MAC 子层之间

的接口，提供物理层数据服务和物理层管理服务。物理层数据服务从无线物理信道上收发数据，物理层管理服务维护一个由物理层相关数据组成的数据库。

MAC 子层使用物理层提供的服务实现设备之间的数据帧传输，而 LLC 在 MAC 子层的基础上，在设备间提供面向连接和非连接的服务。MAC 子层提供两种服务：MAC 层数据服务和 MAC 层管理服务（MAC sublayer management entity, MLME）。前者保证 MAC 协议数据单元在物理层数据服务中的正确收发，后者维护一个存储 MAC 子层协议状态相关信息的数据库。

本协议栈程序基本符合 15.4 规范，在程序中完整的体现了物理层和 MAC 子层的各种服务原语功能，下面将对其进行详细的阐述。

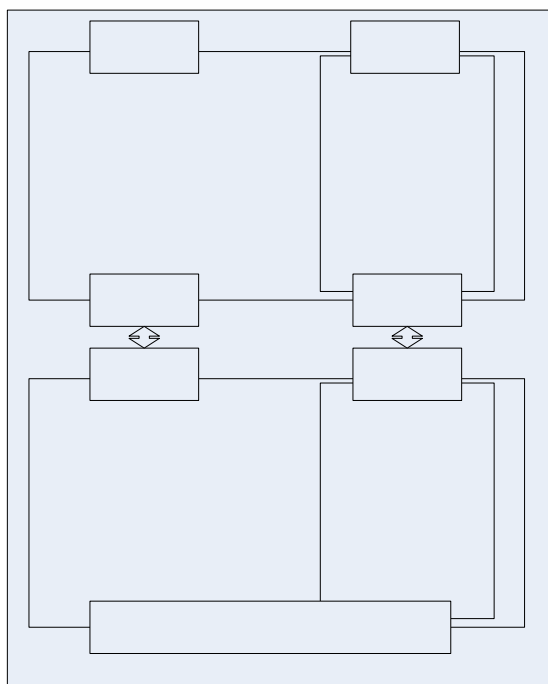


图 7 802.15.4 协议栈软件架构

物理层函数说明

phy_init

该函数初始化物理层，包括 CC2420 和 ATmega128 定时器\计数器 1(系统时钟)的初始化

语法

```
void phy_init(void)
```

参数

无

返回值

无

前提

无

副作用

无

注

这是在这个函数里确定了信道、发射功率及 CCA 模式

示例

```
// 初始化协议栈  
phy_init();  
.....
```

PD_DATA_request

物理层发送数据服务原语。该函数将一个 PSDU 发送出去，同时记录发送时间

语法

```
uint32_t PD_DATA_request (uint8_t psdulength, uint8_t * psdu)
```

参数

Psdulength [in]

—PSDU 长度

Psdu [out]

—发送数据缓存指针

返回值

发送 PSDU 的系统时间

前提

无

副作用

无

注

无

示例

```
// 发送信标帧
uint32_t direct_transmit(uint8_t psdulength, uint8_t * psdu)
{
    //直接发送帧，不采用 csma_ca
    //调用物理层 PD_DATA_request 发送
    uint32_t send_timestamp;
    send_timestamp=PD_DATA_request (psdulength, psdu); //传输数据同时获取传输时间戳
    free(psdu);

    return send_timestamp;
}
```

PLME_CCA_request

该函数通过向 PLME 发送请求，得到当前信道的状态

语法

```
uint8_t PLME_CCA_request (void)
```

参数

无

返回值

当前信道状态：

TRX_OFF —收发器处于关闭状态

TX_ON—发送器打开

BUSY—信道忙

IDLE—信道空闲

前提

无

副作用

无

注

函数的返回值实现了 15.4 协议原语中的 PLME_CCA_confirm 的功能，所以不在单独写

PLME_CCA_confirm 函数

示例

```
// 执行 CSMA-CA 算法
void CSMA_CA_DOING(void)
{
    .....
    case csma_state_delaying:
        cca_result=PLME_CCA_request();
    .....
}
```

PLME_ED_request

该函数使得物理层执行一次信道信号能量检测

语法

```
ED_STATE PLME_ED_request(void)
```

参数

无

返回值

当前的状态及信号强度值

前提

无

副作用

无

注

无

示例

```
// MAC 层进行 ED channel SCAN
MAC_ENUM mlmeScanRequest(BYTE scanType, DWORD scanChannels,
                          UINT8 scanDuration)
{
    .....
    if(temp_return_SET.state==PHY_SUCCESS)
    {
        while(systemtime32()!=object_time)
        {
            temp_return_ED=PLME_ED_request();.....
        }
    }
}
```

PLME_SET_TRX_STATE_request

该函数用来改变当前物理层收发器的状态

语法

```
uint8_t PLME_SET_TRX_STATE_request (uint8_t state)
```

参数

state [in]

—RX_ON

—TRX_OFF

—FORCE_TRX_OFF

—TX_ON

返回值

无

前提

无

副作用

无

注

该函数实现了和 15.4 原语中相同的功能

示例

```
// 判断 MAC 层状态并做出相应的处理
.....
phy_channel_state=PHY_IDLE;

        mac_current_state=MAC_STATE_IDLE;
        PLME_SET_TRX_STATE_request (RX_ON);
.....
```

PLME_GET_request

语法

```
GET_STATE PLME_GET_request(uint8_t pibattribute)
```

参数

Pibattribute [in]

表 1 一个人局域网信息数据库参数，如下所示

属性	描述
phyCurrentChanel_id	物理层当前信道
phyTransmitPower_id	发送信号强度
phyCCAMode_id	CCA 模式

返回值

物理层目前 PIB 状态，获得的属性和属性值

前提

无

副作用

无

注

在 15.4 协议中规定的物理层数据库中的 phyChannelsSupported 没有作为函数的参数，Confirm 原语由该函数的返回值实现

示例

```
// 本程序中没有使用该函数，用户可利用它作为接口获得物理层参数
```

PLME_SET_request

该函数用以设置物理层 PIB 参数

语法

```
SET_STATE PLME_SET_request(uint8_t pibattribute,
                             uint8_t pibattributevalue)
```

参数

Pibattribute [in]

—物理层 PIB 属性

pibattributevalue[in]

—属性值

返回值

参数设置的结果	SUCCESS, UNSUPPORTED_ATTRIBUTE
确认设置的参数	表 1

前提

无

副作用

无

注

在返回值里有 INVALID_PARAMETER 一项，在本程序中统一 UNSUPPORTED_ATTRIBUTE 代替

示例

```
// MLMEscan
MAC_ENUM mlmeScanRequest(BYTE scanType, DWORD scanChannels, UINT8 scanDuration)
{.....
    while((scanChannels&((uint32_t)1<<scan_channel_index))==0)
    {
        scan_channel_index++;
        if(scan_channel_index==16)
            return;
    }
    PLME_SET_request(phyCurrentChanel_id, scan_channel_index++); //设置信道号
```

MAC 层函数说明

MAC_init

MAC 初始化函数，主要是对一些数据结构的初始化

语法

```
void MAC_init(void)
```

参数

无

返回值

无

前提

首先必须完成对物理层的初始化，即调用 Phy_init()

副作用

无

注

无

示例

```
// 主程序 MAIN 初始化
int main(void)
{
    hardwareInit();
    Leds_greenOn();
    Leds_redOn();
    Leds_yellowOn();
    IoInit();
    OS_sched_init();
    phy_init();
    .....
    MAC_init();
    .....
}
```

mlmeResetRequest

语法

```
MAC_ENUM mlmeResetRequest (bool setDefaultPIB)
```

参数

```
setDefaultPIB [in]
```

—TRUE

—FALSE

返回值

关闭收发器是否成功

前提

无

副作用

无

注

该函数对 MAC PIB 进行了初始化，并发送 PLME_SET_TRX_STATE.request 原语，关闭收发器

示例

```
// 主程序中判断 MAC 状态
switch(mac_current_state)
{
    case MAC_STATE_UNSTARTED:
        //printf("\n----START TO FORM NETWORK----\n");
        MAC_init();
        mlmeResetRequest(TRUE);
        IS_NETWORK_FORMED=FALSE;
        mlmeScanRequest(ED_SCAN, 0x00000008, 6);
    .....
}
```

mlmeScanRequest

该函数发起一个扫描，包括对扫描类型和信道及持续时间的确定

语法

```
MAC_ENUM mlmeScanRequest (BYTE scanType, DWORD scanChannels,  
                           UINT8 scanDuration)
```

参数

scanType [in]

—扫描类型

ED_SCAN

ACTIVE_PASSIVE_SCAN

ORPHAN_SCAN

scanChannels [in]

—待扫描的信道

scanDuration [in]

—扫描持续时间

返回值

无

前提

必须在此之前初始化 MAC 层

副作用

无

注

ACTIVE SCAN 和 PASSIVE SCAN 基本相同，在本函数中作为同一种情况来处理

示例

```
// 当 MAC 完成 EDSCAN 后进行 ACTIVE_PASSIVE_SCAN  
switch(mac_current_state)  
.....  
    case MAC_STATE_ENERGY_SCAN_FINISHED:  
        mlmeScanRequest (ACTIVE_PASSIVE_SCAN, 0x00000008, 6);  
        break;  
.....
```

mlmeScanConfirm

该函数根据 mlmeScanrequest 的结果来对 MAC 和 PHY 的 PIB 做出修改

语法

```
void mlmeScanConfirm(MLME_SCAN_CONFIRM scan_result_info)
```

参数

```
scan_result_info [in]
```

—扫描结果信息

返回值

无

前提

已经进行了 MLME_SCAN

副作用

无

注

无

示例

```
//  
Void mlmeBeaconNotifyIndication(MLME_BEACON_NOTIFY_INDICATION beacon_info)  
{.....  
if(scan_result_info.resultListSize>MAC_MAX_PAN_DESCRIPTOR)  
    {head_of_timer_queue=timer3quere_cancel(head_of_timer_queue, 0, WAIT_SCAN);  
    mlmeScanConfirm(scan_result_info);  
.....
```

mlmeStartRequest

该函数开始组织一个新的超帧

语法

```
void mlmeStartRequest(SHORT_ADDR panId, uint8_t logicalChannel, uint8_t  
beaconOrder, uint8_t superframeOrder, bool panCoordinator,  
bool batteryLifeExtension, bool coordRealignment, bool securityEnable, WORD  
StartTime)
```

参数

panId	[in]	PAN 标志符
logicalChannel	[in]	发送 BEACON 帧的信道
beaconOrder	[in]	发送 BEACON 帧的间隔
superframeOrder	[in]	超帧活跃部分的长度
panCoordinator	[in]	是否为协调器
batteryLifeExtension	[in]	选择是否在 IFS 关闭接收器以节能
coordRealignment	[in]	是否在超帧机构改变之前发送重新组合命令
securityEnable	[in]	为 BEACON 发送选择保密模式
StartTime	[in]	发送 BEACON 的时间

返回值

无

前提

完成 ACTIVE SCAN 或 PASSIVE SCAN

副作用

无

注

如果是网络协调者，将忽略该函数的 starttime 参数

示例

```
// 协调器在设置好 PANID 和 SHORTADDRESS 后开始发送超帧  
mlmeStartRequest(mac_pib_data.macPANId, 3, mac_pib_data.macBeaconOrder,  
mac_pib_data.macSuperframeOrder, TRUE, mac_pib_data.macBattLifeExt, FALSE, FALSE,  
0);
```

mlmeAssociateRequest

该函数发起一个与协调器连接请求

语法

```
void mlmeAssociateRequest(UINT8 logicalChannel, BYTE coordAddrMode, SHORT_ADDR  
coordPANId, ADDRESS pCoordAddress, Node_Capability capabilityInformation, bool  
securityEnable)
```

参数

logicalChannel	[in]	在哪个信道上发起连接
coordAddrMode	[in]	协调器地址模式
coordPANId	[in]	待连接的 PAN 标志符
pCoordAddress	[in]	协调器的地址
capabilityInformation	[in]	请求者性能信息
securityEnable	[in]	保密模式选择

返回值

无

前提

无

副作用

无

注

无

示例

```
// 主程序中 MAC 状态判断进行响应的处理  
switch(mac_current_state)  
.....  
    case MAC_STATE_SYNING_END:  
        .....  
        mlmeAssociateRequest(ppib.phyCurrentChanel, 0x02, mac_pib_data.macPANId, temp_  
            addr , temp_node, FALSE);  
        .....  
.....
```

mlmeAssociateConfirm

该函数报告上层连接的结果

语法

```
void mlmeAssociateConfirm(SHORT_ADDR AssocShortAddress, BYTE status)
```

参数

AssocShortAddress	[in]	由协调器分配的地址
status	[in]	连接的结果

返回值

无

前提

已进行 MLME_Associate

副作用

无

注

无

示例

```
//  
void mac_timer_task(void)  
{.....  
case MAC_CMD_ASSOCIATE_REQ:  
  
    if(mac_current_state==MAC_STATE_ASSOC_WAIT_FOR_RESPONSE)  
        mlmeAssociateConfirm(default_short_addr,NO_ACK);  
.....}
```

mlmeAssociateIndication

该函数作用是协调者在收到 associate request 帧后通知高层

语法

```
void mlmeAssociateIndication(MLME_ASSOCIATE_INDICATION  
mlme_associate_indication_info)
```

参数

mlme_associate_indication_info [in]

返回值

无

前提

无

副作用

无

注

该函数的主要任务是调用 mlmeassociateresponse 原语

示例

```
// 处理收到的 MAC 帧  
void mac_frame_rcvd(void)  
.....  
if((mac_current_state==MAC_STATE_IDLE)&&(IS_FFD!=0x00))  
    {  
mlme_associate_indication_info.cap_info=mac_rx_current_frame.node_capinfo_field;  
mlme_associate_indication_info.Deviceaddr=mac_rx_current_frame.src.longAddr;  
mlmeAssociateIndication(mlme_associate_indication_info);  
.....
```

mlmeAssociateResponse

该函数对 MAC 层收到的连接请求做出响应

语法

```
void mlmeAssociateResponse(LONG_ADDR deviceAddress,  
SHORT_ADDR assocShortAddress, BYTE status, bool securityEnable)
```

参数

deviceAddress	[in]	请求连接的设备地址
assocShortAddress	[in]	协调器分配给该设备的地址
status	[in]	连接的状态
securityEnable	[in]	是否使用保密模式

返回值

无

前提

收到 MAC 层的 mlmeAssociateIndication 原语

副作用

无

注

该函数本应该由高层来建立，但是为了调试将该函数在 MAC 层中实现

示例

```
//  
Void mlmeAssociateIndication(  
MLME_ASSOCIATE_INDICATION mlme_associate_indication_info)  
.....  
    mlmeAssociateResponse(mlme_associate_indication_info.Deviceaddr, shortaddr_alloc,status,  
FALSE);  
.....
```

mlmeDisassociateRequest

该函数发起一个请求离开 PAN 的命令

语法

```
void mlmeDisassociateRequest(LONG_ADDR pDeviceAddress, BYTE disassociateReason,  
bool securityEnable)
```

参数

pDeviceAddress	[in]	请求离开 PAN 的设备地址
disassociateReason	[in]	离开的原因
securityEnable	[in]	是否使用保密模式

返回值

无

前提

无

副作用

无

注

该函数本应由高层实现

示例

```
//
```

mlmeDisassociateIndication

通知高层处理 Disassociate 请求，有高层完成功能

语法

```
void mlmeDisassociateIndication(LONG_ADDR deviceAddress, BYTE disassociateReason,  
bool securityUse, BYTE aclEntry)
```

参数

deviceAddress	[in]	请求离开 PAN 的设备地址
disassociateReason	[in]	离开的原因
securityUse	[in]	是否使用保密模式
aclEntry	[in]	访问连接控制

返回值

无

前提

协调者或终端设备在收到对方发过来的 disassociation notification 帧

副作用

无

注

该函数主要作用是确定 MAC 层当前状态，主要由高层实现

示例

```
// 处理 MAC 接收的帧  
void mac_frame_rcvd(void)  
{.....  
    if((IS_FFD!=0x00))  
    {  
        mlmeDisassociateIndication(mac_rx_current_frame.src.longAddr,  
        mac_rx_current_frame.disassociation_reason, FALSE, 0x00);  
        .....  
    }
```

mlmeBeaconNotifyIndication

该函数通知上层收到信标帧

语法

```
void mlmeBeaconNotifyIndication(  
    MLME_BEACON_NOTIFY_INDICATION beacon_info)
```

参数

beacon_info [in] 信标帧所携带的信息

返回值

无

前提

无

副作用

无

注

无

示例

```
//  
void mac_frame_rcvd(void)  
{  
.....  
if((mac_current_state_saved==MAC_STATE_ASSOC_WAIT_FOR_RESPONSE)||  
state_saved==MAC_STATE_ORPHAN_SCAN))  
    {mac_current_state=mac_current_state_saved;}  
    mlmeBeaconNotifyIndication(beacon_info);  
.....
```

mlmeOrphanIndication

该函数通知上层收到了孤立设备发出的 orphan notification command

语法

```
void mlmeOrphanIndication(LONG_ADDR orphanAddress, bool securityUse, BYTE  
aclEntry)
```

参数

orphanAddress	[in]	孤立设备的地址
securityUse	[in]	是否使用保密模式
aclEntry	[in]	访问连接控制

返回值

无

前提

无

副作用

无

注

该函数主要由上层实现

示例

```
//  
void mac_frame_rcvd(void)  
.....  
if((mac_current_state==MAC_STATE_IDLE)&&(IS_FFD!=0x00))  
    {  
mlmeOrphanIndication(mac_rx_current_frame.src.longAddr, 0,0x00);  
//coordinator_realignment_frame_send();由 mlme-orphan.response 原语发送.  
    }  
.....
```

mlmeOrphanResponse

该函数对 MAC 层的 OrphanIndication 做出响应

语法

```
void mlmeOrphanResponse(LONG_ADDR orphanAddress,  
SHORT_ADDR shortAddress, bool associatedMember, bool securityEnable)
```

参数

orphanAddress	[in]	孤立设备的地址
shortAddress	[in]	协调器分配给该设备的地址
associatedMember	[in]	是否成功加入 PAN
securityEnable	[in]	是否使用保密模式

返回值

无

前提

无

副作用

无

注

无

示例

```
//  
void mlmeOrphanIndication(LONG_ADDR orphanAddress, bool securityUse, BYTE  
aclEntry)  
{ .....  
shortaddress=short_address_allocate_req(orphanAddress);  
mlmeOrphanResponse(orphanAddress, shortaddress, TRUE, FALSE);  
}
```

mlmeSyncRequest

该函数发起一个同步的请求

语法

```
void mlmeSyncRequest(UINT8 logicalChannel, bool trackBeacon)
```

参数

logicalChannel	[in]	请求同步的信道
trackBeacon	[in]	是否跟踪信标帧

返回值

无

前提

无

副作用

无

注

无

示例

```
//  
void GET_ADC_value(void)  
{  
    //printf("\n--timer0 interrupt----\n");  
    read_data_task();  
    DATA_READY=TRUE;  
    mlmeSyncRequest(ppib.phyCurrentChanel, TRUE);  
}
```

mcpsDataRequest

该函数由高层发起请求发送数据

语法

```
void mcpsDataRequest(NODE_INFO dst, NODE_INFO src, UINT8 msduLength, BYTE *pMsdu,  
BYTE msduHandle, BYTE txOptions)
```

参数

Dst	[in]	目的节点
src	[in]	源节点
msduLength	[in]	MSDU 长度
pMsdu	[in]	MSDU
msduHandle	[in]	待发送的 MSDU 句柄
txOptions	[in]	发送选择

返回值

无

前提

无

副作用

无

注

无

示例

```
//  
int main(void)  
{.....  
mcpsDataRequest(dst, src, msdulength, pMsdu, msduhandle, txOptions);  
.....
```

PD_DATA_indication

该函数由物理层通知 MAC 层, 将物理层收到的数据包转交给 MAC 层处理

语法

```
uint8_t PD_DATA_indication(uint8_t psdlength, uint8_t * psdu, uint8_t  
ppdulinkquality, uint32_t timestamp)
```

参数

psdlength	[in]	PSDU 长度
psdu	[in]	PSDU
ppdulinkquality	[in]	PPDU 链路质量
timestamp	[in]	时间戳

返回值

无

前提

无

副作用

无

注

无

示例

```
//  
void phy_handlepacket(void)  
{  
.....  
PD_DATA_indication(packetlength,  
(cc2420_rxbuf_now), linkquality, receive_timestamp);  
.....
```

coordinator_realignment_frame_send

该函数的作用是当协调器收到 orphan_notification 命令帧或当 PAN 的属性发生改变时发出重新组成 PAN 的命令

语法

```
void coordinator_realignment_frame_send(bool Isunicast,  
    NODE_INFO dst, SHORT_ADDR allocated_addr)
```

参数

Isunicast	[in]	True	单播
		False	广播
Dst	[in]	目的节点	
allocated_addr	[in]	重新分配的地址	

返回值

无

前提

无

副作用

无

注

该命令帧可采用单播或广播的方式重新组合 PAN

示例

```
//  
void mlmeStartRequest(SHORT_ADDR panId, uint8_t logicalChannel, uint8_t  
beaconOrder, uint8_t superframeOrder, bool panCoordinator, bool  
batteryLifeExtension, bool coordRealignment, bool securityEnable, WORD StartTime)  
{.....  
if(!coordRealignment)  
    beacon_frame_send(TRUE);  
else  
    coordinator_realignment_frame_send(FALSE, default_node, default_short_addr);  
.....
```

timerqueue_adjust

该函数调整定时器 3 的队列

语法

```
void timerqueue_adjust(void)
```

参数

无

返回值

无

前提

无

副作用

无

注

无

示例

```
//  
void timer3queue_interrupt(void)  
{  
    timerqueue_adjust();  
    OS_post(mac_timer_task);  
}
```

mac_timer3queue_getnextinterval

该函数用以得到定时器的下一个定时值

语法

```
uint16_t mac_timer3queue_getnextinterval(void)
```

参数

无

返回值

下一个定时器值

前提

无

副作用

无

注

无

示例

```
//  
uint16_t timer3queue_getnextinterval(void)  
{  
    return mac_timer3queue_getnextinterval();  
}
```

软件协议栈编程注意

本公司提供的演示用802.15.4规范库文件基于状态机原理实现。

PHY状态空间包含：（0，1，2，3，4，5，6），PHY层状态用户基本不可见。

```
typedef enum {
    PHY_IDLE=0, //空闲状态
    DOINGCSMA_CA, //正在做CSMA_CA
    TRANSMITTING_FRAME, //正在发包
    WAITING_FOR_ACK, //正在等待ACK，此时收到其余任何包都丢掉
    WAITING_FOR_BEACON, //正在等待beacon，此时收到其余任何包都丢掉
    INACTIVE_PERIOD //非活跃期间，此时射频关掉或不接收发送任何包
} PHY_CHANNEL_STATE ;
```

MAC状态空间包含：（0，1，****，18）

```
typedef enum {
    MAC_STATE_UNSTARTED, //网络没有开始
    MAC_STATE_ASSOC_STARTED, //开始关联过程
    MAC_STATE_ASSOC_REQ_SENT, //关联请求已经发送
    MAC_STATE_ASSOC_WAIT_FOR_RESPONSE, //等待关联请求回复
    MAC_STATE_ASSOC_FINISHED, //关联结束

    MAC_STATE_ENERGY_SCAN, //正在做ED_SCAN
    MAC_STATE_ENERGY_SCAN_FINISHED, //ED_SCAN结束

    MAC_STATE_ACTIVE_PASSIVE_SCAN, //正在做主动或被动SCAN
    MAC_STATE_ORPHAN_SCAN, //孤儿SCAN
    MAC_STATE_ORPHAN_REALIGNED, //孤儿重新对齐

    MAC_STATE_ACTIVE_PASSIVE_SCAN_FINISHED, //主动或被动SCAN结束

    MAC_STATE_ID_CONFLICT, //Coordinator ID 冲突
    MAC_STATE_REALIGNMENT,
    MAC_STATE_DISASSOCIATED, //正在取消关联

    MAC_STATE_SYNING, //正在做同步
    MAC_STATE_SYNING_END, //同步结束

    MAC_STATE_IDLE, //射频空闲
    MAC_STATE_INACTIVE, //射频部分不可使用

    MAC_STATE_START_TO_SLEEP, //进入基于事件驱动的休眠，不保持与beacon
    同步
} MAC_STATE_TYPE;
```

上述的状态中，部分状态为用户不可见状态，部分状态为返回给高层，由高层根据当前状态做出相应的下一步调整。演示程序中在main函数中使用循环判断网络状态来驱动整个协议栈的运行，在macs.c文件中，协议栈运行后的返回值将直接反馈给各个用户接口函数，由用户根据返回值，调整网络状态，作出相应的下一步调整，在演示文件中已经做了一部分工作。用户可以采取单进程的方式组网，**但我们建议用户在演示文件循环判断的基础上进一步开发应用。**

建议用户使用协议栈编程遵循演示程序中对网络状态的修改。

举例说明：macs.c 文件

void mlmeScanConfirm(MLME_SCAN_CONFIRM scan_result_info)//该函数为做为SCAN后，通知上层结果，由上层作出下一步处理，此处，我们给出示例与说明：

```

{
//根据scan_result_info来修改对应的mac层和PHY层的pib信息
//本来有个复杂的选择算法，这里简单处理第一个
//printf("\n-----it is in scan confirm-----\n");
if((scan_result_info.status==SUCCESS)&&(scan_result_info.scanType==ORPHAN_SCAN))
    { //如果孤儿SCAN成功，则孤儿已经重新加入网络
        //printf("\n---SUCCESSFULL ORPHAN SCAN---\n");
        mac_current_state=MAC_STATE_IDLE;//SCAN成功后，有高层将信道设置为空闲，准备传输
数据或命令
        return;
    }
if((scan_result_info.status==SUCCESS)&&(scan_result_info.scanType==ACTIVE_PASSIVE_SCAN))
    {//如果主动或被动SCAN成功，
        //printf("\n-----it is in scan confirm-----\n");
        mac_current_state=MAC_STATE_ACTIVE_PASSIVE_SCAN_FINISHED;//高层可以将状态设置为
该参数，等待main函数中对该状态的响应，即准备做同步

        mac_pib_data.macBeaconOrder=((scan_result_info.pPANDescriptorList[0].superframeS
pec.SF.bytes.LSB)&0x0f);//提取SCAN得到的参数

        mac_pib_data.macSuperframeOrder=((scan_result_info.pPANDescriptorList[0].superfr
ameSpec.SF.bytes.LSB)>>4);
        .....
        .....
    }
}

```

我们提供的演示程序在mlmeScanConfirm函数中根据返回值修改了网络状态后，退出函数，重新进入到main函数中的循环判断中，再根据演示程序或用户修改的网络状态调用相应的函数推动整个网络的组建。

配套 Makefile 重点参数使用指南

1. MCU 指定

```
# MCU name
MCU = atmega128
```

2. 输出格式指定

```
# Output format.
FORMAT = ihex//输出格式可以为 srec, ihex, binary
```

3. 工作目录指定

```
WORKDIR=c:/GAINZ/mib_beacon_coordinator_save//缺省的安装目录
```

4. C 源文件添加或删减指定

```
# List C source files here. (C dependencies are automatically generated.)
SRC = $(TARGET).c led.c interrupter.c os.c kertimer.c avrhardware.c
```