# ZStack函数说明

# Zmain文件夹

Zmain.c文件包含了main主程序，需要注意的是osal\_init\_system()函数【初始化操作系统，包括了任务初始化】和osal\_start\_system()函数【执行操作系统，进入主循环】。

# OSAL文件夹

该文件夹中有许多文件，其中OSAL.c文件中包含了osal\_init\_system()函数和osal\_start\_system()函数。

# SampleApp文件夹

该文件夹主要包括OSAL\_SampleApp.c、SampleApp.c、SampleApp.h、SampleAppHw.c、 SampleAppHw.h。后两个函数主要通过读取跳线来确定设备是否为协调器，这里不涉及这个功能，所以不做介绍。

### 2.1 OSAL\_SampleApp.c文件

该文件主要包括了osalInitTasks()函数和taskArr[]数组。

#### osalInitTasks()函数

**调用路径：**

该函数由1. Zmain文件夹中的osal\_init\_system()函数调用，主要用于任务初始化。

路径：main.c -> osal\_init\_system() -> osalInitTasks().

**主要功能：**

主要功能是为操作系统的所有任务分配空间，并对各个任务进行初始化，包括MAC层、NWK层、HAL硬件抽象层(不属于Zigbee协议栈) 、APS层、和用户自定义层任务的初始化，其中用户自定义层任务初始化函数为SampleApp\_Init()函数。

#### taskArr[]数组

**调用路径：**

该数组在1.Zmain文件夹中的osal\_start\_system()函数使用，主要用于轮询调用任务处理函数。

路径：main.c -> osal\_start\_system() -> taskArr[]。

**主要功能：**

taskArr[]数组中存放了任务处理回调函数，在此数组中的事件处理函数与osalInitTasks()函数中的任务初始化函数是一一对应的(按顺序排列)。其中需要注意的是用户自定义的应用层任务处理函数SampleApp\_ProcessEvent()。

### SampleApp.c文件

该文件主要包括了SampleApp\_Init()函数、SampleApp\_ProcessEvent()函数、SampleApp\_HandleKeys()函数、SampleApp\_MessageMSGCB()函数、SampleApp\_SendPeriodicMessage()函数和SampleApp\_SendFlashMessage()函数。

#### SampleApp\_Init()函数

**调用路径：**

该函数由osalInitTasks()函数调用。

路径：main.c -> osal\_init\_system() -> osalInitTasks() –> SampleApp\_Init()。

**主要功能：**

该函数主要是实现用户自定义任务的初始化，包括任务ID、网络状态、传输序列号、数据发送地址、端点等信息初始化。包括广播、组播和单播的设置。

#### SampleApp\_ProcessEvent()函数

**调用路径：**

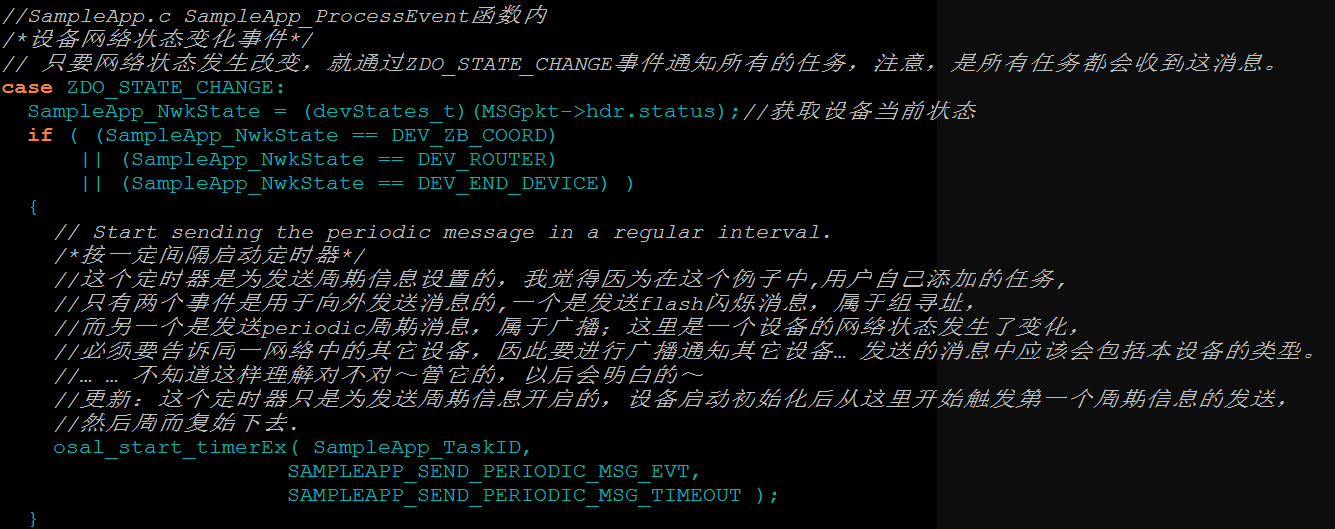
该函数由taskArr[]数组中定义，由osal\_start\_system()函数在轮询任务中调用。

路径：main.c -> osal\_start\_system() -> SampleApp\_ProcessEvent()。

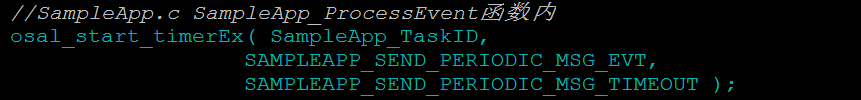
**主要功能：**

实现具体事件处理，包括接收数据和发送数据。在接收数据上，主要是接收系统消息事件，包括按键事件、接收消息事件、网络状态改变事件和定时事件等。按键处理事件主要调用SampleApp\_HandleKeys()函数，接收消息事件主要调用SampleApp\_MessageMSGCB()函数，网络状态改变事件中需要定时广播告诉所有的设备网络状态的变化。在发送数据上,主要是周期性发送数据，利用SampleApp\_SendPeriodicMessage()函数周期性发送数据，可以使传感器定时采集和上传数据。**周期性发送的数据的设备和时间在网络状态改变事件中处理。**

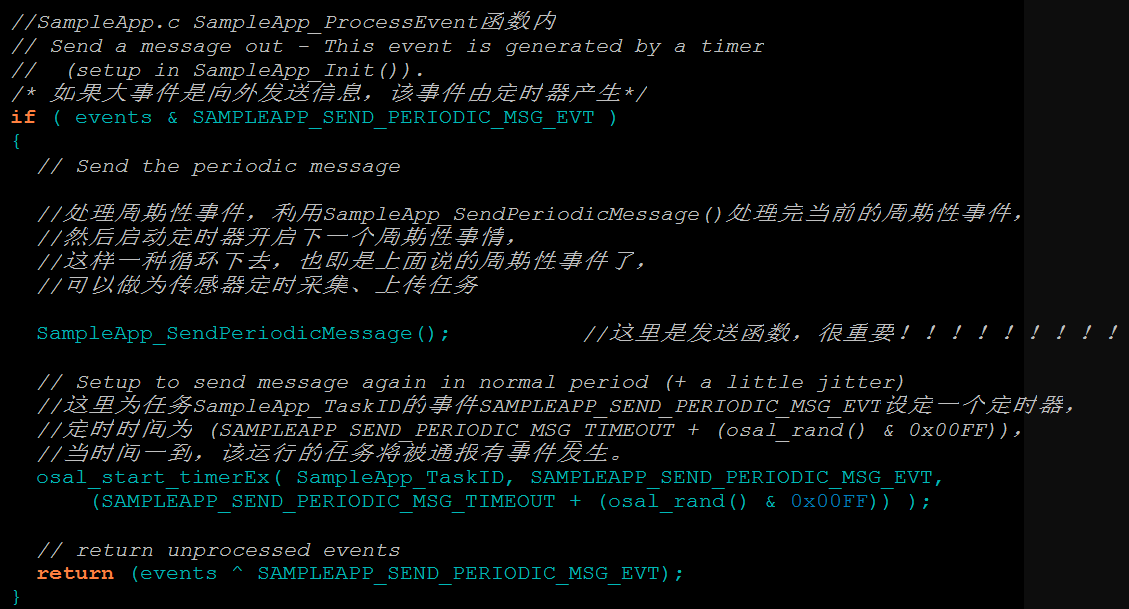
**注：周期性发送的数据的设备和时间在网络状态改变事件中处理。**

****

**If条件中可以定义需要定时发送数据的设备类型，例如协调器、路由器和终端设备。一般来说，协调器是不需要定时发送数据的，往往终端设备是需要定时采集传感数据进行发送给协调器。这里可以在if语句中自定义需要周期发送的设备类型。**

****

**这个就是设置定时发送的函数，第一个是自定义任务的ID，第二个就是周期性发送数据的事件编号，查看定义是0x01,而第三个参数就是设置周期性发送数据的时间间隔，这里是5000，也就是5s定期发送数据。那么发送数据在哪里执行呢？**

****

**定时发送数据时，先比对事件编号，然后利用数据发送函数SampleApp\_SendPeriodicMessage()函数发送数据，需要注意的是发送完以后发送完以后又开启了一个5s发送数据的周期性函数。这里可以做的工作就是周期性发送传感器采集的数据。**

#### SampleApp\_HandleKeys()函数

主要用于按键事件处理。

#### SampleApp\_MessageMSGCB()函数

**调用路径：**

该函数由SampleApp\_ProcessEvent()函数调用。

路径：main.c -> osal\_start\_system() -> SampleApp\_ProcessEvent() –> SampleApp\_MessageMSGCB()。

**主要功能：**

用于实现数据的接收，通过判断簇ID来执行广播或组播的接收信息。需要注意的是afIncomingMSGPacket\_t类型的结构体和其中的afMSGCommandFormat\_t类型结构体。这在6.3数据的接收中讲解。

#### SampleApp\_SendPeriodicMessage()函数

**调用路径：**

该函数SampleApp\_ProcessEvent()函数调用时用作周期性发送数据，当然也可以在其他地方灵活调用，例如接收数据SampleApp\_MessageMSGCB()函数用于发送反馈数据等。

路径：main.c -> osal\_start\_system() -> SampleApp\_ProcessEvent() –> SampleApp\_SendPeriodicMessage()。

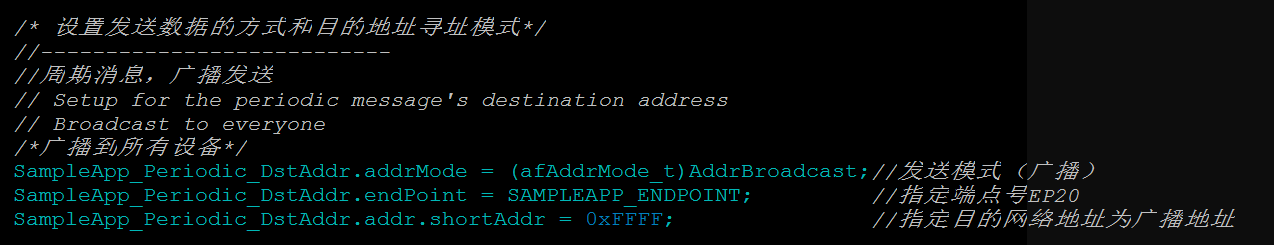
main.c -> osal\_start\_system() -> SampleApp\_MessageMSGCB() –> SampleApp\_SendPeriodicMessage()。

还可以有其他调用路径。

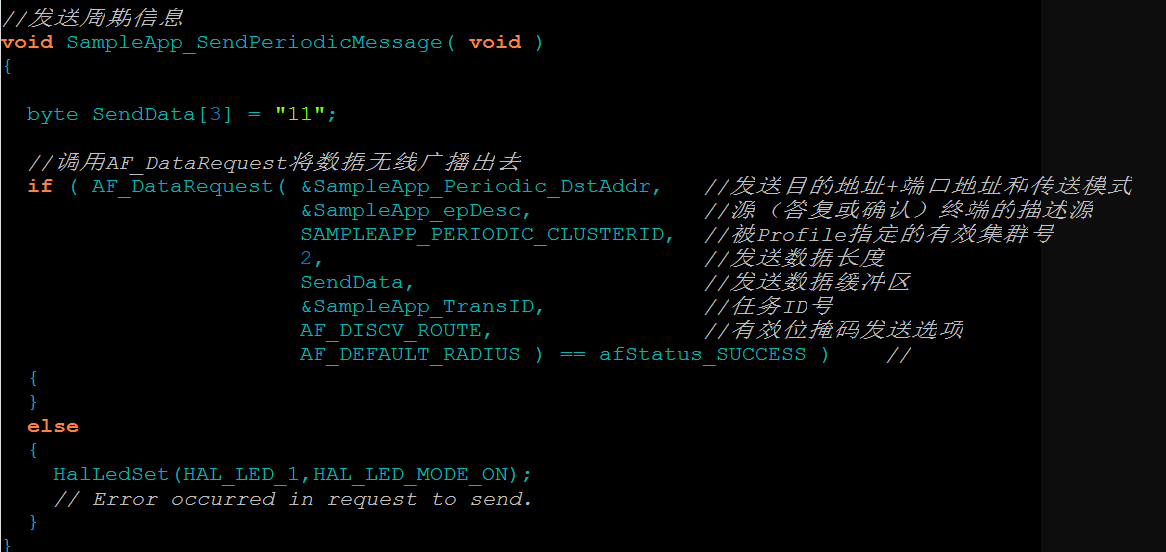
**主要功能：**

可以用于周期性发送数据或者单次发送数据。需要注意的是，这个函数用于广播发送数据，发送的数据任何设备都可以接收到。

注：查看SampleApp\_Init()函数中的定义



再看SampleApp\_SendPeriodicMessage()函数



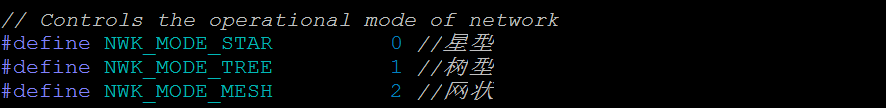
AF\_DataRequest的第一个参数就是广播地址参数。第三个参数就是在接收函数SampleApp\_MessageMSGCB()用来判断数据是广播的还是组播的等等。

#### SampleApp\_SendFlashMessage()函数

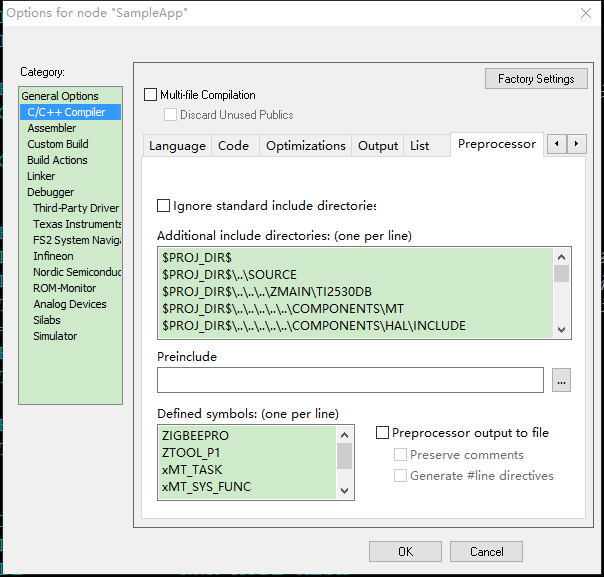
该函数只要用于组播发送数据。

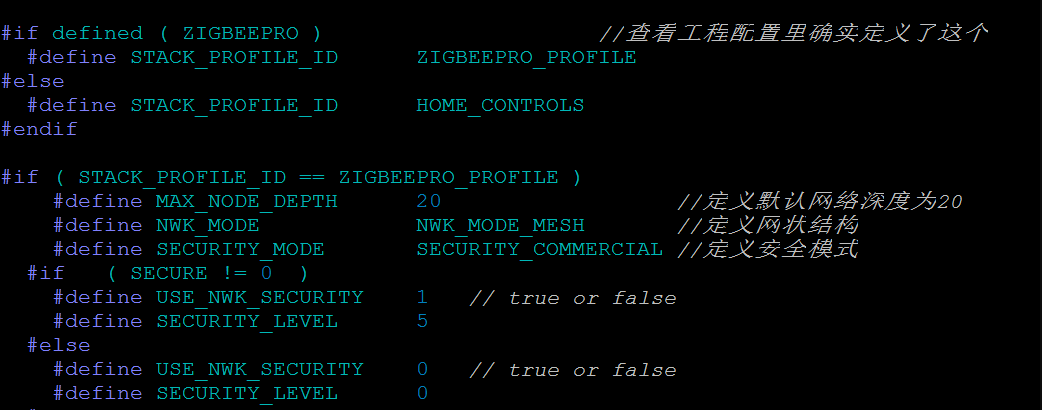
# NWK文件夹(这个如果要配置比较难，暂时不深入研究)

该文件夹中的nwk\_globals.h定义了网络拓扑结构类型

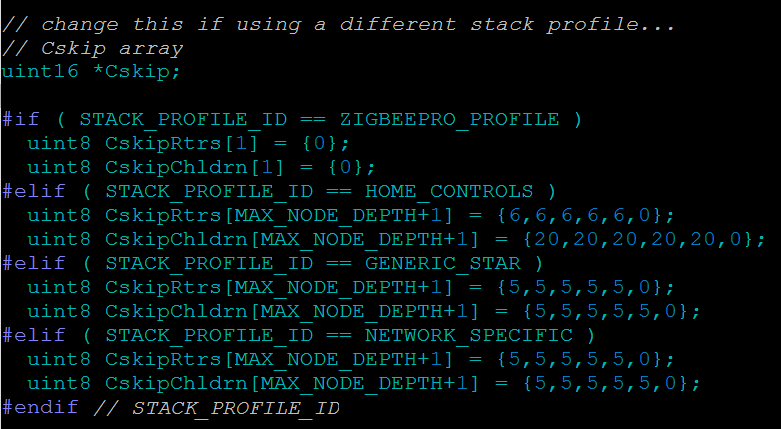


查看配置项宏定义了ZIGBEEPRO



  
这里如何把网络类型设置为星型结构？需要深入研究。

节点地址的分配和深度解析，看nwk\_globals.c



CskipChldrn[]中的每一个元素表示协调器或路由器可以连接的子节点的最大个数，这里还不是很清楚CskipChldrn[1] = {0}为什么是这样？原因是这里默认设置为网状网络结构，该结构的网络地址由协调器随机分配，所以设置默认值即可。但是在树状网络中，就不能这样了，要按照一定的算法分配地址。

# HAL文件夹

主要包含三个文件夹，Common、Include和Target文件夹。

### Common文件夹

Common文件夹中主要的文件是hal\_drivers.c。

#### hal\_drivers.c文件

该文件主要包括硬件初始化Hal\_Init()函数、硬件抽象层驱动初始化HalDriverInit()函数和硬件抽象层事件处理函数Hal\_ProcessEvent()。

Hal\_Init()函数：为硬件抽象层注册任务ID。在osalInitTasks()函数中被调用。

HalDriverInit()函数: 包括了定时器、ADC、DMA、Flash、AES、LED、UART、KEY和SPI等的初始化。在main函数中被调用。

Hal\_ProcessEvent()函数: 任务处理函数，当然是在taskArr[]数组中被定义。主要实现硬件抽象层的各种事件处理，比如系统消息事件、LED闪烁事件、按键事件和睡眠模式事件等。可以注意比较该文件中的程序和SampleApp\_ProcessEvent()事件处理函数的关系。

### Include文件夹

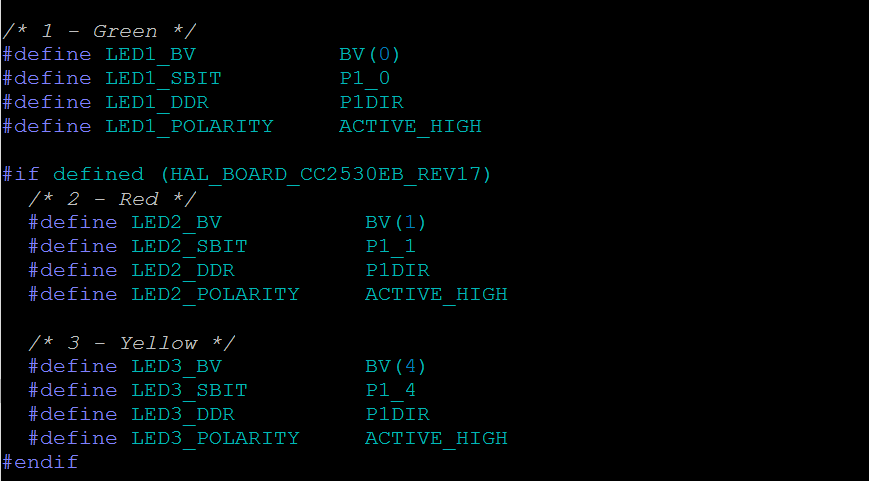
### Target文件夹

该文件夹主要包括CC2530EB文件夹（Config、Drivers、Includes）。

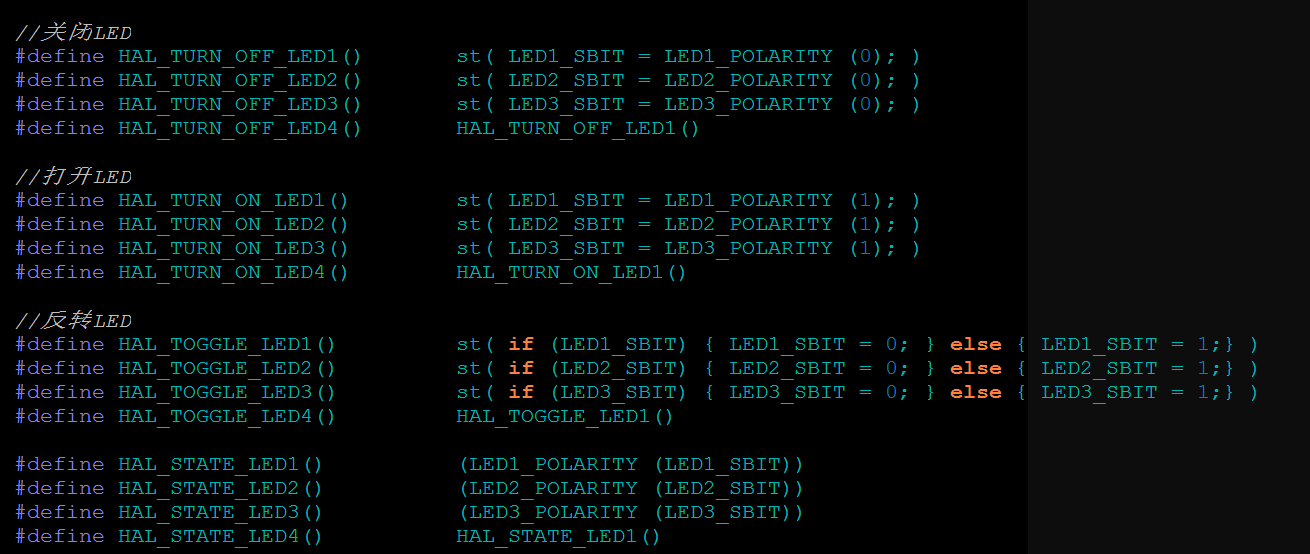
#### Config文件夹

该文件夹中有一个hal\_board\_cfg.h文件。

hal\_board\_cfg.h：为硬件资源LED等进行配置，在官方的协议栈中定义了3个LED，分别是P1\_0，P1\_1和P1\_4引脚。

****

对LED等进行控制

****

#### Drivers文件夹

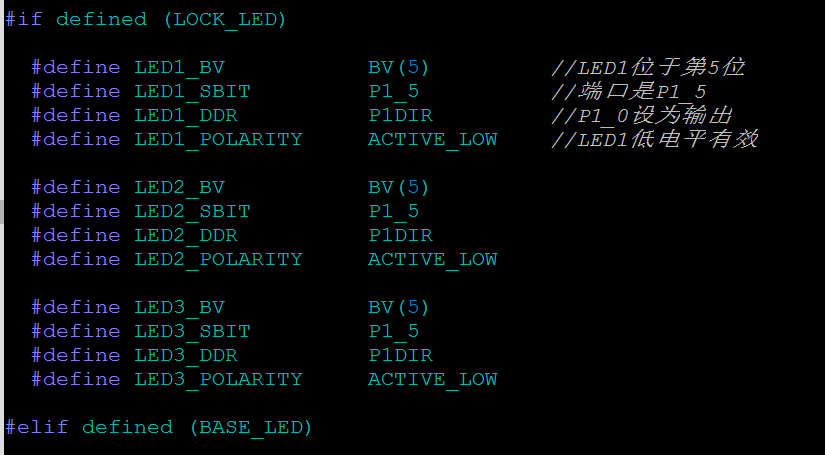
定义了硬件资源的驱动函数文件，包括LED、ADC、KEY、定时器、串口、DMA以及Flash等。

### LED驱动修改

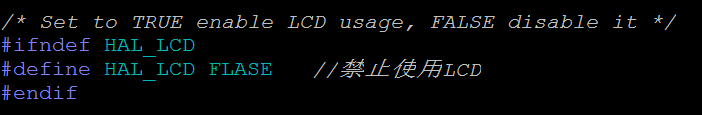
#### 门锁LED驱动修改

在预编译选项中加入LOCK\_LED，更改hal\_board\_cfg.h中的LED宏定义

需要注意的是Z-Stack默认有3个LED，门锁LED只有一个，为了方便起见，设置为3个一样。



需要注意可能改配置与其他的驱动端口有冲突，可以屏蔽其他的驱动端口，仍然在hal\_board\_cfg.h中



可以禁止LCD的端口。

# ZStack应用

# 数据传输

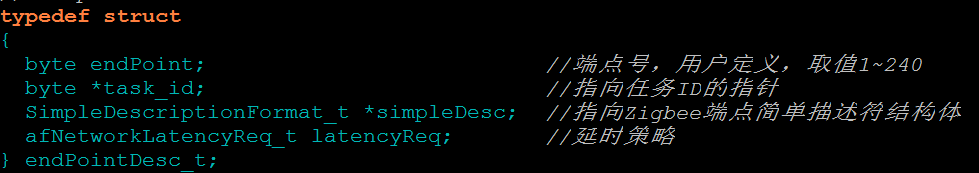
### 端点

在Z-Stack中最多可以实现256个端点，端点0被ZDO使用，端点255被用于广播，端点1~240倍应用层分配，用于数据的发送接收以及绑定。端点241~254为预留扩展端点。

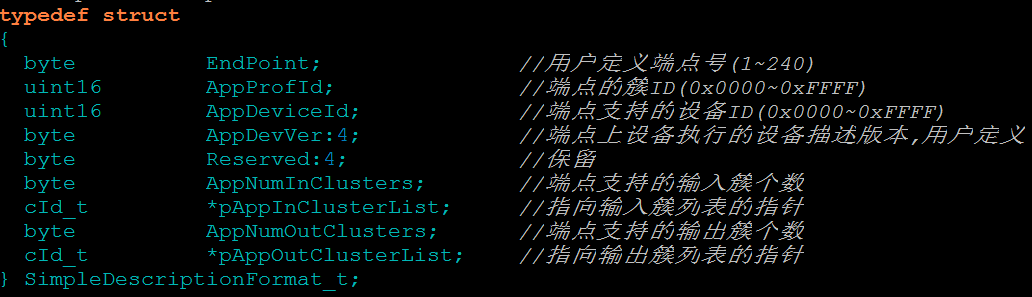
端点1~240的作用：在数据发送和接收时使用了端点的概念，在数据发送方和接收方使用的端点必须一致，否则数据发送不成功。在节点绑定过程中，必须注册一个或多个端点来进行绑定表的建立。

端点的意思就是在一个设备中，最多有240个应用对象！相同端点的应用对象可以互相通信，不同端点的应用对象不能相互通信。

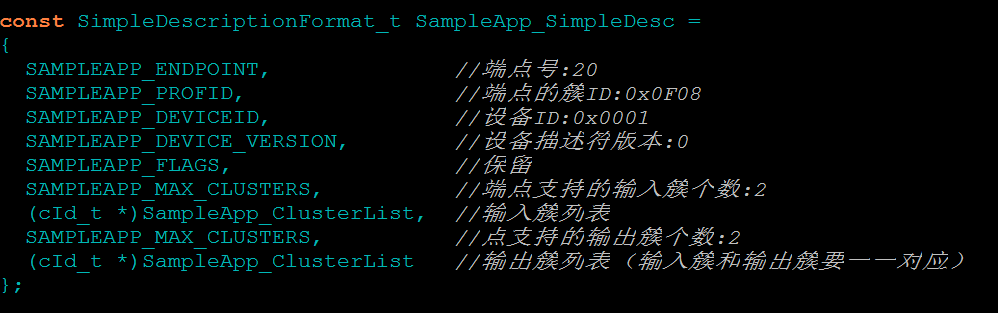
端点的实现由端点描述符来实现，查看AF.h



Zigbee端点简单描述符(AF.h)

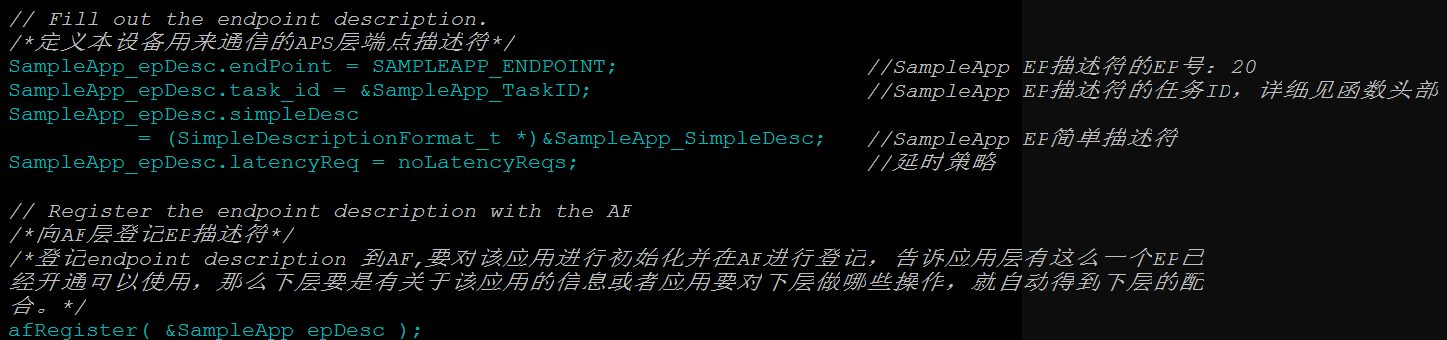


Ti官方的SampleApp.c中的默认Zigbee端点简单描述符



需要注意的是输入簇和输出簇要一一对应，即发送方的输出簇要和接收方的输入簇一致。在TI官方的SampleApp中发送端和接收端的输入/输出簇列表定义如下(SampleApp.c)：  


端点定义了以后需要在Z-Stack应用层注册端点，端点注册了以后才能使用Zigbee协议栈进行数据的发送和接收，详见SampleApp\_Init()函数



### 数据的发送

#### AF\_DataRequest()发送函数

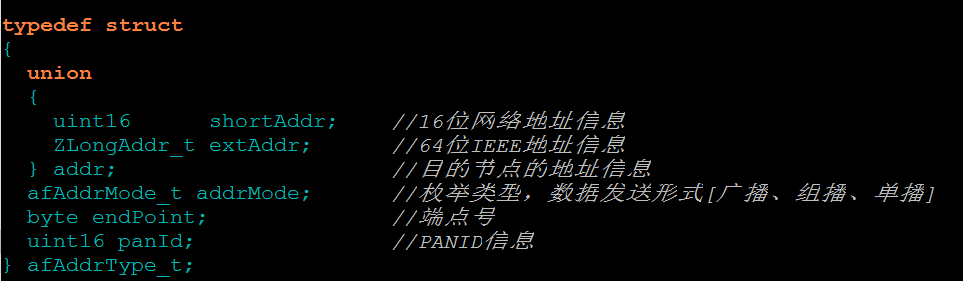
数据的发送函数详见SampleApp.c中的SampleApp\_SendPeriodicMessage()函数和SampleApp\_SendFlashMessage()函数。这两个函数都调用了AF\_DataRequest()函数进行发送。例如广播发送函数



查看第一个参数目的节点的地址信息的定义(SampleApp.c)

afAddrType\_t SampleApp\_Periodic\_DstAddr; //这是广播的目的节点

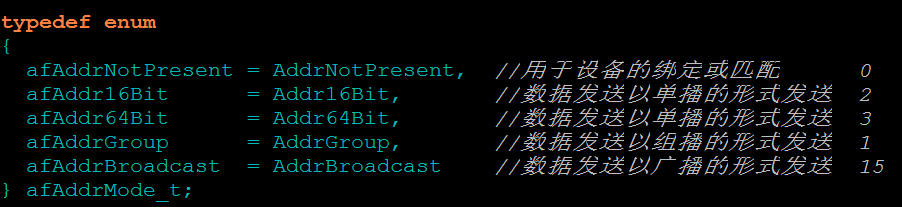
afAddrType\_t结构体如下



这里说明一下，使用64位物理地址和16位网络地址都可以发送数据，[64位](https://www.baidu.com/s?wd=64%E4%BD%8D&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuH0LPWu-nj61nWKBujRs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErH6Yn10kP1nzP1nsPW01rjb4)的地址只能用于单跳寻址，路由的寻址只能用[短地址](https://www.baidu.com/s?wd=%E7%9F%AD%E5%9C%B0%E5%9D%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuH0LPWu-nj61nWKBujRs0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErH6Yn10kP1nzP1nsPW01rjb4)。

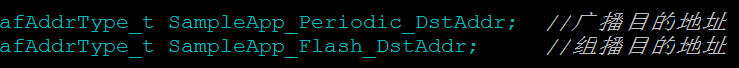
但是使用16位网络地址可以采用路由算法降低功耗，当然对于没有路由的设备是无所谓的，但是也最好使用16位网络地址，发送数据少。其次如果是需要协调器对某个终端设备点对点通信？

查看afAddrMode\_t枚举类型



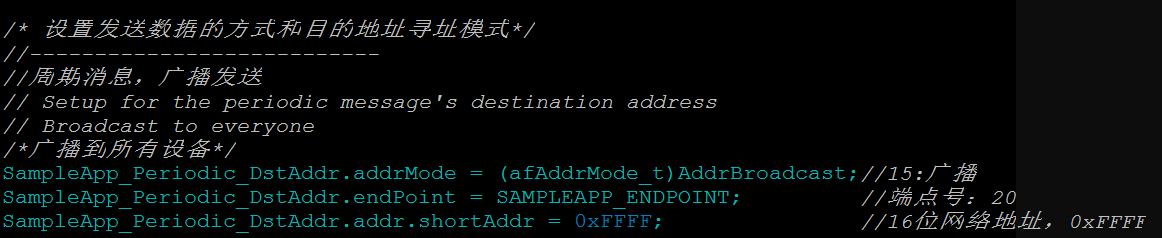
#### 地址信息的设置

6.2.1详细介绍了afAddrType\_t结构体，在SampleApp.c中定义了两个目的地址



##### 广播地址的设置(SampleApp.c SampleInit())

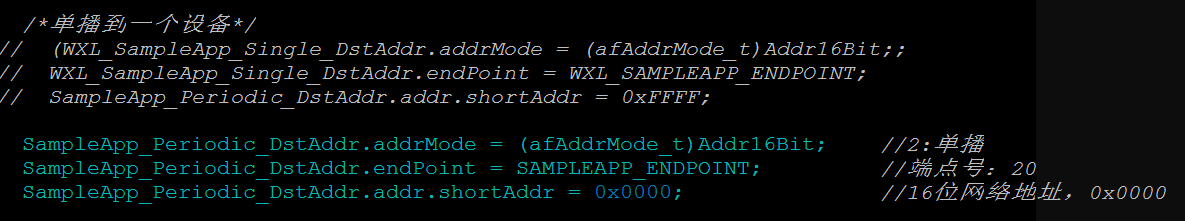
通过变量afAddrType\_t SampleApp\_Periodic\_DstAddr进行设置

  
数据广播的地址afAddrType\_t中的addr取值有4种形式

|  |  |
| --- | --- |
| 地址 | 说明 |
| 0xFFFF | 广播模式地址的默认值，广播至网络中所有设备，对于睡眠的设备，数据包将保留在父节点，当节点被激活后主动到此节点的父节点查询此数据包是否丢失。 |
| 0xFFFD | 传送至网络中所有打开接收功能的空闲节点。 |
| 0xFFFC | 发送至网络中的所有路由器，包括协调器。 |
| 0xFFFE | 应用层不指定目标设备，通过协议栈读取绑定表来获得目标设备的短地址。 |

##### 6.2.2.2 单播地址的设置(SampleApp.c SampleInit())

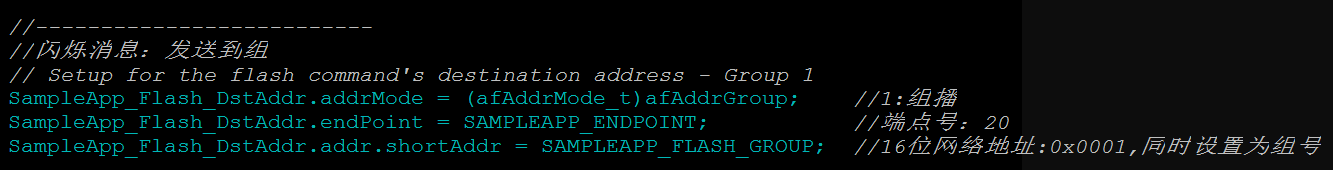
单播和广播一样，通过SampleApp\_Periodic\_DstAddr(也可以重新开辟一个变量，这里方便起见使用了组播的变量进行设置)进行设置。



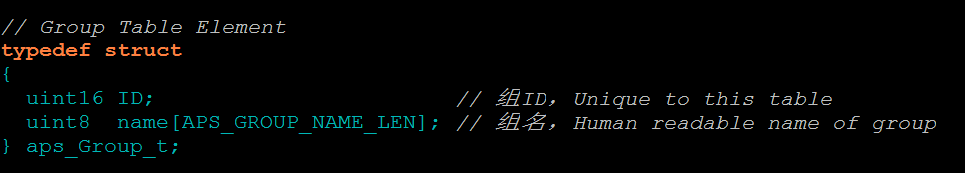
设备的目的地址设置为16位固定的网络地址，例如向协调器发送信息，协调器的网络地址为固定值0x0000,因此单播的地址就是0x0000(除了协调器的地址是固定的，其他设备的网络地址都是随机分配的，所以只能通过函数获取，有专门的获取地址函数，发送时协议会发送网络地址？然后才能根据网络地址反馈？)。还有就是多个协调器的情况下，如何组网？？？分布在不同的楼层？？？信道设置不一样？？？

##### 6.2.2.3 组播地址的设置(SampleApp.c SampleInit())

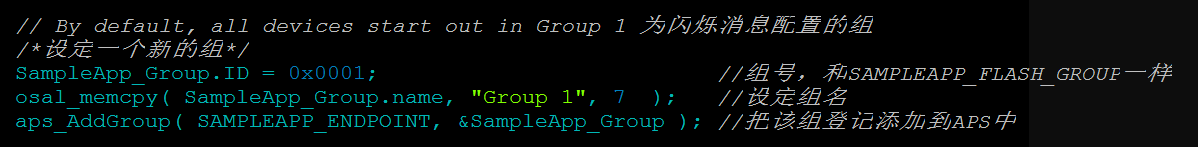
通过变量afAddrType\_t SampleApp\_Flash\_DstAddr进行设置



数组的组播需要设置组ID,组名，在aps\_groups.h文件中定义结构体



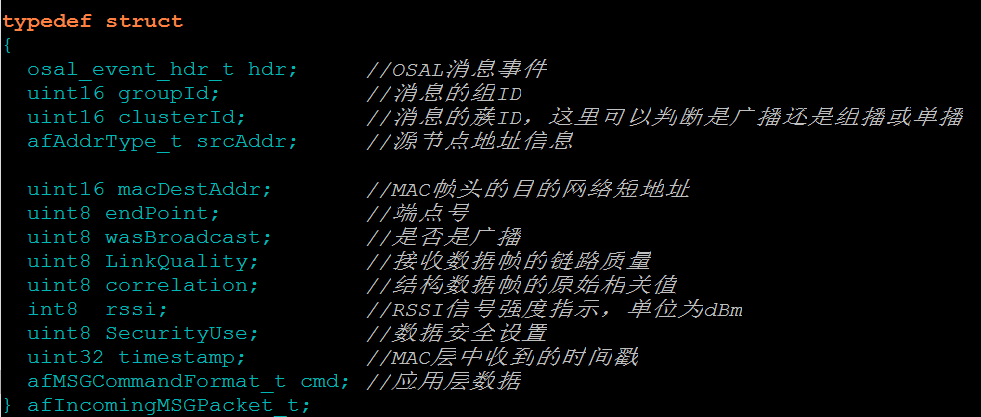
组ID的设置范围为0x0000~0xFFFF，设置完成后需要将设置信息注册到应用支持子层APS中，具体设置在SampleApp.c的SampleApp\_Init()中



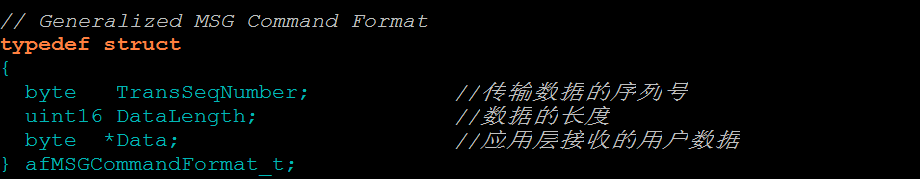
### 数据的接收

查看3.2.2 SampleApp\_ProcessEvent()函数，定义的第一个变量是一个接收消息的结构体afIncomingMSGPacket\_t \*MSGpkt; //接收到的消息，消息包

查看这个结构体(AF.h)



afAddrType\_t这个结构体可以查看6.2.1发送函数中的介绍，可以知道发送的时候发送了目的节点的地址信息，接收的时候同样可以知道源节点的地址信息。查看最后一个应用层数据结构体 afMSGCommandFormat\_t cmd; //应用层数据，仍然在AF.h



在应用层数据的接收通过osal\_msg\_receive()函数(OSAL.c)实现，数据接收完成之后在3.2.4节接收函数SampleApp\_MessageMSGCB()中进行处理，主要有两步

1. 判断事件类型，包括按键事件、AF层接收消息事件（消息到来响应事件）、网络状态改变事件等。
2. 如果接收到的消息是某一个事件类型，将调用相应事件的处理函数，如果是AF层接收消息事件，还会调用SampleApp\_MessageMSGCB()函数执行，并判断数据的簇ID是广播、组播还是单播。

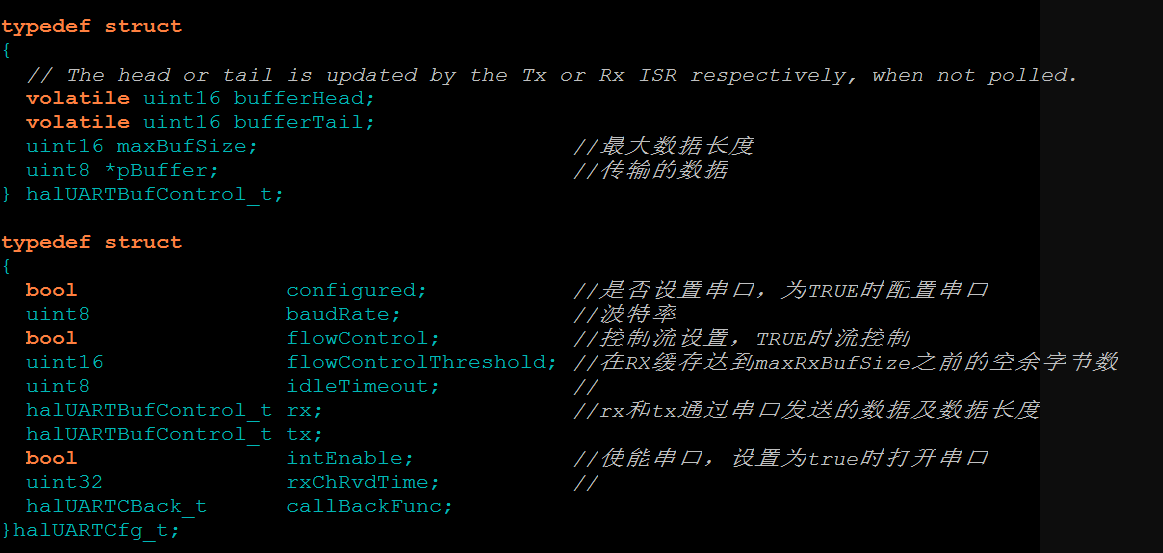
# 串口应用

串口的主要作用当然是给PC发送数据，所以一般来说是协调器会用到的功能。

串口的相关配置在HAL层的hal\_uart.h和hal\_uart.c中。具体路径

HAL -> Target -> CC2530EB -> Drivers -> hal\_uart.c

在hal\_usrt\_h中查看结构体

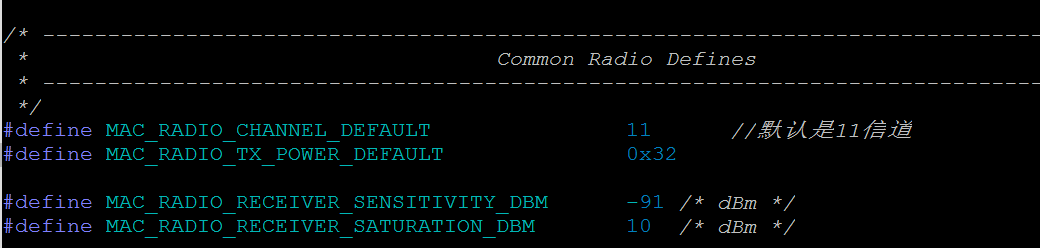


具体的应用查看MT层MT\_USRT.c函数。

# 绑定和匹配（暂时不研究）

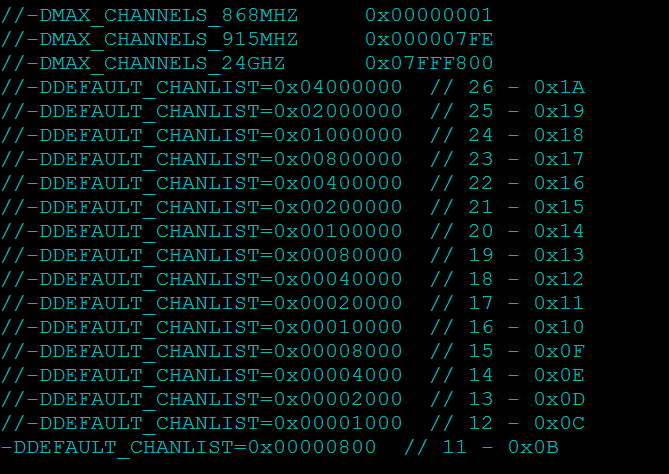
# 信道的设定

查看 MAC -> Low Level -> mac\_radio\_defs.h



也可以通过设置信道设置

更具体的设置可以查看Zigbee技术与实训教程 P186. 查看Tools -> f8Config.cfg文件 注释11开启25，避免与WI-FI冲突



# 协调器网络的建立

查看Zigbee2007/PRO协议栈实验与实践 P92/94

这里还说明了协调器的16位网络地址是0x0000.

### 网络的形成

查看ZDApp\_event\_loop任务下的ZDO\_StartDevice()函数，对于协调器创建网络，对于终端和路由器加入网络。该工作是通过操作系统管理完成的，我们不需要考虑。

# 设置操作系统的事件

触发事件有三种情况：1.外部中断 2.定时器 3.对设备进行轮询

外部中断和定时器不需要进行干预，但是例如串口这样的设备，需要使用Hal\_ProcessPoll()进行轮询。

Zigbee技术实训教程 P183

Osal\_set\_event()设置任务事件的函数

例如中断的触发事件

查看Zigbee2007/PRO协议栈实验与实践 P219/216

查看Zigbee技术原理与实战 P234对任务和事件的理解

一个任务在处理过程中是不可以被中断的，必须前一个任务被处理完毕后才能够执行任务切换，每个任务都只有调用osal\_set\_event()函数后才能够得到处理。

### 操作系统流程图.

查看Zigbee2007/PRO协议栈实验与实践 P254

### Z-Stack执行流程

查看Zigbee技术原理与实战P214

### Z-Stack重要术语关系

查看Zigbee技术原理与实战P219 包括节点、端点、设备以及群集等之间的关系，这里可以结合ZigBee无线网络与收发器P85。并说明了端点和TCP/IP的端口的相似性。

### Z-Stack获取父节点地址和自身地址

查看Zigbee技术原理与实战 P230

### Packet Sniffer入网过程分析

查看Zigbee技术原理与实战 P253

### 数据测试过程

查看Zigbee技术原理与实战 P297

### Zigbee网络多信道调度

查看Zigbee技术原理与实战 P316 建立多个PAN网络！

# 低功耗

### PM2休眠

默认情况下电源管理设置为正常工作模式，如果要设置轻度休眠，设置为PM2模式：

第一步：添加编译选项：POWER\_SAVING

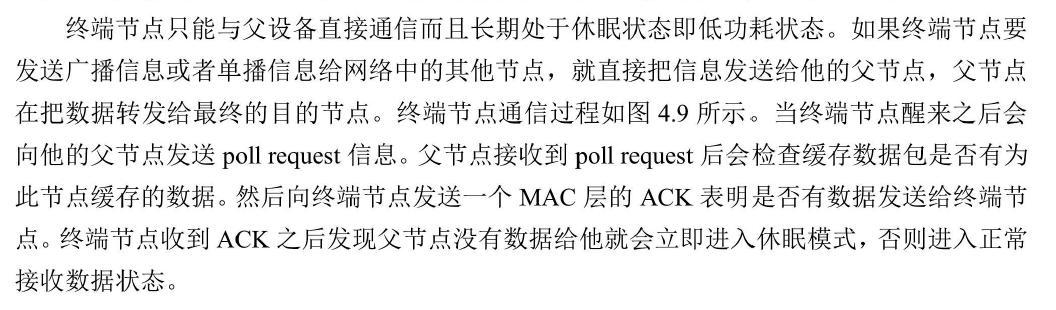
说明：默认情况下，Z-stack里终端设备的电源管理是关闭的。添加编译选项POWER\_SAVING以开启此功能。

第二步：在f8wConfig.cfg里设置：-DRFD\_RCVC\_ALWAYS\_ON = FALSE

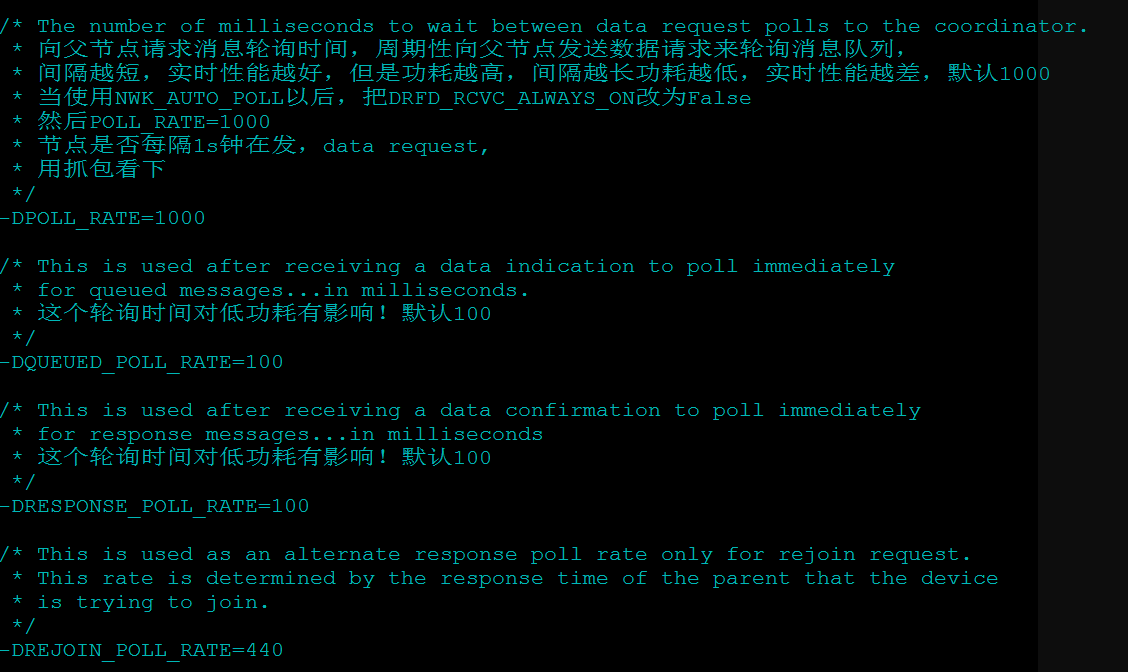
说明：如果设置为TRUE会导致MAC层在休眠时不关闭，从而会阻止设备进入休眠模式。

第三步：设置DPOLL\_RATE、DQUEUED\_POLL\_RATE、DRESPONSE\_POLL\_RATE以及DREJOIN\_POLL\_RATE参数。例如要使终端设备进入深度休眠，则需要设置三个参数都为0。

说明：低功耗的设备如何接收协调器的数据呢？



以上说明有问题，poll request其实是data request,发送的时间间隔就是DPOLL\_RATE设置的时间。

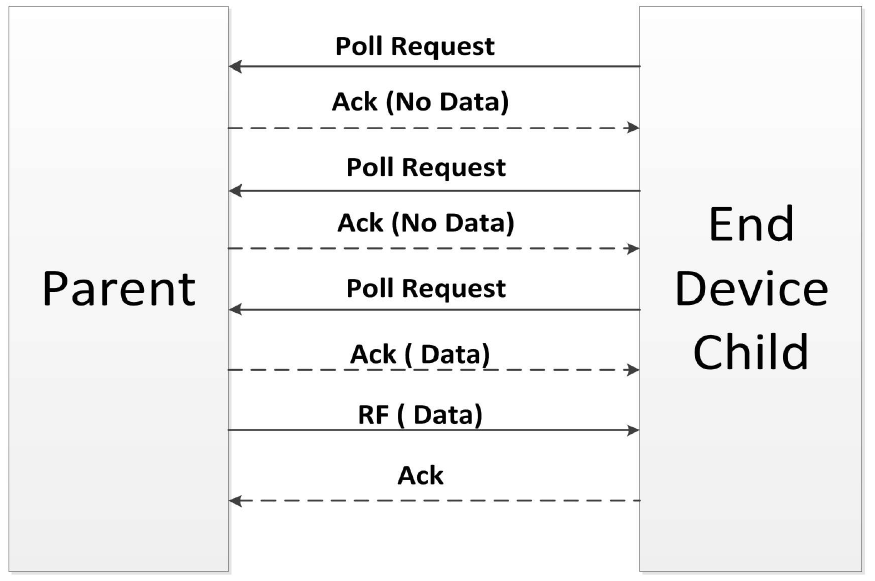


三个参数说明：

DPOLL\_RATE：终端设备向协调器发送的数据请求的时间间隔，默认设置为1s。(查看编译选项的时候我们发现有NWK\_AUTO\_POLL选项，该项设置就是为了使终端设备能够自动从父节点获取数据)。

DQUEUED\_POLL\_RATE：在收到数据指示后，就会向父节点请求消息。**这个应该是终端每隔一段时间向父节点发送数据。**

DRESPONSE\_POLL\_RATE：在收到数据确认指示后，就会向父节点请求响应消息。**这个时间是每隔一段时间向父节点发送ACK消息**



### 休眠时间的取值

休眠时间是Z-Stack协议栈自行计算的，取决于OSAL定时器下一次溢出的时间timeout,下一次溢出的时间一般是取所有打开的定时器中最小的时间作为睡眠时间，例如DPOLL\_RATE数据请求轮询时间间隔设置为1s,设定的睡眠时间为6s,那么1s就是睡眠时间。