1. **调整发送周期：**

在SampleApp.c的uint16 SampleApp\_ProcessEvent( uint8 task\_id, uint16 events )中调用了这么一条语句：

osal\_start\_timerEx( SampleApp\_TaskID, SAMPLEAPP\_SEND\_PERIODIC\_MSG\_EVT,

(SAMPLEAPP\_SEND\_PERIODIC\_MSG\_TIMEOUT + (osal\_rand() & 0x00FF)) );

其中，SAMPLEAPP\_SEND\_PERIODIC\_MSG\_TIMEOUT指的就是你的发送周期，它可以在SampleApp.h中定义为小于510s（即510000）的数

经测试，发送周期不能调到大于一分钟，即60000，超过这个值，默认一分钟发一次。

1. **调整发送频率：**

参考<http://m.blog.csdn.net/blog/u011352311/42011539>

Z-stack提供了两种sleep模式，LITE和DEEP（PM2/PM3）。当系统需 要唤醒去执行一些周期性的事件时，使用LITE sleep；当系统没有被定制周期事件而是通过外部中断（如按键）来唤醒设备，使用DEEP sleep。例如带有传感器的终端设备，它需要周期性的唤醒发送采集的数据，又如远程控制终端设备通过按键唤醒来发送一信息，它们的大部分时间都是处于 sleep，以降低功耗。

第一步：添加编译选项：POWER\_SAVING

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

-DRFD\_RCVC\_ALWAYS\_ON=FALSE

第三步：根据需要，在f8wConfig.cfg里设置三个参数：

-DPOLL\_RATE=1000

-DQUEUED\_POLL\_RATE=100

-DRESPONSE\_POLL\_RATE=100

Z-stack工程终端设备默认情况下为电源管理关闭，自动轮询消息这一功能是开启的。这里有三个轮询选项，每一个都由一个不同的时间延迟参数控制。当电源管理功能开启后（添加POWER\_SAVING），任一个轮询选项的设置都会影响到睡眠模式。时间延迟的设置不能用于DEEP skeep中的轮询，因此限制了降低功耗。这三个轮询选项分别如下：

**Data Request Polling—周期性向父节点发送数据请求来轮询消息队列。轮询的时间间隔由NLME\_SetPollRate()或gNWK\_POLL\_RATE设定，如果事先没有使能它，那么在调用时就立即进行轮询。**

**赋值下载到程序后，终端就会按照这个速率去发data request，我们可以通过调用函数 NLME\_SetPollRate()来立即改变终端的速率。尤其在低功耗模式下，这个参数的值非常的影响功耗，为了达到深度睡眠，这个值应该设置为零。**

**改成0以后，父设备是无法发数据给子设备，因为父设备给子节点的数据时Indirect message，需要子设备去data request获取数据。**

**Queued Data Polling—在收到数据指示后，就会向父节点请求消息。这个时间间隔可由NLME\_SetQueuedPollRate()或gQUEUED\_POLL\_RATE设定。这个应该是终端每隔一段时间向父节点发送数据。**

**Response Data Polling—在收到数据确认指示后，就会向父节点请求响应消息，这个时间间隔可由NLME\_SetResponsePollRate()或gRESPONSE\_POLL\_RATE设定。这个时间是每隔一段时间向父节点发送ACK消息。。。。。。这个我们可以明天来测试一下**

函数说明：

NLME\_SetPollRate()——设置/改变网络检测速率，仅终端设备可用。

NLME\_SetQueuedPollRate()——设置/改变队列检测速率，仅终端设备可用。

NLME\_SetPollRate()——设置/改变响应检测速率，仅终端设备可用。

**如果只是使用默认的轮询频率进入睡眠态，则只能进入LITE sleep。为了进入DEEP sleep则必须将gNWK\_POLL\_RATE设为0（也就是通过设置Data Request Polling 为零可以使用NLME\_SetPollRate（）函数来实现），不让它反复轮询。这个样子就不会采集数据了。（在f8wConfig.cfg中已修改）**

设置这个三个选项可以实现多种轮询方式，例如，对于一个不需要接收消息的设备，在它加入网络后，就将这三个选项都设为0。如果APS层使用了ACK，则必需确保在消息发送后到收到ACK这一段时间内，轮询是使能的。

在有的系统中，可能需要使用可变的轮询频率，根据具体应用而进行设置。  
对于这个功耗管理最好参考TI的官方资料，顺便还说一句上面那种情况一般是出现在ZC和ZD通信时才考虑.上面的设置方式还不能最大的降低功耗，好像只能进入到PM2模式，如果需要进入PM3模式还需要调用HAL\_SLEEP（）这个函数，具体怎么弄可以看协议栈代码。对于低功耗的数据采集，有些网友不知道如何保存之前的网络，也就无法知道是那个模块在采集数据，对于这个，我觉得最好可以加入 NV\_INIT,NV\_RESTORE编译。还有一个大伙可能会出现的，就是有些人把虽然保存了网络信息也组好了网络，但是发现用TI的smartRF给协调器烧写hex文件之后无法入网了，这是为什么呢？因为smartRF把NV擦除了，所以协调器重启建立PANID所以无法入网。有时甚至会出现PANID冲突，对于这个解决办法是用IAR BEGUG 工程里面选择不擦除NV就OK了。

3. 在hal\_led.c中的HalLedExitSleep(void)函数中注释：（在协议栈中修改）

HalLedOnOff(HalSleepLedState, HAL\_LED\_MODE\_ON);

/\* Restart - This takes care BLINKING LEDS \*/

HalLedUpdate();

1. **测试灯的使用情况：**

在Led.c 中修改HalLedInit(void)，将HalLedSet (HAL\_LED\_ALL, HAL\_LED\_MODE\_OFF);变为HalLedSet (HAL\_LED\_ALL, HAL\_LED\_MODE\_ON); （协议栈修改）

在ZMain.c 中添加：

/\* 张立超测试\*/

HAL\_TURN\_ON\_LED1(); //LED1 open

HAL\_TURN\_OFF\_LED2();

HAL\_TURN\_OFF\_LED3();

/\* 张立超测试\*/

1. **修改部分：**

搞定灯，现在是发一条数据LED2改变当前状态。

进入休眠，LED1灭，唤醒LED1亮。