ZigBee和IEEE802.15.4是基于标准的协议，为无线传感器网络应用程序提供所需的网络基础设施。802.15.4定义物理层（PHY）和媒体访问控制（MAC）层，ZigBee定义网络层（NWK）和应用层（APL）。

对于传感器进行无线组网应用而言，最主要的设计就是要求电池电量够长时间使用，成本低，网络稳定等问题，这样才能支持在一个应用环境中设备相互通信传输数据。

2.4GHz的CC253x片上系统解决方案适合于广泛的应用。它们可以很容易建立在基于IEEE802.15.4标准协议（RemoTI网络协议、TIMAC软件和用于ZigBee兼容解决方案的Z-Stack软件）上面，或是专门的SimpliciTI网络协议上面。但是它们的使用不限于这些协议。

CC2530设备系列使用的8051 CPU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个输入的扩展中断单元。USART 0和USART 1每个被配置为一个SPI主/从或一个UART。它们为RX和TX提供了双缓冲，以及硬件流控制，因此非常适合于高吞吐量的全双工应用。每个都有自己的高精度波特率发生器，因此可以使普通定时器空闲出来用作其他用途。

低功率运行采用不同的运行方式或供电方式。超低功耗操作的实现通过关闭电源模块来避免静电（泄漏）。使用门控时钟和闭合振荡器也可以降低功耗。

有五种不同的操作模式（电源模式），分别称为主动模式、空闲模式、PM1、PM2和PM3。主动模式是通用模式，而PM3的功耗最低。图中给出了不同供电方式对系统运行的影响，给出了电压调节器和振荡器的选择。

|  |  |  |  |
| --- | --- | --- | --- |
| 供电模式 | 高频振荡器 | 低频振荡器 | 稳压器（数字） |
| 配置 | A 32MHz XOSC  B 16MHz RCOSC | C 32KHz XOSC  D 32KHz RCOSC |  |
| 主动/空闲模式 | A或B | C或D | ON |
| PM1 | 无 | C或D | ON |
| PM2 | 无 | C或D | OFF |
| PM3 | 无 | 无 | OFF |

主动模式：完全功能模式。稳压器的数字内核开启，16 MHz RC 振荡器或32 MHz 晶体振荡器运行，或者两者都运行。32 kHz RCOSC 振荡器或32kHz XOSC 运行。

空闲模式：除了CPU 内核停止运行（即空闲），其他和主动模式一样。

**PM1**：稳压器的数字部分开启。32 MHz XOSC和 16 MHz RCOSC都不运行。32 kHz RCOSC或 32 kHz XOSC

运行。复位、外部中断或睡眠定时器过期时系统将转到主动模式。

**PM2**：稳压器的数字内核关闭。32 MHz XOSC和 16 MHz RCOSC都不运行。32kHz RCOSC或 32 kHz XOSC

运行。复位、外部中断或睡眠定时器过期时系统将转到主动模式。

**PM3**：稳压器的数字内核关闭。所有的振荡器都不运行。复位或外部中断时系统将转到主动模式。

在PM2/PM3下，POR是活跃的，但BOD是断电的，这使得电压管理有限。如果PM2/PM3期间的电压降至低于1.4V，温度是70°C 或更高，并且然后重新进入主动模式之前，回到合适的运行电压，寄存器和RAM

在PM2/PM3 下保存的内容可能会改变。因此，在设计系统电压时要小心，以确保这种情况不会发生。电压可

以通过进入主动模式进行精确的定期监控，因为如果电压低于大约1.7V 就触发一个BOD 复位。

首先，需注意：ZIGBEE的本次开发是基于CC2530硬件，使用的是TI公司的Z-Stack协议栈。该协议栈为半开源。

1. 讲解下协调器与终端的“.c”程序：

4个工程都是要找到main()函数作为函数入口。Main()函数位于ZMain文件夹里的ZMain.c文件中。在main()里面可以看到大部分都是用于配置板子上ZIGBEE的网络层的初始化函数，或者是配置原工程对应的板子上的硬件参数，以上都不用理会，如果不做深入研究的话。现在先看到main()函数下面有osal\_init\_system()该函数用于初始化操作系统，F12打进去函数看到有osalInitTasks()函数，该函数用于创建在系统中task table里的任务。里面有osalInitTasks()用于初始化系统中的任务，F12打进去看到最后一行SampleApp\_Init( taskID )函数，F12打进去。这个函数就是主要的创建网络初始化的函数了，先看工程下有个App的文件夹，打开Coordinator文件，里面就有SampleApp\_Init( taskID )函数。

1.1协调器：Coordinator.c文件void SampleApp\_Init( uint8 task\_id )：

Coord\_UartInit()函数F12 进去，发现就是初始化了了协议栈里的串口。115200的波特率。右键SampleApp\_CoordinatorEB-Pro工程选择options下的C/C++ Compiler的Preprocessor里的Defined symbols。发现里面初始化了ZTOOL\_P1。这下再看Coord\_UartInit()函数，就可以知道是打开了串口一。

MT\_UartRegisterTaskID(task\_id); //注册串口任务 ，没什么好说的。

接下来就是网络参数配置：

//定义广播模式

SampleApp\_Periodic\_DstAddr.addrMode = (afAddrMode\_t)AddrBroadcast;

//指定端口号

SampleApp\_Periodic\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

//指定目的网络地址为广播地址

SampleApp\_Periodic\_DstAddr.addr.shortAddr = 0xFFFF;

SampleApp\_P2P\_DstAddr.addrMode = (afAddrMode\_t)Addr16Bit; //点播

SampleApp\_P2P\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_P2P\_DstAddr.addr.shortAddr = 0x0000; //发给协调器

以上的配置，个人觉得没有任何作用。因为在你的工程里面，协调器只用来接收三个终端无线发送的数据。它自身并没有向终端发送数据。所以配置不配置都可以。

//定义本设备即协调器用来通信的APS层端点描述符

SampleApp\_epDesc.endPoint = SAMPLEAPP\_ENDPOINT; //端点号

SampleApp\_epDesc.task\_id = &SampleApp\_TaskID; //任务ID

SampleApp\_epDesc.simpleDesc=(SimpleDescriptionFormat\_t\*)&SampleApp\_SimpleDesc; //简单描述符

SampleApp\_epDesc.latencyReq = noLatencyReqs; //无延时

以上的配置，用默认配置即可。可以不用考虑到底层APS层去。了解下就行。

afRegister( &SampleApp\_epDesc ); //就是把刚才初始化的简单描述符注册到AF层去。AF层就是管理终端的。

SampleApp\_Group.ID = 0x0001; //又选择了组播模式，创建组ID为0X0001,所有组内的zigbee才能通信。

osal\_memcpy( SampleApp\_Group.name, "Group 1", 7 );

aps\_AddGroup( SAMPLEAPP\_ENDPOINT, &SampleApp\_Group );

以上就完成了协调器的网络开启初始化过程。还有相关的网络配置如果感兴趣可以在Tools文件夹下的f8wConfig.cfg查看。顺便说一句，在一个工程里，可以分别配置成coordinator模式和enddevice模式都是因为选择了不同的文件夹后，也是Tools文件夹下面的f8wCoord.cfg和f8wEndev.cfg文件将对应的网络进行配置好。

Delay\_ms(500);

HalUARTWrite(0, "AT+CIPMUX=1\r\n", 13);

Delay\_ms(200);

HalUARTWrite(0, "AT+CIPSERVER=1,8080\r\n", 21);

Delay\_ms(200);

上面的HalUARTWrite(0, "AT+CIPMUX=1\r\n", 13); HalUARTWrite(0, "AT+CIPSERVER=1,8080\r\n", 21);为用ZIGBEE的串口一与连接的WIFI模块ESP8266进行AT命令进行配置初始化。

1.2终端设备(三个终端我只讲一个终端的配置(因为都是一样的，就讲第二个压力pressure文件吧)，采集会分开来讲)

进入pressure.c文件下的SampleApp\_Init()的过程与1.1协调器的一样，从main()一步一步打进来。

void SampleApp\_Init( uint8 task\_id )：

SampleApp\_TaskID = task\_id;

SampleApp\_NwkState = DEV\_INIT; //设备网络初始化

SampleApp\_TransID = 0;

//初始化了ZIGBEE板上的ADC脚

ADCinital()F12打进去，看注释即可

ADCH&=0X00;//清EOC标志

P0DIR &= ~0x40;

//将 P1.1 定义为外设功能

P0SEL |= 0x40;

ADCCON3=0XB6;//单次转换，参考电压为电源电压，对P0.6采样，精度12位

ADCCON1=0X30;//停止AD

ADCCON1|=0X40;//开始AD

第一个终端采集并上传的压力是用ad来采集到的，以上就是配置的引脚功能。

osal\_start\_timerEx(SampleApp\_TaskID,PERIDOIC\_CMD,PERIODIC\_CMD\_TIME );

osal\_start\_timerEx()函数就是调用了协议栈中的定时器函数，F12打进PERIDOIC\_CMD，

#define PERIDOIC\_CMD 1 //终端定时发送指令事件，无需改动

这就是个任务标志，F12打进去PERIODIC\_CMD\_TIME，

#define PERIODIC\_CMD\_TIME 2000 //Modified by user，终端定期发送周期2S 。也就是定时器每到2S钟发送一个此任务的标志位出去。

SampleApp\_P2P\_DstAddr.addrMode = (afAddrMode\_t)Addr16Bit; //点播地址初始化

SampleApp\_P2P\_DstAddr.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_P2P\_DstAddr.addr.shortAddr = 0x0000; //发给协调器，协调器地址固定为0X0000（此处要记住，协调器的地址为固定的，不能改动）

SampleApp\_epDesc.endPoint = SAMPLEAPP\_ENDPOINT;

SampleApp\_epDesc.task\_id = &SampleApp\_TaskID;

SampleApp\_epDesc.simpleDesc= (SimpleDescriptionFormat\_t \*)&SampleApp\_SimpleDesc;

SampleApp\_epDesc.latencyReq = noLatencyReqs;

afRegister( &SampleApp\_epDesc );

以上为定义本设备用来通信的APS层端点描述符。即配置网络完成。

2. 数据的采集到显示的整个传输流程；

2.1数据采集

2.1.1 pressure.c

继续按照1.2往这里看，uint16 SampleApp\_ProcessEvent( uint8 task\_id, uint16 events )，这个函数可以说是最关键的函数。它一直在判断接收到了什么任务事件(if ( events & SYS\_EVENT\_MSG )该句代码的作用)。中间你这个程序什么都没有用到，所以就不用看了。一直往这个函数最下面看：

if ( events & PERIDOIC\_CMD ) //按照1.2所说，他一开始初始化了一个定时器任务。当2S种到了，就会触发任务进行压力采集并发送事件。此时，检测到了PERIDOIC\_CMD这个任务事件标志位，就做以下的事情：

计算压力值

{

SampleApp\_Send\_PERIDOIC\_CMD()

//启动定时事件，周期为2S，其实之前就启动过了，这里再写就是多余的。 osal\_start\_timerEx(SampleApp\_TaskID,PERIDOIC\_CMD,PERIODIC\_CMD\_TIME );

}

F12打进去SampleApp\_Send\_PERIDOIC\_CMD()函数，F12打进去get\_pressure\_Message()函数：

int get\_pressure\_Message()

{

unsigned short NUM;

while ( !ADCIF ) ;

NUM = ADCL;

NUM |= ((uint16)ADCH) << 8 ;//这里注意一下

NUM >>= 4;

ADCinital();

if (NUM&0x8000) NUM=0;

if (NUM>4000) NUM=0;

ADC=(float)((float)NUM\*3.30/2048);

pressure =(ADC-1.5)\*10/33\*160-15;

}

这里就是用了之前的ADC来采集电压值从而来计算pressure。这里注意，AD的参考电压值为3.3V，没问题。这个2048说明ZIGBEE的ADC为11位的ADC。get\_pressure\_Message()函数的返回值就是得到的pressure压力值。

回到uint8 SampleApp\_Send\_PERIDOIC\_CMD(void)函数，定义的有str\_t数组。

其中的str\_t[0] = 2，str\_t[1] = (uint8)pressure;

if(pressure <= 7)

{

str\_t[2] = 1;

}

else

{

str\_t[2] = 0;

}

AF\_DataRequest(&SampleApp\_P2P\_DstAddr,&SampleApp\_epDesc,ENDDEVIC1\_MSG,3,str\_t,&SampleApp\_TransID,AF\_DISCV\_ROUTE,AF\_DEFAULT\_RADIUS);

AF\_DataRequest()函数为zigbee无线传输发送数据函数。SampleApp\_P2P\_DstAddr为在pressure.c文件里定义的将要接收数据的协调器。

SampleApp\_epDesc为本设备的描述。ENDDEVIC1\_MSG为将要发送的消息簇。3代表已经定义好的str\_t数组的长度。str\_t[0] = 2;就代表协调器接收到的数据帧中第一个字节为2，就代表接收到到的是压力传感器的zigbee发送过来的压力值，将str\_t[1]提取出来就是压力值。同理，在Temperature.c中，str\_t[0] = '1' ;//终端1温度编号。在vibrnce.c，str\_t[0] = 3;//终端3振动编号。这样一看，就知道了数据帧通过第一个字节的数据来决定数据是怎么区分的。

2.1.2Temperature.c

从void SampleApp\_Init( uint8 task\_id )开始，前面的和pressure.c文件的都一样的配置，包括温度的采集也是用的ADC。所以也有ADCinital()函数。同样也是用定时器来触发每隔2S进行一次温度采集并且发送至协调器。后面的uint16 SampleApp\_ProcessEvent( uint8 task\_id, uint16 events )函数与pressure.c文件的过程一样。

2.1.3vibrnce.c

振动也是采用的ADC。过程与pressure.c一致。

3. 能不能看到采样周期、数模转换在那块儿；

if ( events & PERIDOIC\_CMD )

{

SampleApp\_Send\_PERIDOIC\_CMD();

osal\_start\_timerEx(

SampleApp\_TaskID, PERIDOIC\_CMD, PERIODIC\_CMD\_TIME );

}

PERIODIC\_CMD\_TIME为2秒钟。每2s会向协调器发送一次数据。

模数转换都是先ADCinital(void)初始化后，分别在int get\_pressure\_Message()函数中的ADC=(float)((float)NUM\*3.30/2048);

pressure =(ADC-1.5)\*10/33\*160-15;，

int get\_temperature\_Message()函数中的ADC=(float)((float)NUM\*3.30/2048);

temperature = ADC\*10/33\*150-50;

int get\_vibrance\_Message()函数中的ADC=(float)((float)NUM\*3.30/2048);

vibrance = ADC\*10/33\*200;。

4. 协调器怎么创建网络、接收到3个数据怎么区分；

在1.1的协调器的Coordinator.c文件的说明里已经说明了协调器是如何创建网络的过程。

接收到的3个数据，str\_t[0] = 2;就代表协调器接收到的数据帧中第一个字节为2，就代表接收到到的是压力传感器的zigbee发送过来的压力值，将str\_t[1]提取出来就是压力值。同理，在Temperature.c中，str\_t[0] = '1' 。在vibrnce.c，str\_t[0] = 3;//终端3振动编号。这样一看，就知道了数据帧通过第一个字节的数据来决定数据是怎么区分的。

5. 终端和协调器简历链接的过程；

终端：

AF\_DataRequest(&SampleApp\_P2P\_DstAddr,&SampleApp\_epDesc,ENDDEVIC1\_MSG,3,str\_t,&SampleApp\_TransID,AF\_DISCV\_ROUTE,AF\_DEFAULT\_RADIUS);为终端向协调器的发送数据的函数。

协调器中：

uint16 SampleApp\_ProcessEvent( uint8 task\_id, uint16 events )函数中有

case AF\_INCOMING\_MSG\_CMD:

SampleApp\_MessageMSGCB( MSGpkt );

break;

SampleApp\_MessageMSGCB( MSGpkt )这里就是协调器当接收到终端发送来的数据时，会触发AF\_INCOMING\_MSG\_CMD事件，调用了SampleApp\_MessageMSGCB( MSGpkt )函数。

void SampleApp\_MessageMSGCB( afIncomingMSGPacket\_t \*pkt )

{

uint16 flashTime;

uint8 i;

for (i=0;i<3;i++)

afRxData[i] = pkt->cmd.Data[i];

HalUARTWrite(0, "AT+CIPSEND=0,3\r\n", 15);

Delay\_MS(10);

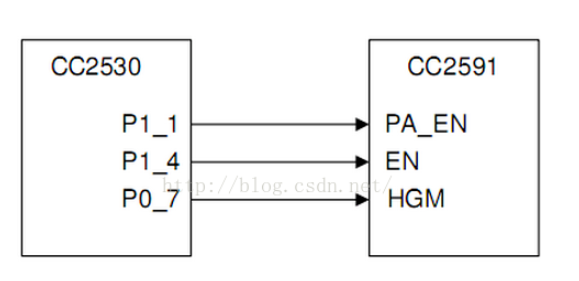
HalUARTWrite(0, afRxData, 3); //输出接收到的数据

}

SampleApp\_MessageMSGCB( MSGpkt )函数就是用来接收终端发送的数据。for (i=0;i<3;i++)

afRxData[i] = pkt->cmd.Data[i];将接收到的数据从pkt复制到afRxData数组中。然后HalUARTWrite(0, "AT+CIPSEND=0,3\r\n", 15)用串口配置wifi为传输数据模式后。HalUARTWrite(0, afRxData, 3); 将afRxData接收到的数据使用wifi模块发送出去。电脑端的wifi模块就可以接收到了zigbee协调器接收到的数据。

CC2590功率放大芯片

CC2590是一款经济高效的高性能RF前端，适用于低功耗和低电压2.4GHz无线应用。CC2590是德州仪器（TI）所有现有和未来2.4GHz低功耗RF收发器，发送器和偏上系统产品的范围扩展器。CC2590通过提供增加输出功率的功率放大器和低噪声系数的LNA来提高链路预算，从而提高接收器灵敏度。CC2590采用4\*4-mm QFN-16封装，提供小尺寸，高输出功率RF设计。CC2590包含PA,LNA,RF匹配和巴伦，可用于高性能无线应用的简单设计。

CC2590有三个脚和CC2530连接，如果硬件管脚连接的是其他的管脚则需要在协议栈上修改。值得注意的是PA\_EN与EN脚只能连接在P1\_0到P1\_5之间，这是由CC2530决定的。HGM可以任意定义。