设备：KW36

例程：Beacon

使用的手机app：nRF connect

1. Beacon是什么？

简单来说就是一种广播数据的规定。基于ble广播，则意味着我们不需要连接到ble设备就可以发现设备的消息。

Beacon又有ibeacon和普通beacon，ibeacon是用于苹果系统的，安卓一般使用普通beacon。

数据格式是怎样的呢。

普通beacon数据格式

2字节公司ID + 1字节Beacon符 + 18字节UUID + 6字节数据 + 1字节发送功率

如果我们需要发送一些变化的数据，比如温度等等，就可以将数据放在6字节数据里。

例程中定义在app\_config.c ，变量adData1[26]

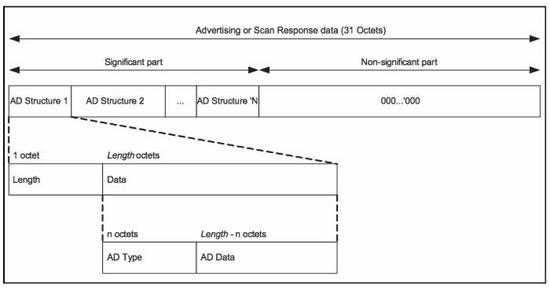
1. ble广播数据

ble广播数据首先就长度而言被限制为31字节。

这31字节里，可以发送不同类型的数据。每一种类型数据格式按照

length + AD type + AD Data.

你可以发N个，用以上方法组织的数据，但是总字节数必须小于等于31



广播数据也定义在app\_config.c，变量advScanStruct

例程中广播数据有两种，一个adData0，用于设置广播的模式—例程中设置为可发现模式且不支持经典蓝牙，adData1则是beacon数据。

static const gapAdStrucature\_t advScanStruct[] = {

{

.length = NumberOfElements(adData0) + 1,

.adType = gAdFlags\_c,

.aData = (uint8\_t \*)adData0

},

{

.length = NumberOfElements(adData1) + 1,

.adType = gAdManufacturerSpecificData\_c,

.aData = (uint8\_t\*)adData1

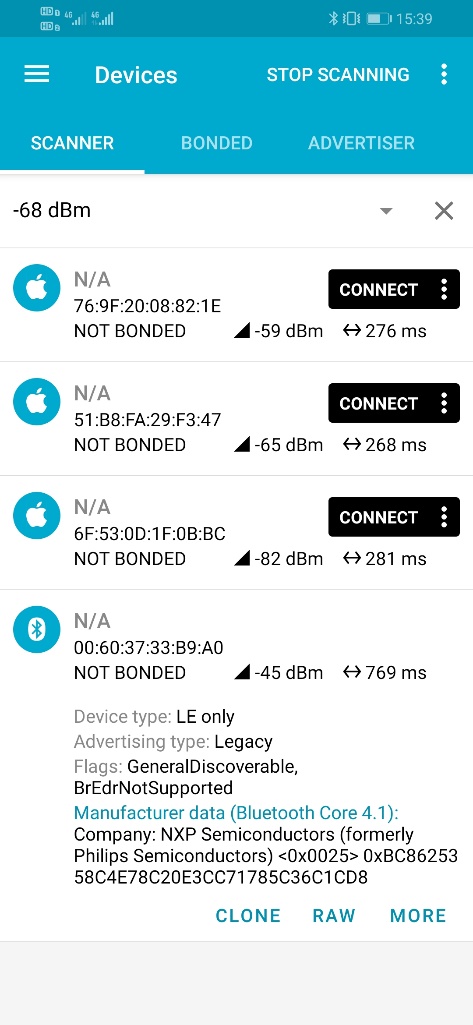
}

}

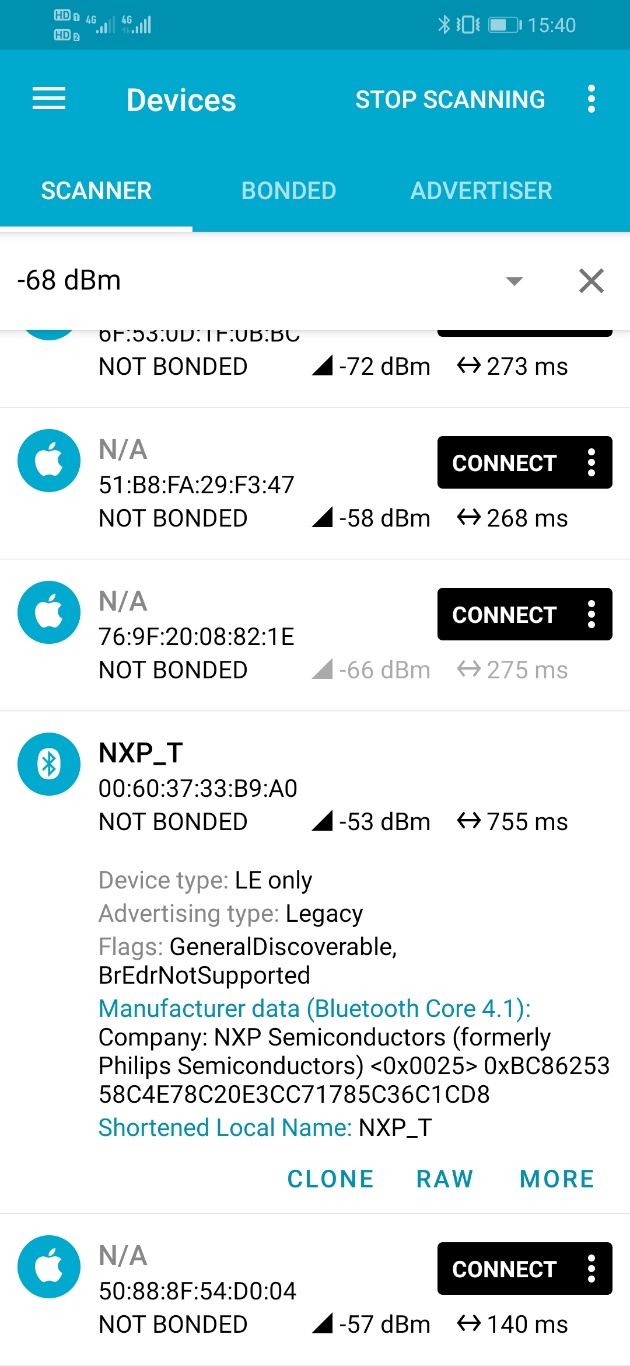
advScanStruct里的数据长度之和应当小于等于31，length长度的计算是，数据长度加上1字节，NumberOfElements(adData0) + 1 + NumberOfElements(adData1) + 1等于 29字节。所以这个广播报文只用了29字节？千万不要忘了还有adType也占用一个字节，但是这个字节是没有被计算在length里的。所以29还要加2字节的adType，这样就是31字节。

这样的广播是没有设备名的。设备名也可以是广播中的一个adType。如果添加这种type，那么app就可以发现设备名。

不添加时候 N/A



添加之后NXP\_T



但是直接在advScanStruct结构体里添加一个名字结构体是行不通的。比如这样添加名字

static const gapAdStrucature\_t advScanStruct[] = {

{

.length = NumberOfElements(adData0) + 1,

.adType = gAdFlags\_c,

.aData = (uint8\_t \*)adData0

},

{

.length = NumberOfElements(adData1) + 1,

.adType = gAdManufacturerSpecificData\_c,

.aData = (uint8\_t\*)adData1

}，

{

.adType = gAdShortenedLocalName\_c,

.length = 6,

.aData = (uint8\_t\*)"NXP\_T"

}

}

刚才说过广播数据要小于等于31，直接在后面加是没有用的，超出了广播数据长度，所以如果非要显示设备名的话，只能削减beacon的数据，如果名字部分占用了7字节，那么beacon数据就要少7字节，但这样来说，beacon数据最后的几个用于实际动态数据被牺牲了。但是你可以自己定义数据结构，而不是非要遵循beacon。剩下的字节也足够自己做其他的文章。

那么beacon数据和local name在数组advScanStruct，相互换位置吗？不能，因为BleApp\_Init有FLib\_MemCpy(&gAppAdvertisingData.aAdStructures[1].aData[3], ctx.hash, 16); 他把uuid的hash值拷贝的地址是scan结构体的第二个数据，所以数组第二个数据必须是beacon数据。但是你也可以把这段代码注释掉，这样就可以随意换顺序，或者当你换顺序了把，aAdStructures[] 里的标号，根据beacon数据在advScanStruct实际位置，改一下也行。

1. 代码运行过程

使用低功耗情况下#define cPWR\_UsePowerDownMode 1，在app\_preinclude.h

就KW36而言，简言之，程序启动以后，代码会直接进入低功耗模式，用SW3唤醒。唤醒以后，定时唤醒设备，发送广播数据。由创建的AppIdle\_Task任务调用PWR\_enterLowPower进入低功耗模式5。然后唤醒以后切到模式8，不断由AppIdle\_Task来休眠设备。

睡眠时间PWR\_ProgramDeepSleepTime

不使用低功耗使用低功耗情况下#define cPWR\_UsePowerDownMode 0

则不断在App\_Thread循环，处理发送来的消息，发来的消息实现等等对我们都是不可见的。开机后白光闪烁，按下sw3后红灯闪烁

3.1

Kw36有两个低功耗模式，一个是模式5，一个是模式8，都对应VLLS2。但唤醒源不同，用途也不同。模式5用于开机后，使用按键唤醒，按键处理会切换模式8。模式8用于模式5被唤醒以后，模式8唤醒源是定时器，定时器定时唤醒mcu，让mcu广播，广播完又睡眠。

3.2

VLLS2是复位唤醒，也就是说一唤醒代码从头开始跑。0x2000\_0000 to 0x2000\_3FFF这段sram被remain。

但是kw36使用了其它方法，使得在唤醒以后，可以从睡眠处继续运行下去。

首先使用了setjmp，在进入睡眠前，保存了运行上下文，使用BOARD\_VLLS\_EnterCb保存中断等等消息。

其次复位入口正常来说只有一个，但是kw36有两个，一个NORMAL\_BOOT一个WARM\_BOOT，一般复位从NORMAL\_BOOT进，但是在低功耗唤醒以后是从WARM\_BOOT进，而warm\_boot的函数一方面恢复了中断等寄存器状态，一方面使用jmpcontext，恢复运行上下文，从睡眠之后的代码继续运行。