前言

> 适用范围：BlueNRG-LP 软件， BlueNRG-1/2的也可以参考此方法
> 下文主要描述使用BlueNRG-LP SDK进行开发时,如何搭建使用静态协议栈环境。

背景

> 在使用BlueNRG-LP调试的时候，有时遇到Flash空间不够的问题，如果应用还需要包含OTA，如果OTA中又包含协议栈，为了解决Flash空间不够这个问题，我推荐使用静态协议栈的方式，节省空间，加速OTA的升级过程。

知识背景须知

> 下文描述具体的操作，我默认你已经知晓ST 官方的BlueNRG-xx 基本开发环境搭建和OTA基本知识，ST官方提供比较多类型的OTA方式，详细请自行看官方文档《AN4869 Application note The BlueNRG-1, BlueNRG-2 BLE OTA (over-the-air) firmware upgrade.pdf》
> 或者《AN5463 Application note The BlueNRG-LP (over-the-air) firmware upgrade.pdf》。

> 温馨提示:请使用最新版本**SDK**进行开发

搭建静态协议栈环境

> ST官方SDK中已经提供了静态协议栈的Demo:

- C:\Users\Lucien\ST\BlueNRG-LP DK 1.0.0\Projects\BLE_Examples\BLE_StaticStack
  - Release
  - OTA_BTL_ResetManager
- C:\Users\Lucien\ST\BlueNRG-LP DK 1.0.0\Projects\BLE_Examples\BLE_SensorDemo_StaticStack
  - Release
  - LowerApp_OTA
  - HigherApp_OTA

> 其中，一个工程负责生成协议栈，另一个工程负责应用，那么这里BLE_StaticStack中的Release与OTA_BTL_ResetManager怎么和Release，LowerApp_OTA和HigherApp_OTA组合呢？

- BLE_StaticStack中的Release + BLE_SensorDemo_StaticStack中的Release //不带OTA的使用固定协议栈的方法
- BLE_StaticStack中的OTA_BTL_ResetManager + BLE_SensorDemo_StaticStack中的LowerApp_OTA or HigherApp_OTA //带OTA的使用固定协议栈的方法

> 如果你的应用工程移植起来不繁琐或者还没有开发应用并且ST官方SDK中已经提供了静态协议栈的Demo满足你的基本要求，你可以直接使用它。如果不是，那可能就需要直接在你目标的工程上改为静态协议栈，并且可能需要裁剪协议栈。下面我介绍一个比较复杂些的使用静态协议栈的修改方法。我们先通过了解Flash分布来知晓接下来这个实验的整个框架。

Flash分布:

Flash区域	说明
NVM	4KB
APP	客户应用程序，暂定92KB
Boot + BLE_OTA_ServiceManager	应用OTA,12kb
BLE Stack	协议栈，暂定138KB，可以裁剪
2.4G OTA (10KB 升级协议栈)	占10KB 升级协议栈和应用代码

> BlueNRG-355 的Flash空间是256kB,实现的基本思路是:由于协议栈的升级几率比较低，这里又不太想占用太多的代码空间，所以协议栈的升级改用走2.4G私有协议进行升级，如果是非安全性要求很高很高的应用（一般的应用）我个人认为不需要升级协议栈。这里对协议栈的空中升级只是留一手，万一需要升级，又不想通过串口或者SW口升级。

- 2.4G OTA 使用的工程是RADIO_OTA_ResetManager中的Client，
- 如果对协议栈和应用一起升级的工程是RADIO_OTA_ResetManager中的Server_FixedImage。
- BLE full Stack 的工程是 BLE_StaticStack
- Boot + BLE_OTA_ServiceManager 的工程是BLE_OTA_ServiceManager ，主要实现不备份方式的OTA（即OTA服务程序和应用不捆绑在一起）
- APP 客户应用程序，这块可使用的大小可以根据实际使用BLE Stack size进行调整，后面会描述调整的方法

具体操作和分析

> 下面我们一步一步，一个工程一个工程调试，从“基本的协议栈+app”到“支持应用OTA的协议栈+app”再到“支持协议栈OTA和支持应用OTA的静态协议栈Demo".我主要给出基于Keil 优化分析过程，IAR的操作类似，相对更简单一些，可以同样类比过去。

更改**BLE_MultipleConnections**为静态协议栈版本

1. 编译生成静态协议栈: BlueNRG-LP DK 1.0.0\Projects\BLE_Examples\BLE_StaticStack
   编译成功后，会在目录BlueNRG-LP DK 1.0.0\Middlewares\ST\Bluetooth_LE\library\static_stack下生成libbluenrg_lp_static_stack.a文件，这里Keil 和IAR都一样，但想要成功编译并且生成static_stack下生成libbluenrg_lp_static_stack.a 需要使用安装arm-none-eabi工具链，none表示不带操作系统（普通的RTOS不算在这类操作系统中，这里主要是稍微大型一些的如Linux操作系统）。工具链请自行到GNU官网上下载，安装好后，然后设置环境变量.

   安装工具链和测试可以参考文档:BlueNRG-LP DK 1.0.0\Projects\BLE_Examples\BLE_StaticStack\README.txt

   这里主要是keil或者IAR编译后，运行脚本:
   ".....\Utility\create_sym_lib.exe --symbols ....\sym_export.txt --rename BLE_STACK_Init=BLE_STACK_Init_nocallbacks Release\Objects\BLE_StaticStack.axf .....\Middlewares\ST\Bluetooth_LE\Library\static_stack\libbluenrg_lp_static_stack.a"
   主要作用是把文件sym_export.txt中的列表中的函数，从编译生成的.*axf文件中找出对应的地址信息，然后通过create_sym_lib.exe生成.a库，这个*.a文件就是应用于协议栈的桥梁，应用层是在协议栈之上的，这个是接口，就算将来升级协议栈，接口一般都是不允许改动的，这样不会影响应用。
2. 修改BLE_MultipleConnections调用静态协议栈

默认BLE_MultipleConnections工程，使用的不是固定协议栈的方式。我们需要更换为调用上一步骤生成的*.a文件
此过程可以参考官方文件:
BlueNRG-LP DK 1.0.0\Projects\BLE_Examples\BLE_SensorDemo_StaticStack\README.txt

操作步骤如下:

- 移除或者禁止加入编译老的libbluenrg_lp_stack.a和stack_user_cfg.c
  - Keil平台禁止加入编译可以右击该文件，然后选择Option for File xxx ... 接着把Include target build 去掉，最后点击OK，可以看到文件前面有个红色的禁止图标
  - IAR平台禁止加入编译可以右击该文件，然后选择Options，接着勾选左上角的Exclude form build，最后点击OK，可以看到文件图标颜色变灰色。
  - 禁止整个文件夹加入编译也是如是操作，后续不再啰嗦。
- 移除或者禁止整个NVM文件夹，Middlewares/Profiles文件夹和bluenrg_lp_hal_vtimer.c bluenrg_lp_ll_radio.c， 这两部分已经在协议栈中包含了，如果不禁止掉会重复包含重新定义。
- 添加文件libbluenrg_lp_static_stack.a和bluenrg_lp_stack_init_if.c到工程，文件路径:Middlewares\ST\Bluetooth_LE\Library\static_stack
  - Keil 平台添加.*a库后，需要将其文件类型手动选择为库文件类型。//右击*.a文件，Options for file *.a ,然后选择File type 下拉框选择Library file。
- 定义宏MEMORY_FLASH_APP_OFFSET，这个宏定义的值能决定生成文件的绝对地址，由于Flash的擦除必须是每次一整页(2KB),所以加入协议栈的bin文件占用地址0x1A438 Byte（105.05KB） 则应用偏移为0x1A800(106KB)
  - Keil 平台MEMORY_FLASH_APP_OFFSET这个宏定义需要定义在Options for target -> linker->Misc controls中，格式可以参考BLE_SensorDemo_StaticStack工程。
  - IAR 平台MEMORY_FLASH_APP_OFFSET这个宏定义需要定义在Options->linker->config->configuration file symbol definitions中，格式也可以参考BLE_SensorDemo_StaticStack工程。
- 定义MEMORY_RAM_APP_OFFSET宏，因为静态协议栈这种方式，变量在两个不同的工程中，所以协议栈的变量和应用的变量得区分开来，这个宏需要根据协议栈变量占用RAM的大小来决定，查看默认的工程，可以看到BLE_StaticStack.map文件中变量存放的最后一行是:

Base Addr	Size	Type	Attr	Idx	E Section Name	Object
0x200007ec	0x00000024	Zero	RW	4255	.bss	libbluenrg_lp_stack.a(mem_alloc.o)

- RAM的起始地址为0x20000000, 可以算出协议栈占用的内存总数为0x7EC+ 0x24 = 0x810, IAR平台下这个数值很可能是不一样（0x7AC），IAR的编译效率一般会更高一些，这里如果担心算得不准，也可以设置稍微偏大一点点。
- 还有一点需要注意的是协议栈中定义的宏BLE_STACK_FULL_CONF 需要和应用中保持一致，如果不一致可能有有些问题。
- 协议栈中代码裁剪可以精确到函数，除了选择合适的协议栈配置宏定义外，可以通过是否注释bluenrg_lp_cmd_if.c中cmd_call_table[]数组中的函数，用来决定是否将该函数编译进协议栈。（这种方式相当灵活，暂时没有看到过可以做到这样的其他蓝牙芯片厂家）
- 编译成功后，分别通过下载BLE_StaticStack.hex 和 BLE_MultipleConnections_MasterSlave.hex， 可以使用Keil 或者IAR直接download，也可以使用Flash utility下载。实际测试发现，如果在线调试，可能会有些错误，应该是平台工具的问题，RAM变量的衔接问题，建议使用静态协议栈后就不要用在线debug单步调试了，如果想用单步调试则先不要改成静态协议栈这种方式。

        - 如果下载bin文件，可以合称后直接下载，也可以分别下载，但需要设定下载地址，操作和合成细节在这就先不说。
    - 小结: 这个阶段我们将一个普通使用非静态协议栈工程改成了使用静态协议栈的方式，将协议栈占用的Flash和RAM 区分开来，然后通过不同的协议栈宏配置和注释cmd_call_table[]数组中的函数，可以实现对协议栈的裁剪

固件	MEMORY_FLASH_APP_SIZE [Linker]	MEMORY_FLASH_APP_OFFSET[Linker]	MEMORY_RAM_APP_OFFSET[Linker]	RESET_MANAGER_SIZE[PreDefine]			
BLE_StaticStack	0x22800	none(default 0)	none(default 0)	0x22800			
BLE_MultipleConnections	none	0x22800	0x810	none			

- 测试: 成功更改后，应该可以通过串口日志和使用BLE测试工具成功看到应用程序跑起来

```
    Flash分布:
```

Flash	说明
NVM	4KB
APP	114KB

| BLE Stack | 138KB
| | |

更改增加BLE_OTA_ServiceManager为静态协议栈版本

- 将BLE_OTA_ServiceManager更改为支持静态协议栈版本比较简单，和上述步骤很类似，我就不一一描述了，下面我只描述一些细节，供参考。
- 更改OTA处代码OTA_btl.h中将

```
#define SERVICE_MANAGER_OFFSET     (0x0)
#ifdef CONFIG_DEVICE_BLUENRG_LP
    #define SERVICE_MANAGER_SIZE       PAGE_SIZE_ROUND(90* 1024) /* BlueNRG-LP,
BLE stack v3.x with modular approach TBR*/
#endif

#define SM_APP_SIZE                PAGE_SIZE_TRUNC((_MEMORY_FLASH_SIZE_-
SERVICE_MANAGER_SIZE-NVM_SIZE))
```

```
更改为:
```

```
#ifndef SERVICE_MANAGER_OFFSET
    #define SERVICE_MANAGER_OFFSET     (0)
#endif
#ifndef SERVICE_MANAGER_SIZE
    #define SERVICE_MANAGER_SIZE       PAGE_SIZE_ROUND(90* 1024) /* BlueNRG-LP,
BLE stack v3.x with modular approach TBR*/
#endif

#define SM_APP_SIZE                PAGE_SIZE_TRUNC((_MEMORY_FLASH_SIZE_-
SERVICE_MANAGER_SIZE-NVM_SIZE-SERVICE_MANAGER_OFFSET))
```

- 注意修改BLE_OTA_ServiceManager中协议栈配置宏和静态协议栈保持一致（我这里是BLE_STACK_FULL_CONF）
- BLE_MultipleConnections做如下修改:
    - 在linker中增加宏定义(注意Keil格式和IAR格式稍微有点不一样)：

```
        CONFIG_OTA_USE_SERVICE_MANAGER=1  // 修改链接文件使按照
CONFIG_OTA_USE_SERVICE_MANAGER格式划分
        MEMORY_FLASH_APP_OFFSET=0x25800 // 应用其实偏移地址修改为:
BLE_StaticStack+BLE_OTA_ServiceManager
        MEMORY_RAM_APP_OFFSET=0x0810 // 内存不能占用到协议栈使用那部分的
```

```
    + 增加 修改链接脚本文件: BlueNRG_LP.sct or BlueNRG_LP.icf
```

```
Keil:  // #ifdef CONFIG_OTA_USE_SERVICE_MANAGER分支下

#define SERVICE_MANAGER_SIZE     (0x16800)

#define MEMORY_FLASH_APP_SIZE  (_MEMORY_FLASH_SIZE_ - SERVICE_MANAGER_SIZE -
FLASH_NVM_DATASIZE)
#define MEMORY_FLASH_APP_OFFSET  (SERVICE_MANAGER_SIZE)

更改为:
#ifndef SERVICE_MANAGER_SIZE
#define SERVICE_MANAGER_SIZE     (0x16800)
#endif

#ifndef MEMORY_FLASH_APP_OFFSET
    #define MEMORY_FLASH_APP_OFFSET  (SERVICE_MANAGER_SIZE)
#endif

#define MEMORY_FLASH_APP_SIZE  (_MEMORY_FLASH_SIZE_ - MEMORY_FLASH_APP_OFFSET -
FLASH_NVM_DATASIZE)

IAR:  // CONFIG_OTA_USE_SERVICE_MANAGER分支下

同样类比更改，这里不再复述
```

固件	MEMORY FLASH APP SIZE [Linker]	MEMORY FLASH APP OFFSET [Linker]	MEMORY RAM APP OFFSET [Linker]	RESET MANAGER SIZE [PreDefine]	SERVICE MANAGER SIZE [PreDefine]	SERVICE MANAGER OFFSET [PreDefine]	
BLE_StaticStack	0x22800	none(default 0)	none(default 0)	0	none	none	
BLE_OTA_ServiceManager	0x3000	0x22800	0x810	none	0x25800	0x22800	
BLE_MultipleConnections	none	0x25800	0x810	none	none	none	

- Flash分布如下图

Flash	说明
NVM	4KB
APP	102KB
BLE_OTA_ServiceManager	12KB
BLE Stack	138KB

增加支持协议栈的升级

- 这里协议栈的升级由于空间不够，这里就采用2.4G 升级方式。工程路径:
  BlueNRG-LP DK
  1.0.0\Projects\Peripheral_Examples\Examples_MIX\RADIO\RADIO_OTA_ResetManager-->Client
- 主要修改配置宏，前面描述比较详细，这里我就不一一描述了。主要罗列一些不同工程的linker配置和宏配置。
  - RADIO_OTA_ResetManager:

```
linker:
MEMORY_FLASH_APP_SIZE=0x2800
MEMORY_RAM_APP_OFFSET=0x0810

Preprocessor Defined symbols:
SERVICE_MANAGER_SIZE=0x2800
```

  - BLE_StaticStack:

```
linker:
MEMORY_FLASH_APP_SIZE=0x22800
MEMORY_FLASH_APP_OFFSET=0x2800

Preprocessor Defined symbols:
RESET_MANAGER_SIZE=0x25000
BLE_STACK_FULL_CONF
```

  - BLE_OTA_ServiceManager:

```
linker:
MEMORY_FLASH_APP_SIZE=0x3000
MEMORY_FLASH_APP_OFFSET=0x25000
MEMORY_RAM_APP_OFFSET=0x0810

Preprocessor Defined symbols:
BLE_STACK_FULL_CONF
SERVICE_MANAGER_OFFSET=0x25000
SERVICE_MANAGER_SIZE=0x3000
```

- BLE_OTA_ServiceManager:

```
linker:
CONFIG_OTA_USE_SERVICE_MANAGER=1
MEMORY_FLASH_APP_OFFSET=0x28000
MEMORY_RAM_APP_OFFSET=0x0810

Preprocessor Defined symbols:
BLE_STACK_FULL_CONF
```

Flash	说明
NVM	4KB
APP	92KB
BLE_OTA_ServiceManager	12KB
BLE Stack	138KB
2.4G_OTA_ServiceManager	10KB

总结

通过上述一步一步的优化，我们完成了将普通工程改成使用静态协议栈的方式，并且支持OTA。
实际应用可能有各种不同的应用需求，用户可以根据同样原理去优化适配合适你们的工程。