分散加载+增量固件升级方案

升级流程



说明：

* 当网关接收到节点发送的节点状态报告请求升级后，判断是否有升级需要。如果没有升级任务，网关判断节点状态正常，则发送跳转命令（跳转至应用）；如果没有升级任务安排，网关判断节点状态异常，或者网关需要等待一段时间进行升级，则发送节点休眠命令，T1时间，单位ms；如果有升级任务则发送包含待升级模块信息的升级命令。
* 网关发送应答信息后，进入xmodem模式，在T2时间内等待节点发送xmodem起始命令开始xmodem数据传输。如果没有接收到节点发送的起始命令，则跳转回正常运行状态等待下一条节点的升级请求。
* 节点发送升级请求后没用收到网关的应答，则休眠T3时间，再次请求升级，最大升级时间后未进行升级则取消升级任务。
* Xmodem传输失败则校验固件程序，程序完整则自动跳转到应用程序，否则进入错误状态。

Flash内存映射：



自定义协议：

帧格式：

|  |  |
| --- | --- |
| 帧头，1字节 | |
| 目的地址，8字节 | |
| 帧类型，1字节 | 帧长度，1字节 |
| 数据，帧长度\*1字节 | |
| （CRC16校验，2字节(非xmodem帧)） | |

说明：目的地址为8字节devEUI，CRC16校验在xmodem类型数据帧中不出现。

帧大小为11+len(data) + ()

帧类型：

节点->网关

* 节点状态报告：01H
* Xmodem数据：03H

网关->节点

* 网关应答： 02H
* Xmodem数据：03H

节点状态报告：5字节

数据内容

|  |  |
| --- | --- |
| 状态代码，1字节 | 固件版本号，4字节 |

状态代码：

01H：节点正常，请求升级

02H：节点异常，升级过程中传输出错

03H：节点异常，升级过程中节点运行出错

网关应答：1~13字节

数据内容

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 时间/起始地址 | 模块大小 | CRC32校验值 |
| 1字节 | 4字节 | 4字节 | 4字节 |

类型：

01H：某模块需要进行升级，起始地址，模块大小，CRC32校验值

02H：跳转入应用程序，类型后不跟其他数据

03H：节点休眠，发送休眠时间

XMODEM数据：133字节

数据内容

|  |
| --- |
| XMODEM数据包，1字节（命令），133字节（数据包） |

Bootloader：

存放在flash起始处的一块空间中，用于接收上位机发送的升级包对对应模块进行升级。

V0.1，升级包内容直接为程序内容，首先写入ram中的一个buffer内，合适时机将buffer内的数据考入对应的flash区域中。

V0.2，升级包内容为mdcd算法生成的固件包生成指令，首先根据指令在buffer内构建固件，然后将构建完成的固件写入对应的flash区域中。如果模块大小大于buffer的大小，将模块按buffer大小进行分割，分别计算固件生成命令。

V0.3 升级包内容为新算法生成的固件包升级指令，可以直接根据指令在flash中进行固件更新，并且不需要外的flash和内存空间，不受此限制。

Entry

Entry应用程序的入口模块，程序的中断向量表，初始化过程在该模块内，该模块位于应用程序所处flash的起始位置。

Entry模块中可以调用其他模块所暴露的函数，但是该模块不向外部暴露函数，没有函数跳转表。

Module\_desc

Module\_desc模块紧接着entry模块，用于存放模块信息表。模块信息表位于module\_desc模块的顶端位置，表中每一个元素为一个结构体，包含一个对应模块的入口地址，模块版本号，模块大小，模块的crc32校验值。

每进行程序更新时，该模块都会被更新（因为至少被更新模块的版本号发生了变化）

Module

普通模块的代码。模块的顶端存放着模块对外接口的函数表，函数表的地址即模块的起始地址（入口地址）。

说明：

* 需要使用分散加载文件，将每个模块的加载域和执行域分割开。模块之间不能直接引用全局变量，需要设置接口函数完成。
* 每个模块的头文件中声明了接口函数表的类型结构体，当一个模块需要引用另一个模块的接口函数时，首先包含该函数的头文件和应用描述，声明一个接口函数表的变量，然后从模块描述表中取出要引用的模块的入口地址，根据入口地址调用需要的函数。

（如何做到增加/删除模块时，不至于所有模块都要进行升级）

（这是一个问题，如果某个很基础的模块进行了升级，那么可能几乎所有模块都必须进行升级，同时如果该模块的改动很小，其他模块对应的改动也很小时，这样的变动使得升级数据量很大，这时就体现出增量升级的优势了）

目前问题：  
1、当程序中模块数量有增减时，是否会让整个固件需要进行更新，如何解决该问题？

当程序中模块数量有变化时，每个模块中使用的module\_ID可能发生变化，从而每个模块都可能发生微小的变化，如何解决该问题：

暂时方案：

Module\_desc中的模块列表放在DESC模块的末尾段

对module\_desc有以下几种操作：添加，删除，修改，需要遵循以下方式：

添加：新添加的模块信息，应该放在module\_desc末端进行追加

删除：将原模块信息改为{0,0,0}

修改：修改appConfig.h即可

目前来看，这种方式可以使module\_desc里面其他模块的位置保持不变

2、当某个某块的接口函数发生变化时，也需要保持其他函数的位置不变，方法类似对模块进行修改：

添加：在结构体末尾部分追加函数声明

删除：声明不变，将定义中对应的函数赋为NULL

修改：修改函数即可

问题：

1、考虑什么情况下需要全模块更新？

2、堆栈位置如何设置？

目前解决办法：将堆栈放在尺寸较大的且不经常进行更新的库函数处

3、需要使用C标准库库函数和hal库函数的模块如何放置？

想法：需要用到库函数的模块地方：

* 外设配置，如时钟，PWR，RTC，UART等
* 中断
* C标准库strcpy等

将.ANY，用到标准库，HAL库的模块放在一个module之中，外部只用接口调用内部的函数

4、节点代码可否同时开启多个app，如果可以同时开启多个app，sct文件如何使用？

暂时先设置一个大的区域存放app，app使用者自己在该区域内分配flash和ram空间

5、问题：desc模块在flash中，因此只能调用flash读写函数进行修改，而flash读写需要至少擦出一个page 的数据，因此每次有更新需求时，desc整个模块必然要进行全部更新。

这里出现一个问题，在类似这种情况出现的时候，增量固件是否就派上了用场。

解决：可以先将desc模块拷入ram当中，然后对ram中的desc模块中需要修改的部分进行修改，再一次性写回flash中。