STM32F769DK 云端固件升级例程软件开发文档

1. 前言

STM32F769DK 云端固件升级例程分为两部分：Bootloader 和用户应用程序。

该软件基于 STM32F7Cube 库运行在 STM32F769I-DISCO 探索板上，利用百度的 IoT 平台实现了从云端更新固件的功能。

Bootloader 部分主要负责将新版本用户应用程序从 QSPI Flash 烧到 MCU 内部 Flash 中，并跳转到应用程序运行，功能相对简单。

用户应用程序部分除了用户自己的功能程序外，还包括固件版本云端推送，固件文件云端下载以及断点续传等功能，程序结构相对复杂。

本文档主要介绍用户应用程序这部分。也会对 Bootloader 做简单介绍。

用户应用程序软件包包括以下功能组件

- 从云端更新固件的应用程序代码
- MbedTLS（用于建立和云端的安全连接）
- LwIP（使用有线连接时的 TCP/IP 协议栈）
- FreeRTOS
- Paho Embedded MQTT（MQTT 客户端在 STM32 上的实现）
- cJSON（封装和解析 MQTT 数据包负载）
- STM32F769I-Disc 的板级驱动（包括 QSPI Flash，LCD，Wifi 模块等驱动）
- STM32F7 系列的 HAL 库

支持 IAR Embedded Workbench IDE，V7.80.4 及以上版本。

本例程仅作为 STM32 无线远程升级功能的参考。

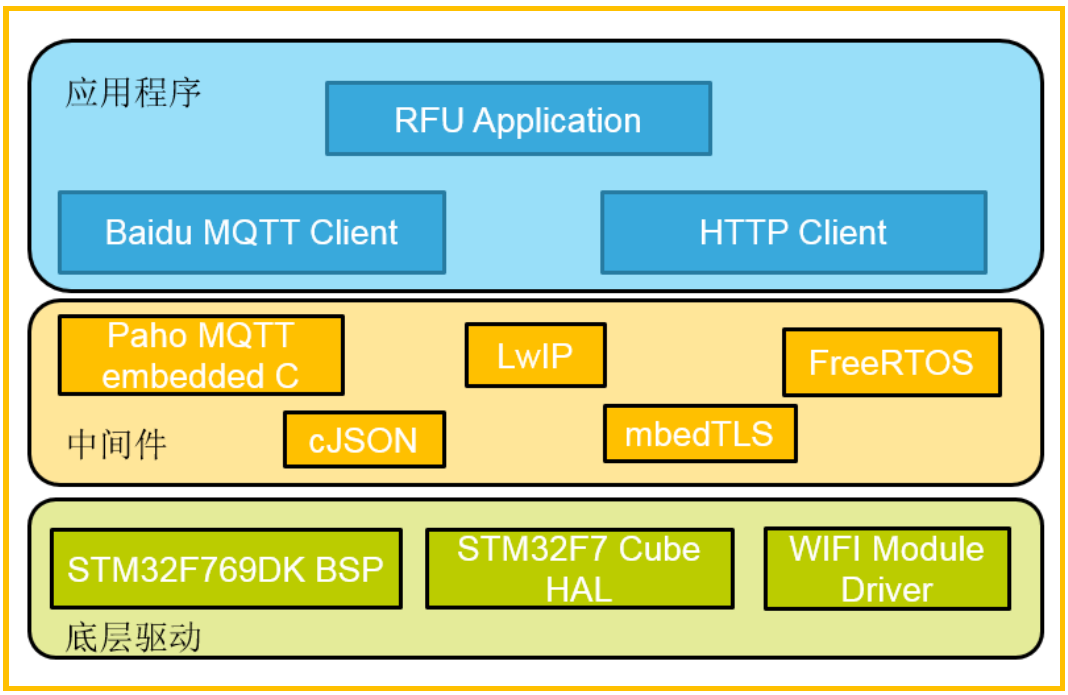
2. 软件架构

用户应用程序的软件分层如下图，应用层程序可以通过接口函数调用各层的服务。

- STM32F7Cube HAL 库：HAL 驱动层向上层软件提供了使用各个外设的接口函数。包括通常的或扩展属性的 APIs。上层的中间件，应用程序等可以通过调用这些 API 函数来操作外设，这样便不会使得上层的软件依赖于某个特定的 MCU。使得程序更具有复用性，并且容易移植到其他的 MCU 系列上去。
- STM32F769DK BSP 板级驱动：针对 STM32F769DK 这块板子上使用到的外设和资源，提供了相应的 API 函数，比如 LED 灯的控制，按键的操作等。
- LwIP 协议栈：TCP/IP 协议的实现。
- mbedTLS: 支持设备与云端建立 TLS 连接。
- Paho：Paho Embedded MQTT 实现 MQTT 客户端部分的协议。支持与云端建立 MQTT 通信。
- cJSON：针对嵌入式系统，提供 JSON 格式数据的解析。
- FreeRTOS: 轻量级的实时操作系统。

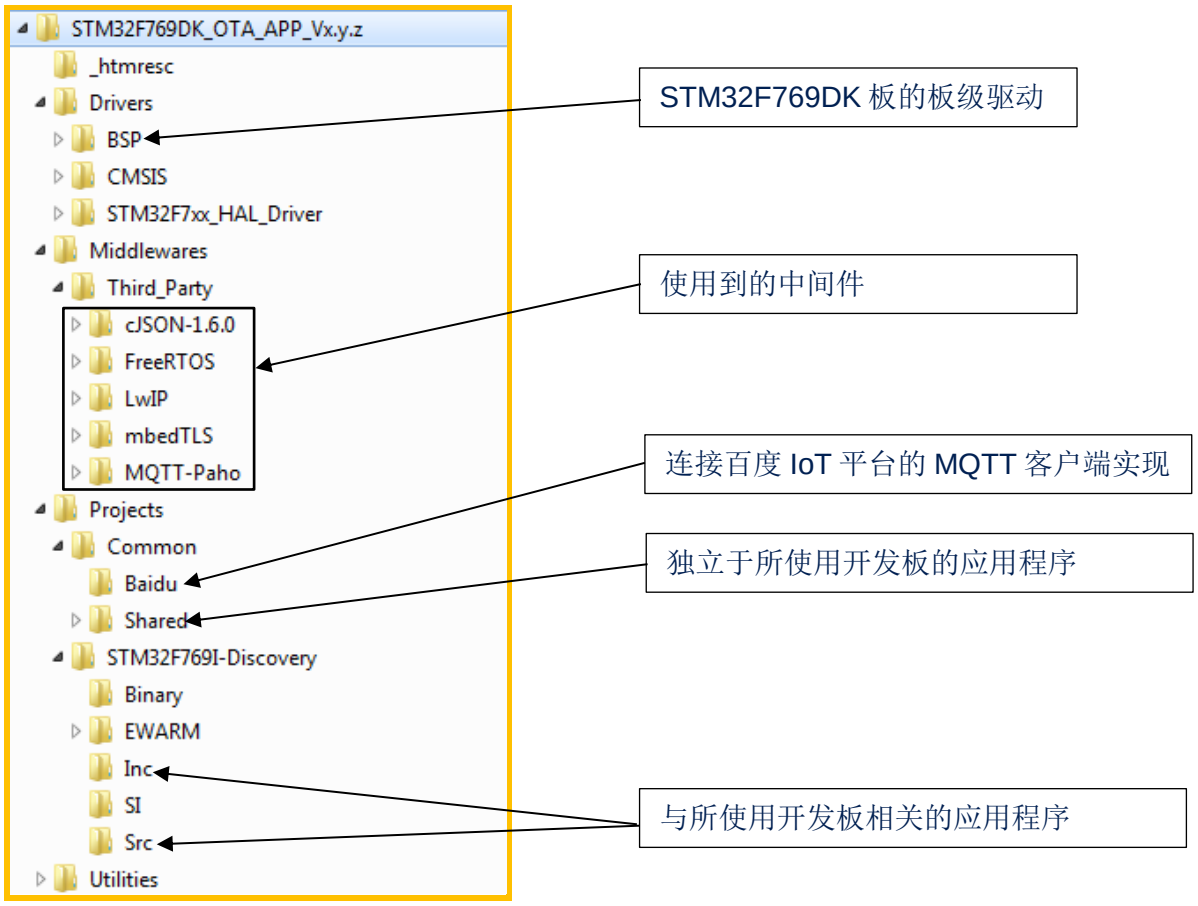
除此以外，应用层还实现了:

- 连接百度 IoT 平台的 MQTT 客户端: 基于 Paho，根据百度 IoT 平台的连接逻辑进行封装实现。
- HTTP 客户端：在 MCU 端实现了 HTTP 客户端，支持通过 TCP 或者 TLS 的方式从云端下载文件。
- 远程固件更新的程序：实现云端推送，自动下载，以及断点续传的逻辑。



3. 文件结构

用户应用程序的文件结构如下：



4. 软件模块

4. 1 网络抽象层

在实际的应用中，设备端可能通过多种方式连接到云端。比如以太网，WIFI，2G/3G 等。所以程序中增加了网络抽象层，来支持在不同连接方式上的切换，见下图。

应用层的代码访问网络的接口函数都在 net.c 文件里,主要函数有：

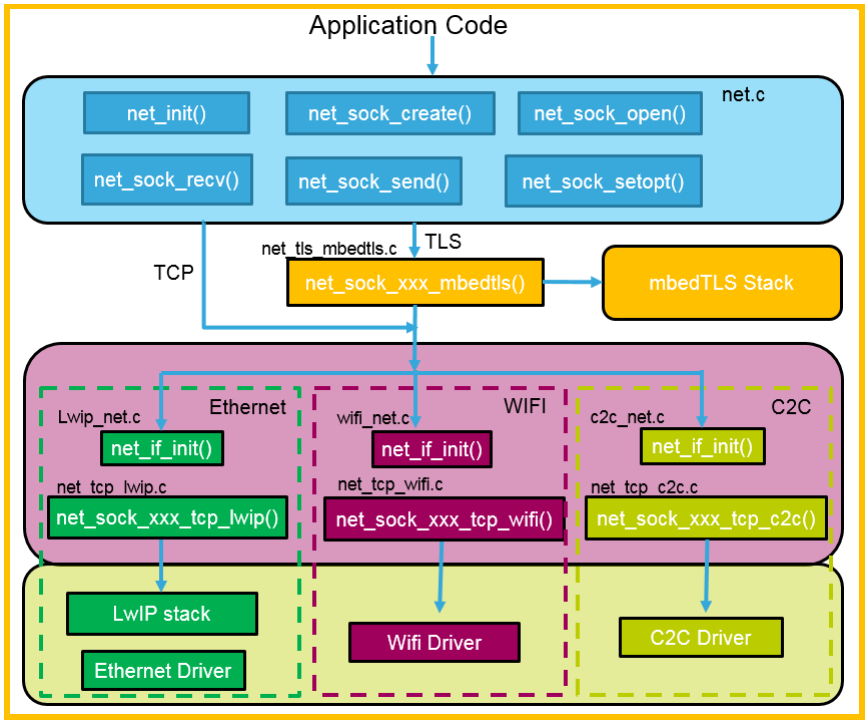
- **net_int()**函数： 该函数里会调用 net_if_init 函数对网络接口进行初始化。根据不同的网络接口，net_if_init 函数的实现各不一样。对于以太网来说，就是完成 LwIP 协议栈的初始化，GPIO，MAC，DMA 等的初始化等。net_if_init 函数在 lwip_net.c 中实现。

  对于 WIFI，就是 WIFI 模块的初始化和连接到 AP，在 wifi_net.c 中实现。

  对于 2G/3G，就是 2G/3G 模块的初始化以及连接到相应的网络，在 c2c_net.c 中实现。
- **net_sock_create()**函数：根据 net_int 中初始化的不同网络接口，调用对应的 net_sock_create_tcp_xxx()函数。初始化 Sock 结构体，注册对应的输入输出函数。如果在调用 net_sock_create 时，输入的协议参数是 USE_MBED_TLS，则会调用 net_sock_create_mbedtls 函数来初始化 sock 结构以及初始化 tlsData 结构。
- **net_sock_open()**函数：调用在 net_sock_create()中注册的相关函数，新建一个 socket。
- **net_sock_recv()**函数：调用在 net_sock_create()中注册的相关函数来接收数据。
- **net_sock_send()**函数：调用在 net_sock_create()中注册的相关函数来发送数据。
- **net_sock_setopt()**函数：设置 socket 属性，比如 blocking,noblocking,read_timeout, write_timeout 等。

net_sock_xxxx_tcp_xxx()函数的实现分别在 net_tcp_lwip.c, net_tcp_wifi.c, net_tcp_c2c.c 中。向下根据不同的网络，调用对应的驱动。比如以太网，就会用到 Lwip 协议栈和 STM32 MCU 的以太网驱动。而 WIFI 和 2G/3G 的方案，就会用到对应模块的驱动。如果 WIFI 和 2G/3G 模块中没有实现 TCP/IP 协议栈，那么也会需要在 MCU 端实现 TCP/IP 协议栈，就也会用到 LwIP 协议栈。

net_sock_xxxx_mbedtls()函数的实现在 net_tls_mbedtls.c 中。



4.2 Baidu MQTT 客户端

MQTT 协议在 TCP 之上，属于应用层的协议，它基于发布/订阅模式的轻量级通信协议，适用资源受限设备，低带宽，高延时，不稳定网络中进行消息传输。

Baidu 的 MQTT 客户端，基于 Paho 的 MQTT 客户端实现。这部分的代码由三部分组成：对网络接口函数的封装，百度 IoT 客户端的操作和对 MQTT 消息的处理。分别位于 baidu_iot_network_wrapper.c，baidu_iotclient.c 和 mqtt_msg_handler.c 文件中。

1）网络接口函数的封装

Paho MQTT 里已经定义了对网络接口的输入输出接口形式，如下：

int (*mqttread) (Network*, unsigned char*,int,int);

int (*mqttwrite) (Network*,unsigned char*,int,int);

所以程序中，需要将上一节中介绍的网络抽象层提供的访问网络的接口函数，封装成符合 Paho MQTT 的形式。在 baidu_iot_network_wrapper.c 中包括了封装好的接口函数，与服务器建立 TCP/TLS 连接的函数等：

- int baidu_connect_network(Network* n, const char * host_address, int port)：与百度 IoT 平台服务器建立 TCP 连接。
- int baidu_connect_network_tls(Network* n, const char *server, const int port, const char* cert, size_t cert_len)：与百度 IoT 平台服务器建立 TLS 连接。
- int mqtt_network_new(Network *network)：注册新的网络接口给 Paho MQTT。
- int mqtt_socket_recv(Network *network, unsigned char *buf, int len, int timeout)：从网络接口接收数据。
- int mqtt_socket_send(Network *network, unsigned char *buf, int len, int timeout)：向网络接口发送数据。

2）百度 IoT 客户端的操作

这部分代码，提供了函数完成连接到百度 IoT 的 MQTT 服务器，订阅和发布消息等操作。

- int baiduiothub_connect(void)：连接到百度 IoT 的 MQTT 服务器。服务器地址，MQTT 连接的用户名和密码定义在 baidu_iotclient_conf.h 头文件中。
- int baiduiot_devicestatus_update(void)：向云端发布设备的状态信息。
- int baiduiot_sub_shadow(void)：订阅云端的主题。现在只订阅了 BAIDU_DEVICE_SHADOW_DELTA_TOPIC 一个主题，开发者可以自己添加。

3）MQTT 消息的处理

需要上传到云端的消息，以及接收到的云端推送内容。在 mqtt_msg_handler.c 中的 PrepareMqttPayload 和 MQTTcallbackHandler 两个函数中进行处理。开发者可以根据自己的需要进行修改。

4.3 远程固件下载模块

收到云端推送的新固件下载信息后，程序会自动通过 HTTP 来下载新的固件，然后根据用户的选择进行更新。这个模块由 2 部分内容组成：

- 通过 HTTP 下载文件
- 将下载的固件烧写到外部 QSPI Flash 中

代码在 http_util.c 和 rfu.c 两个文件中。

4.3.1 HTTP 下载

为了及时发现文件下载时出现的错误，以及在出现网络故障或其他问题，下载被中断后不需要重新从第一个字节开始传输，并节省 RAM 的占用量。从云端下载的过程是分块进行下载的。通过 http 的 range request 请求每次最多从服务器接收 1K 字

节的数据，然后将收到的数据写到 QSPI Flash 中，接收到的数据的校验和会先被记录下来。当一个数据块接收完后，对这个数据块的校验和进行校验，如果不正确，则马上进行重传。当整个文件被接收完后，再对整个文件进行校验和的检查，如果不正确，则对整个文件重新进行下载。

http 接口函数有：http_sock_open(),http_sock_close(),http_sock_rcv()。

*请注意本示例中的 http 客户端代码仅实现了部分 http 客户端的功能来完成固件下载的功能，不是一个完整的 http 客户端实现。

4.3.2 HTTP 下载文件的格式

为了支持断点续传，分区校验，版本检查等功能，通过 HTTP 下载的文件在原始的 BIN 文件里增添了一些信息。这些信息仅在 HTTP 下载的过程中用到，最终烧写到 QSPI Flash 中的是去掉了这些冗余信息的原始 BIN 文件。

下面是文件格式的相关定义：

以固定长度的头开始，里面包括整个文件的信息。接下来是分为多个相同大小区块的数据区，每个数据区块开头也有一段描述数据，包括这块数据的起始地址，长度，checksum 等。数据区划分的大小可以通过转换工具，根据所使用的 QSPI flash 的规格进行定义。

文件的头：

名称	大小（字节）
识别码*	4
保留	4
整个 bin 文件的长度**	4
数据块的大小	4
版本信息	4

*识别码用来在一开始就可以确认下载的文件是否已转换成符合要求的格式。

**该长度不包括文件头和数据块头，结束标志等的长度，仅指实际可执行 bin 文件数据的长度。

数据块的头：

名称	大小（字节）
数据块的标识#	1
保留	3
起始地址	4
数据长度	4
Checksum*	4

*每个数据块的 Checksum

结束标志：

名称	大小（字节）
数据块的标识#	1
保留	3
Checksum*	4

*整个 bin 文件的 checksum (不包括文件头和数据块头)

数据块的标识#的定义： 0： 无效； 1： 数据； 2：文件结束

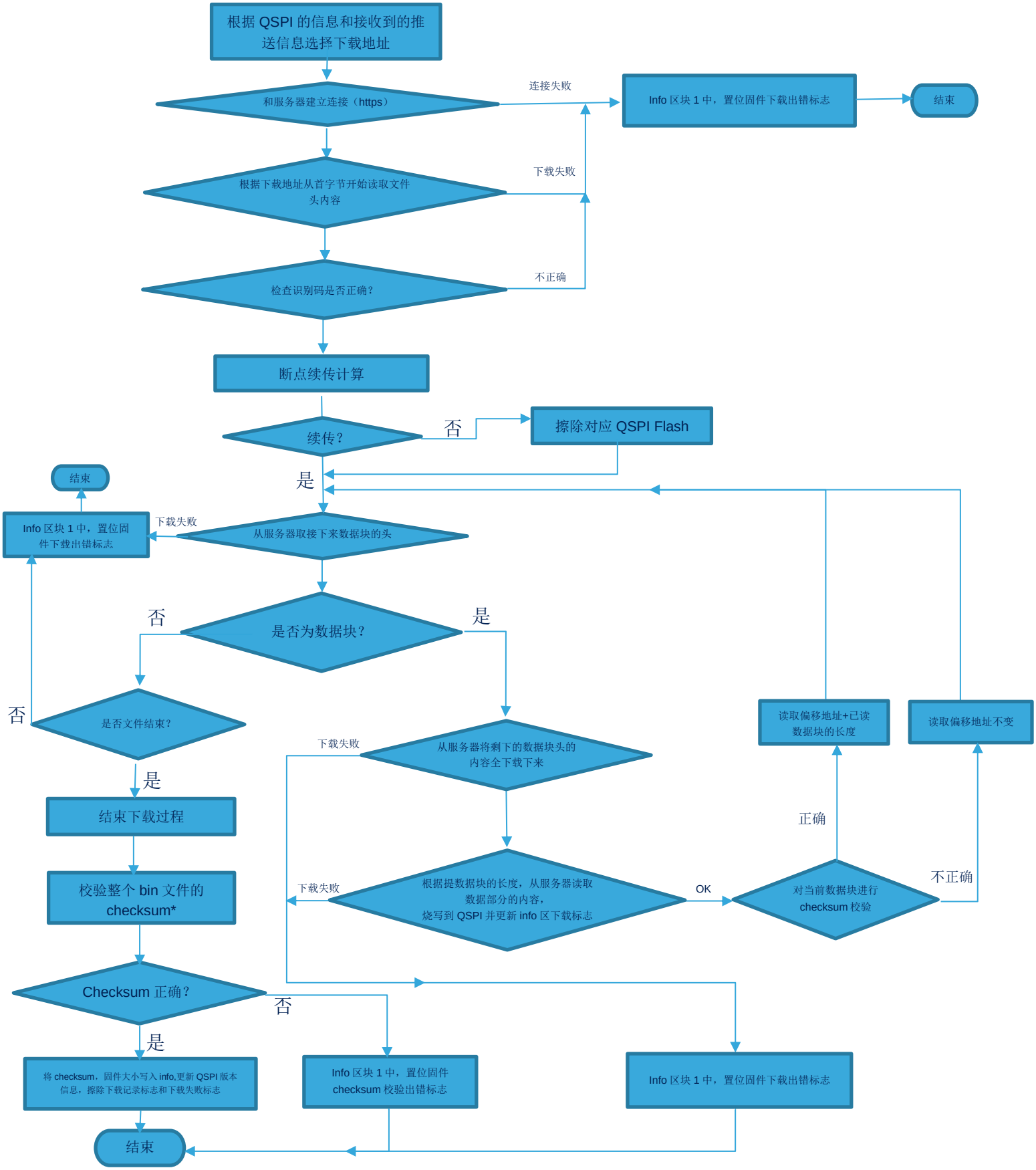
Bin 文件结构示意：

<table>
<tr><td colspan="2">文件的头（20 字节）</td><td>数据块 1 的头（16 字节）</td></tr>
<tr><td></td><td colspan="2">数据块 1 的数据</td></tr>
<tr><td colspan="3">数据块 1 的数据</td></tr>
<tr><td colspan="3">数据块 1 的数据</td></tr>
<tr><td colspan="2">数据块 2 的头（16 字节）</td><td>数据块 2 的数据</td></tr>
<tr><td colspan="3">数据块 2 的数据</td></tr>
<tr><td colspan="3">……</td></tr>
<tr><td>……</td><td>数据块 n 的头（16 字节）</td><td>数据块 n 的数据</td></tr>
<tr><td colspan="2">数据块 n 的数据</td><td>结束标志（8 字节）</td></tr>
</table>

可以通过软件包带的 FOTABinConverter 工具对 bin 文件进行转换。具体转换步骤见“STM32F769DK 云端固件升级例程使用说明”

4.3.3 下载流程图

例程中的下载流程见下图：


根据 QSPI 的信息和接收到的推送信息选择下载地址
和服务器建立连接（https）
连接失败
Info 区块 1 中，置位固件下载出错标志
结束
根据下载地址从首字节开始读取文件头内容
下载失败
检查识别码是否正确？
不正确
断点续传计算
续传？
否
擦除对应 QSPI Flash
是
结束
Info 区块 1 中，置位固件下载出错标志
下载失败
从服务器取接下来数据块的头
否
是否为数据块？
是
否
是否文件结束？
是
结束下载过程
校验整个 bin 文件的 checksum*
Checksum 正确？
否
是
将 checksum，固件大小写入 info,更新 QSPI 版本信息，擦除下载记录标志和下载失败标志
结束
下载失败
从服务器将剩下的数据块头的内容全下载下来
下载失败
根据提数据块的长度，从服务器读取数据部分的内容，烧写到 QSPI 并更新 info 区下载标志
OK
对当前数据块进行 checksum 校验
正确
不正确
读取偏移地址+已读数据块的长度
读取偏移地址不变
Info 区块 1 中，置位固件 checksum 校验出错标志
Info 区块 1 中，置位固件下载出错标志

4.4 QSPI Flash 操作接口

从云端下载的固件先被保存在 QSPI Flash 中。QSPI Flash 除了保存从云端下载的固件，还保存下载过程的一些状态信息，比如下载的版本，下载的连接，下载失败的标志等。

在 stm32f769i_discovery_qspi.c 文件中，提供了读写 QSPI Flash 的接口函数：BSP_QSPI_Erase_Sector(), BSP_QSPI_Erase_Block(), BSP_QSPI_Write(), BSP_QSPI_Read()等。

在 qspi_flash_if.c 文件中，则提供了对状态信息读写的接口函数。

4.4.1 QSPI 的 INFO 区的定义

QSPI Flash 中保存这些状态信息的区域我们把它叫做 INFO 区。

Info 区块保存在 STM32F769DK 板上的 QSPI Flash 中。占用 QSPI Flash 的前 64KB 空间。

Info 区块分为两个部分：Info1，存储擦写不太频繁的数据。比如只在烧写完成或失败后进行数据更新。Info2，存储擦写频繁的数据。比如在烧写 QSPI Flash 时，记录烧写进度的标志位。具体定义如下：

Info 区块 1：

起始地址：0x0000000

结构定义：

<table>
<tr><td>Byte0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td colspan="4">Current Version</td><td colspan="4">New Version</td><td colspan="4">Version in server</td><td>Update</td><td>Downloading</td><td>MCU 更新前固件校验失败</td><td>Reserved</td></tr>
<tr><td>QSPI 中固件 1 下载失败</td><td>QSPI 中固件 1 不完整</td><td>QSPI 中固件 2 下载失败</td><td>QSPI 中固件 2 不完整</td><td colspan="12">Reserved</td></tr>
<tr><td colspan="4">QSPI flash 中固件 1 的长度，4 字节（从云端新下载的固件）</td><td colspan="4">QSPI flash 中固件 1 的 checksum，4 字节</td><td colspan="4">QSPI flash 中固件 2 的长度（默认固件），4 字节*</td><td colspan="4">QSPI flash 中固件 2 的 checksum，4 字节*</td></tr>
<tr><td colspan="16">固件 1 下载链接，n 个字节</td></tr>
<tr><td colspan="16">固件 2 的默认下载链接, n 个字节**</td></tr>
<tr><td colspan="16">QSPI sector 大小(单位 KB)，2 字节</td></tr>
<tr><td colspan="16">Info 区块 1 的 checksum，4 字节</td></tr>
</table>

*固件 2 的长度和 checksum 通过 st-link utility 烧写到 QSPI flash

**固件 2 的默认下载链接，在首次上电运行时烧写到 INFO 区（INFO 区为空时）

Current Version (4 字节): 当前运行的 APP 的版本号。成功更新 MCU 片上 Flash 的程序后，更新此版本号。APP 上电运行时，应该检查 Info 区的 Current Version 版本号是否和当前版本号一致。并将当前的版本号写到 Info 区。

New Version (4 字节): 当前下载在 QSPI flash 中固件 1 的版本号。固件 1 是指接收到推送信息后，从云端新下载到 QSPI flash 的固件。下载过程成功完成后，更新此字段。

Version in server (4 字节): 服务器上的最新版本号。接收到服务器推送的版本信息后，写到 info 区。根据这个版本号，可以在下载程序被意外中断后，判断是否需要重新或者继续从服务器下载程序（上电时，会对比 New Version 和 Version in server，来检查是否有新程序需要下载）。

Current version/ New Version/ Version in server 版本号的格式：

Byte0	1	2	3
Major version	Minor version	Test version	Not used

Update 标志（1 字节）：当用户选择升级后（按下 USER 按键），该标志字节置为 0x01。系统启动后，bootloader 会检查该字段结合版本信息来决定是否要更新固件。固件全部更新完成后，擦除该标志（0xFF）。

Downloading 标志（1 字节）：0x01 表示正在从服务器下载新固件到 QSPI。固件全部下载，并成功烧写到 QSPI 后，擦除该标志（0xFF）

MCU 更新前固件校验失败标志（1 字节）:0x01 表示更新 MCU 片上 flash 前对 QSPI 中的固件校验失败，说明 QSPI 中的固件不完整，需要重新从云端下载。更新 MCU 的片上 flash 前，校验 QSPI 固件失败时，置该标志为 0x01. 从云端重新成功下载到 QSPI 后，擦除该标志（0xFF）。

QSPI flash 中固件 1 下载失败标志（1 字节）：固件 1 是指接收到推送信息后，从云端新下载到 QSPI 的固件。0x01 表示固件 1 下载到 QSPI 时失败，可能是校验出错，可能是网络连接或下载失败。固件全部下载，并成功烧写到 QSPI 后，擦除该标志（0xFF）。

QSPI flash 中固件 1 不完整：0x01 表示从 QSPI 将固件 1 烧写到 MCU 片上 flash 时对固件的 checksum 校验失败。当重新成功下载固件到固件 1 位置后，擦除该标志（0xFF）。

QSPI flash 中固件 2 下载失败标志（1 字节）：0x01 表示固件 2 下载到 QSPI 时失败，可能是校验出错，可能是网络连接或下载失败。固件全部下载，并成功烧写到 QSPI 后，擦除该标志（0xFF）。

QSPI flash 中固件 2 不完整（1 字节）：0x01 表示从 QSPI 将固件 2 烧写到 MCU 片上 flash 时对固件的 checksum 校验失败。当重新成功下载固件到固件 2 位置后，擦除该标志（0xFF）。

QSPI flash 中固件 1 的长度（4 字节）：固件 1 是指从云端新下载到 QSPI 的固件。将固件 1 下载到 QSPI 后，将固件 1 的长度记录在该字段。

QSPI flash 中固件 1 的 checksum 值（4 字节）：使用校验和的方式。将固件 1 下载到 QSPI 后，将固件 1 的 checksum 记录在该字段。

QSPI flash 中固件 2 的长度（4 字节）：固件 2 是指出厂时就保存在 QSPI 中的默认固件，通过工具预先烧录在 QSPI flash 中。该长度值通过 ST-LINK Utility 预先烧录在 QSPI flash 中。

QSPI flash 中固件 2 的 checksum 值（4 字节）：使用校验和的方式。通过 ST-LINK Utility 预先烧录在 QSPI flash 中。

固件 1 下载链接: 新固件在服务器的地址。使用 UTF-8 编码。这部分的结构如下：

Byte1	String length MSB
Byte2	String length LSB
Byte3	UTF-8 Encoded String Data, if length > 0
…	

固件 2 默认下载链接: 出厂的默认固件在服务器的地址。使用 UTF-8 编码。这部分的结构如下：

Byte1	String length MSB
Byte2	String length LSB
Byte3	UTF-8 Encoded String Data, if length > 0
…	

上电后，程序会检查 INFO 区是否为空，如果为空则将程序中默认的下载地址烧写到 INFO 区对应的位置。

Download sector size（2 字节）: 这里的下载是指从服务器下载到外部 QSPI flash。所以 sector 是指 QSPI Flash 数据手册中所定义的 sector 大小，单位是 KB。跟 bin 文件中的数据块划分大小应该一致。预先烧录在 QSPI 中。

Info 区块 1 的 checksum（4 字节）：Info 区块 1 的所有内容的校验和，不包括 Info 区块 1 checksum 本身.

Info 区块 2：

起始地址：0x0008000，该区块保存在下载烧写过程中需要频繁更新的字段

结构定义：

Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Download Flag															
……															
Checksum Record															

Download Flag (512 字节)：一个字节对应表示 QSPI 中一个 sector 的烧写状态。0 表示已烧写，0xff 表示还未烧写。当全部下载完成后，擦除所有 Download Flag，并擦除 Info 区块 1Downloading 位标志。

*根据 Download sector size 和 Download flag 可以计算出下载终止的位置，实现断点续传。

Checksum Record：记录从云端下载时，每下载成功一个数据块，当时计算的 checksum 值（4 字节）。记录该值，是因为断点续传时，不是从第一个数据块开始传输，不知道前面数据块的 checksum 值，如果要算出整个文件的 checksum 就需要记录之前的 checksum 值。

5. 基于本软件进行再次开发

在这一章内，将介绍如何基于本例程开发自己的程序。

5.1 使用自己的百度 IoT 服务

如果你希望使用自己的百度 IoT 服务进行 MQTT 通信，首先需要在百度的 IoT 平台创建自己的服务（如何创建请参考“STM32F769DK 云端固件升级例程使用说明”）。然后做下面几个简单的改动，就可以将板子连接到你自己的百度 IoT 服务器上了。

在百度的 IoT 平台上创建好服务后，你就会得到以下信息。

包括云端服务器的地址，连接的用户名和密码等。

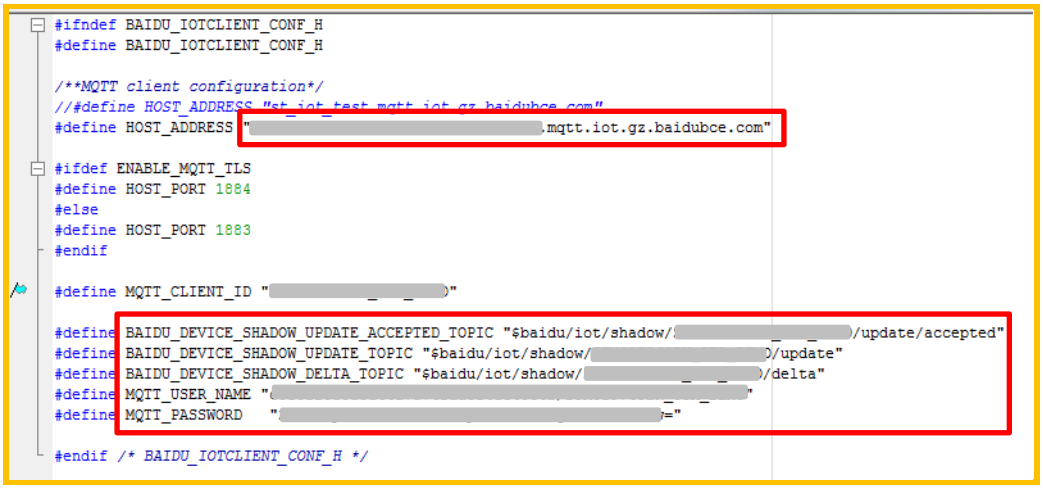
根据你创建的物影子的名称，也知道了：

将设备状态更新到云端的设备影子使用的主题是：$baidu/iot/shadow/{deviceName}/update。

获取设备影子更新状态使用的主题是：$baidu/iot/shadow/{deviceName}/update/accepted。

这里的 deviceName 就是建立的物影子的名称。

将这些信息加到 baidu_iotclient_conf.h 文件中，见下图：



```
#ifndef BAIDU_IOTCLIENT_CONF_H
#define BAIDU_IOTCLIENT_CONF_H

/**MQTT client configuration*/
//#define HOST_ADDRESS "st_iot_test.mqtt.iot.gz.baidubce.com"
#define HOST_ADDRESS "[redacted].mqtt.iot.gz.baidubce.com"

#ifdef ENABLE_MQTT_TLS
#define HOST_PORT 1884
#else
#define HOST_PORT 1883
#endif

#define MQTT_CLIENT_ID "[redacted]"

#define BAIDU_DEVICE_SHADOW_UPDATE_ACCEPTED_TOPIC "$baidu/iot/shadow/[redacted]/update/accepted"
#define BAIDU_DEVICE_SHADOW_UPDATE_TOPIC "$baidu/iot/shadow/[redacted]/update"
#define BAIDU_DEVICE_SHADOW_DELTA_TOPIC "$baidu/iot/shadow/[redacted]/delta"
#define MQTT_USER_NAME "[redacted]"
#define MQTT_PASSWORD  "[redacted]="

#endif /* BAIDU_IOTCLIENT_CONF_H */
```

MQTT_CLIENT_ID 用户自己定义就可以，保证在同一个实例下没有重复就行。

重新编译就 Ok 了。

5.2 连接其他云的 IoT 平台

在本示例中连接百度 IoT 平台的实现是基于 Paho MQTT 来做的。如果想连接到其他云的 IoT 平台，可以有两种方法：

第一种方法，搞清楚这个 IoT 平台的连接过程，还是基于 Paho MQTT，自己在上面实现连接。这种方法，涉及到的就是 Baidu MQTT 客户端实现的这部分代码，包括 baidu_iot_network_wrapper.c，baidu_iotclient.c，mqtt_msg_handler.c 等文件。

第二种方法，如果该 IoT 平台提供的有 SDK，可以使用对应的 SDK 来替代 Paho MQTT。再基于这个 SDK 来做应用层的开发。同样 Baidu MQTT 客户端实现这部分代码，包括 baidu_iot_network_wrapper.c，baidu_iotclient.c，mqtt_msg_handler.c 等文件需要重写。一般云平台提供的 SDK 会有和它所支持的功能相关的更丰富的 API 可使用，并且有一些保证连接可靠性的处理。

5.3 使用其他的对象存储服务

在本示例中，使用了百度的 BOS 来保存要下载的文件。你也可以用任何其他的支持 HTTP1.1 的存储服务来保存这些文件。如果需要支持 HTTPS，那么在 MCU 的代码端，你还需要替换对应的根证书。百度云的 BOS 服务器使用的证书是 VeriSign_Class3_Public_Primary_CA_G5，证书内容放在 certs.h 中。

你需要：

1. 将新的证书内容放在 certs.h 中
2. 修改 http_util.c 中，httphost_cas_pem 的定义。改成你使用的证书名
3. 根据调试的情况，修改 mbedTLS 的配置（关于 mbedTLS 的配置见 5.5 节）

5.4 从以太网切换到 WIFI/3G

对于不同的物理连接方式，程序中已经通过网络抽象层进行了封装。切换不同的网络接口，不会影响到应用层的代码。所以虽然现在例程提供的是以太网的连接方式，要切换其他的连接方式也是很容易的。

根据 4.1 节“网络抽象层”的说明，从以太网切换到 WIFI 或者 3G 的话，只需要将 net_tcp_lwip.c，换成 net_tcp_wifi.c 或者 net_tcp_c2c.c。根据不同 WIFI/3G 模块提供的驱动更新 net_tcp_wifi.c 或者 net_tcp_c2c.c 文件中的接口函数，但要保证其接口函数形式不变。

以从以太网换到 wifi 来举例：

1. 从工程中去掉 net_tcp_lwip.c，以及 lwip_net.c, Ethernetif.c 等文件；
2. 向工程中添加 net_tcp_wifi.c，wifi_net.c,以及 WIFI 模块的驱动文件；
3. 根据 WIFI 模块的驱动，重新修改 net_tcp_wifi.c，wifi_net.c 文件；
4. 保证 net_tcp_wifi.c，wifi_net.c 中的函数接口不变，比如还是 net_sock_open_tcp_wifi()，还是 net_if_init()。

5.5 TLS 的配置

5.5.1 宏定义

在 IAR 工程的编译配置中，mbedTLS 相关的宏定义有三个：USE_MBED_TLS，ENABLE_MQTT_TLS 以及 MBEDTLS_CONFIG_FILE。

- USE_MBED_TLS：定义 USE_MBED_TLS 后，在网络抽象层才会支持 TLS 连接。所以如果希望支持 HTTPS 或者 MQTT+TLS，必须定义宏 USE_MBED_TLS。打开宏 USE_MBED_TLS 后，固件下载时，会自动通过 URL 地址解析来判断是 http 还 https，自动建立 TCP 或者 TLS 连接。
- ENABLE_MQTT_TLS：对于 MQTT 通信协议，要支持 TLS，除了定义 USE_MBED_TLS，还要定义宏 ENABLE_MQTT_TLS。应用程序才会在 TLS 安全连接的基础上进行 MQTT 通信，进一步保证通信的安全。
- MBEDTLS_CONFIG_FILE：使用 mbedTLS 时，可以通过修改 mbedTLS 的配置文件来对其功能进行配置和裁剪。mbedTLS 协议栈的 include/mbedtls/config.h 是默认的配置文件。如果不想使用默认配置文件，可以通过定义 MBEDTLS_CONFIG_FILE 来很方便的指定自己的配置文件。例如，在本示例中通过在 IAR 的预编译项中定义宏 MBEDTLS_CONFIG_FILE，指定了配置文件为 mbedtls_config.h。 在 mbedtls_config.h 中，又 include 了头文件 baidu_mbedtls_config.h。

  所以程序是根据 baidu_mbedtls_config.h 中的定义对 mebdTLS 进行配置。

  一些宏在 baidu_mbedtls_config.h 中重新进行了定义以减少 RAM,ROM 的占用量，比如：

```
#define MBEDTLS_MPI_WINDOW_SIZE            1 /**< Maximum windows size used. */
```

```
#define MBEDTLS_MPI_MAX_SIZE            512 /**< Maximum number of bytes for usable MPIs.
*/
#define MBEDTLS_ECP_WINDOW_SIZE           2 /**< Maximum window size used */
#define MBEDTLS_ECP_FIXED_POINT_OPTIM     0 /**< Disable fixed-point speed-up */
      #define MBEDTLS_ENTROPY_MAX_SOURCES                2//20 /**< Maximum number of
sources supported */
      #define MBEDTLS_SSL_MAX_CONTENT_LEN             1024*6 /**< Maxium fragment length
in bytes, determines the size of each of the two internal I/O buffers */
```

5.5.2 根证书

TLS 为 MQTT 和 HTTP 通信提供了加密的通信通道。在建立通信之前，客户端先向服务器端索要并验证服务器的证书，双方约定加密方式，协商生成加密密钥，最后采用密钥进行加密通信。客户端要验证服务器发来的证书，就需要先保存一个根证书，这个根证书里包含了可以验证服务器发来证书的公钥，由 CA 机构发布。一般可以从要连接的服务器网站上下载到。

根证书的内容在代码里，保存在 certs.h 文件里的。

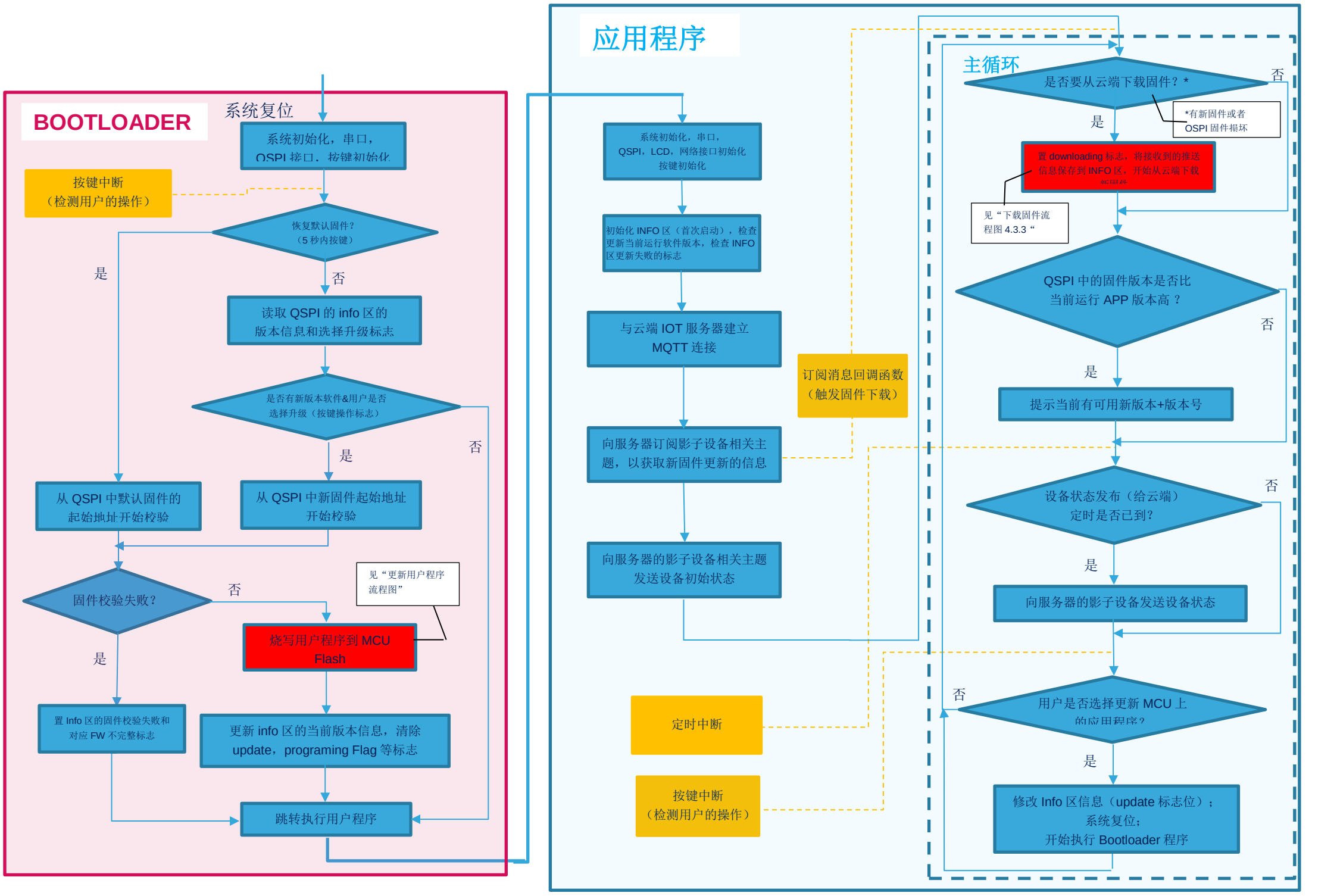
```
//for baidu iot hub
#define BAIDU_CA_CRT                                                \
"-----BEGIN CERTIFICATE-----\r\n"                                    \
"MIIENjCCAx6gAwIBAgIBATANBgkqhkiG9w0BAQUFADBvMQswCQYDVQQGEwJTRTEU\r\n"  \
"MBIGA1UEChMLQWRkVHJ1c3QgQUIxJjAkBgNVBAsTHUFkZFRydXN0IEV4dGVybmFs\r\n"  \
"IFRUUCBOZXR3b3JrMSIwIAYDVQQDExlBZGRUcnVzdCBFeHRlcm5hbCBDQSBSb290\r\n"  \
"MB4XDTAwMDUzMDEwNDgzOFoXDTIwMDUzMDEwNDgzOFowbzELMAkGA1UEBhMCU0Ux\r\n"  \
"FDASBgNVBAoTC0FkZFRydXN0IEFCMSYwJAYDVQQLEx1BZGRUcnVzdCBFeHRlcm5h\r\n"  \
"bCBUVFAgTmV0d29yazEiMCAGA1UEAxMZQWRkVHJ1c3QgRXh0ZXJuYWwgQ0EgUm9v\r\n"  \
"dDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALf3GjPm8gAELTngTlvt\r\n"  \
"H7xsD821+iO2zt6bETOXpClMfZOfvUq8k+0DGuOPz+VtUFrWlymUWoCwSXrbLpX9\r\n"  \
"uMq/NzgtHj6RQa1wVsfwTz/oMp50ysiQVOnGXw94nZpAPA6sYapeFI+eh6FqUNzX\r\n"  \
"mk6vBbOmcZSccbNQYArHE504B4YCqOmoaSYYkKtMsE8jqzpPhNjfzp/haW+710LX\r\n"  \
"a0Tkx63ubUFfclpxCDezeWWkWaCUN/cALw3CknLa0Dhy2xSoRcRdKn23tNbE7qzN\r\n"  \
"E0S3ySvdQwAl+mG5aWpYIxG3pzOPVnVZ9c0p10a3CitlttNCbxWyuHv77+ldU9U0\r\n"  \
"WicCAwEAAaOB3DCB2TAdBgNVHQ4EFgQUrb2YejS0Jvf6xCZU7wO94CTLVBowCwYD\r\n"  \
"VR0PBAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wgZkGA1UdIwSBkTCBjoAUrb2YejS0\r\n"  \
"Jvf6xCZU7wO94CTLVBqhc6RxMG8xCzAJBgNVBAYTAlNFMRQwEgYDVQQKEwtBZGRU\r\n"  \
"cnVzdCBBQjEmMCQGA1UECxMdQWRkVHJ1c3QgRXh0ZXJuYWwgVFRQIE5ldHdvcmsx\r\n"  \
"IjAgBgNVBAMTGUFkZFRydXN0IEV4dGVybmFsIENBIFJvb3SCAQEwDQYJKoZIhvcN\r\n"  \
"AQEFBQADggEBALCb4IUlwtYj4g+WBpKdQZic2YR5gdkeWxQHIzZlj7DYd7usQWxH\r\n"  \
"YINRsPkyPef89iYTx4AWpb9a/IfPeHmJIZriTAcKhjW88t5RxNKWt9x+Tu5w/Rw5\r\n"  \
"6wwCURQtjr0W4MHfRnXnJK3s9EK0hZNwEGe6nQY1ShjTK3rMUUKhemPR5ruhxSvC\r\n"  \
"Nr4TDea9Y355e6cJDUCrat2PisP29owaQgVR1EX1n6diIWgVIEM8med8vSTYqZEX\r\n"  \
"c4g/VhsxOBi0cQ+azcgOno4uG+GMmIPLHzHxREzGBHNJdmAPx/i9F4BrLunMTA5a\r\n"  \
"mnkPIAou1Z5jJh5VkpTYghdae9C8x49OhgQ=\r\n"  \
"-----END CERTIFICATE-----\r\n"
```

```
const char mqtthost_cas_pem[] = BAIDU_CA_CRT;
```

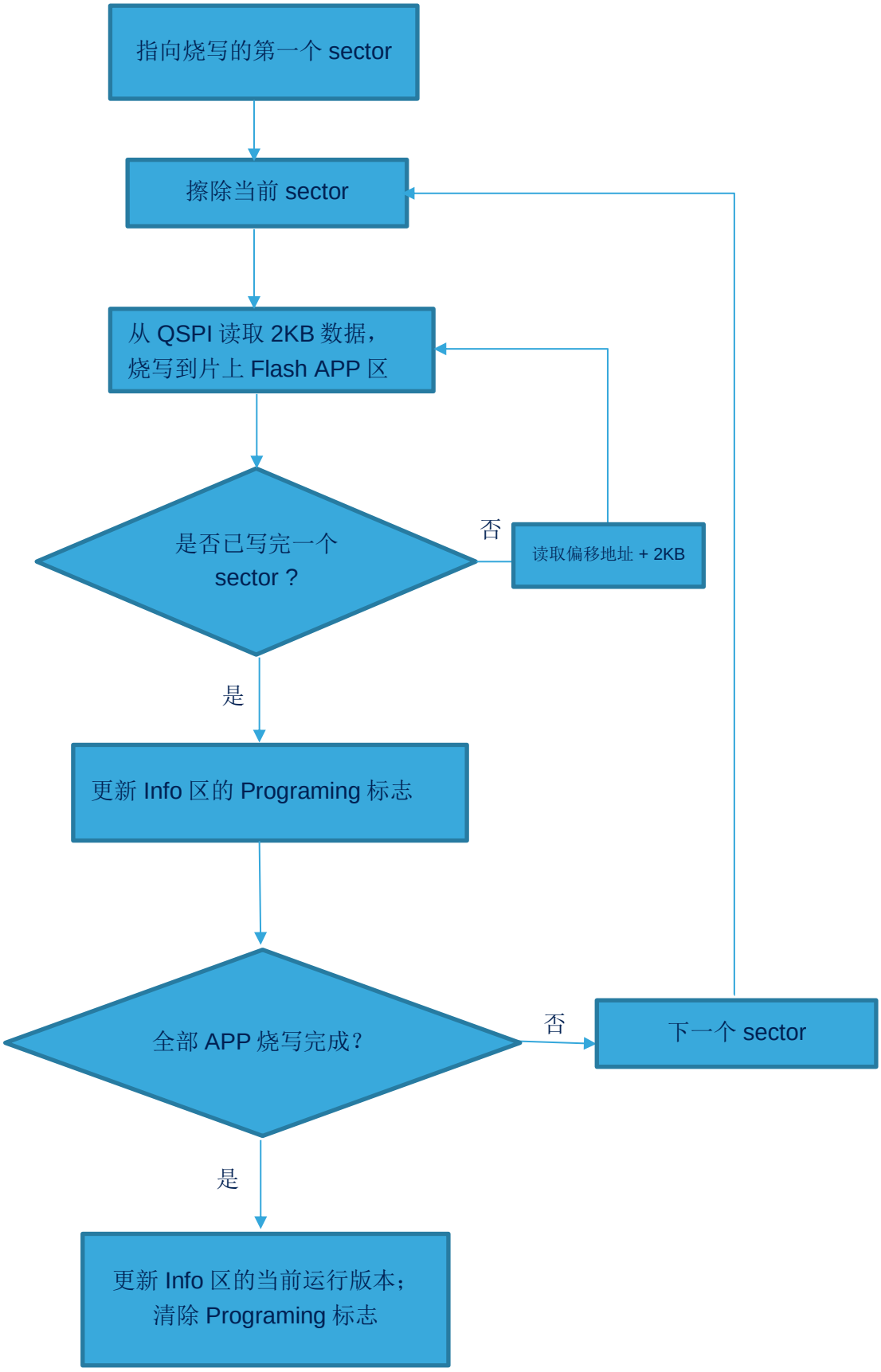
如果要换其他的 IoT 平台，或者将固件保存在其他的网站上，同时又要用 TLS 连接，就需要将新的根证书添加进来。并将 mqtthost_cas_pem（在 baidu_iotclient.c 中定义）或者 httphost_cas_pem（在 http_util.c 中定义）替换成对应的证书名。

5. 系统流程图

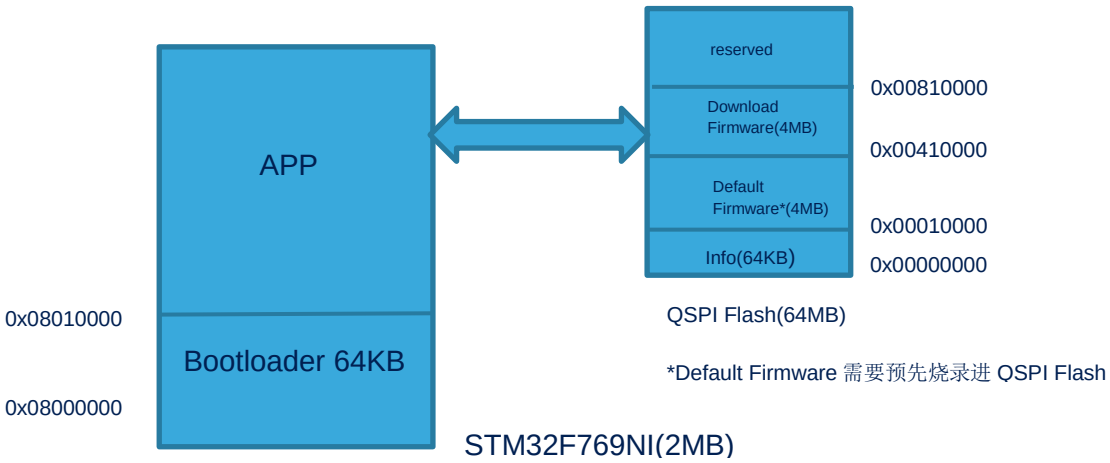
下面的系统流程图，描述了包括 bootloader 在内的整个系统工作的流程。复位后，程序从 bootloader 启动，再跳到用户应用程序执行。


BOOTLOADER
系统复位
系统初始化，串口，
QSPI 接口，按键初始化
按键中断
（检测用户的操作）
恢复默认固件？
（5 秒内按键）
是
否
读取 QSPI 的 info 区的
版本信息和选择升级标志
是否有新版本软件&用户是否
选择升级（按键操作标志）
是
否
从 QSPI 中默认固件的
起始地址开始校验
从 QSPI 中新固件起始地址
开始校验
固件校验失败？
否
是
见“更新用户程序
流程图”
烧写用户程序到 MCU
Flash
置 Info 区的固件校验失败和
对应 FW 不完整标志
更新 info 区的当前版本信息，清除
update，programing Flag 等标志
跳转执行用户程序
应用程序
系统初始化，串口，
QSPI，LCD，网络接口初始化
按键初始化
初始化 INFO 区（首次启动），检查
更新当前运行软件版本，检查 INFO
区更新失败的标志
与云端 IOT 服务器建立
MQTT 连接
订阅消息回调函数
（触发固件下载）
向服务器订阅影子设备相关主
题，以获取新固件更新的信息
向服务器的影子设备相关主题
发送设备初始状态
定时中断
按键中断
（检测用户的操作）
主循环
是否要从云端下载固件？*
否
是
*有新固件或者
OSPI 固件损坏
置 downloading 标志，将接收到的推送
信息保存到 INFO 区，开始从云端下载
新固件
见“下载固件流
程图 4.3.3 “
QSPI 中的固件版本是否比
当前运行 APP 版本高 ？
否
是
提示当前有可用新版本+版本号
设备状态发布（给云端）
定时是否已到？
否
是
向服务器的影子设备发送设备状态
用户是否选择更新 MCU 上
的应用程序？
否
是
修改 Info 区信息（update 标志位）；
系统复位；
开始执行 Bootloader 程序

6. Bootloader 中更新用户程序流程图



7. STM32F769DK 的内存划分



参考文档

- STM32F769DK 云端固件升级例程使用说明
- UM2033-Discovery kit with STM32F769NI MCU
- STM32 OTA 例程之 cJSON 使用
- STM32 OTA 例程之 ESP8266 使用
- STM32 OTA 例程之 Paho embedded C 使用

Revision History

日期	版本	说明
2017.12.18	1	初始版本
2018.4.20	2	添加对 wifi 模块的支持，更新图例

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。