MiCO Documentation Working Group (MDWG) Jenny Liu

Track Number: 0003CN MXCHIP Co., Ltd

Version: 1.1 2015.9.8.

Category: Technical Manual

MiCOKit 固件开发手册

摘要（Abstract）

本文档以 MiCO 总动员开发板固件为例，介绍 MiCOKit 固件开发方法，使用户对 MiCO 智能硬件开发有个初步认识，方便 MiCO-IoT 物联网开发者使用 MiCOKit 开发板及 FogCloud 进行物联网应用开发。

适用对象（Status of This Document）

本文档适用于初级 MiCOKit 开发者，对所有 MiCO-IoT 物联网智能硬件开发者公开。

获取更多帮助（More Help）

MiCO 开发团队向您推荐：MiCO 开发者学习网站：http://mico.io/（至开发者中心），获取更多最新资料。

手机微信“扫一扫”关注：“MiCO 总动员”公众号，获取 MiCO 团队小伙伴最新活动信息。

登录上海庆科官方网站：http://mxchip.com/，获取公司最新产品信息。

目 录

图目录

表目录

1. 概述

MiCOKit 固件基于庆科 MiCO 物联网操作系统，快速接入 Wi-Fi 网络；并通过庆科的 FogCloud 云完成设备和手机 APP 之间的交互，从而实现物联网应用的各种功能。

本文档主要介绍 MiCOKit 固件开发方法以及所涉及到的相关内容，主要包括 MiCO 系统、FogCloud 云接入、以及固件 App 框架，帮助开发者深入理解 MiCOKit 的工作机制，以便进行更深入的二次开发工作。

2. 固件使用方法

MiCOKit 固件开发所需的相关学习资料和开发工具，在上海庆科 MiCO 开发者网站：mico.io 的 WiKi 中心提供。主要包括：开发环境准备，第一个 MiCO 应用程序，在线调试方法，MiCO 系统 Demo 应用实例工程，MiCOKit 相关资料文档等。

2.1. 固件 SDK 使用步骤

（1）登录 MiCO 开发者网站 mico.io –WiKi 中心，参考 MiCO SDK 开始学 - 开发环境准备，安装集成开发环境，包括：

a. 安装 IAR workbench for ARM on Windows(7.30.1 及以上);

b. 安装 ST-LINK 或 J-LINK 驱动；

c. 安装 USB 虚拟串口驱动（FTDI）;

d. 安装 secureCRT 串口调试工具；

e. 安装 TCP/UDP 测试工具；

f. 安装 HFS ~ HTTP File Server 2.3e

（2）在 MiCOKit 发布中心页面，下载最新 MiCOKit SDK （如 SDK_MiCOKit_V2.2.0.6），内含 MiCOKit 的 Demo 工程（微信端控制和 APP 端控制）;

（3）打开 MiCOKit SDK 固件工程，编译代码，下载到 MiCOKit 开发板中；

下载方法请参考开发者网站 mico.io 中 wiki 中心 – MiCO SDK 开始学中：MiCO SDK 固件下载。

（4）按开发板上的 Reset 键，运行程序，可通过串口 LOG 查看固件运行情况，并使用 MiCOKit 配套的手机 APP-MiCO 总动员的测试功能。

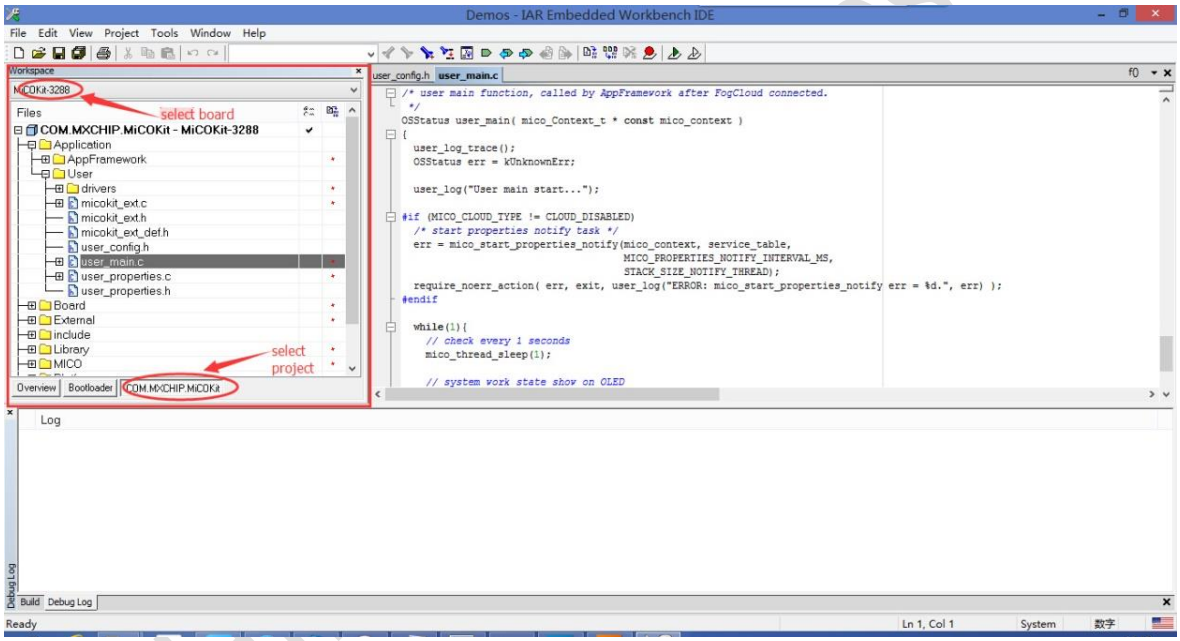
2.2. 实例工程说明

（1）以 MiCOKit-3288 为例，工程路径： \Projects\STM32F4xx。

Figure 1. 工程路径

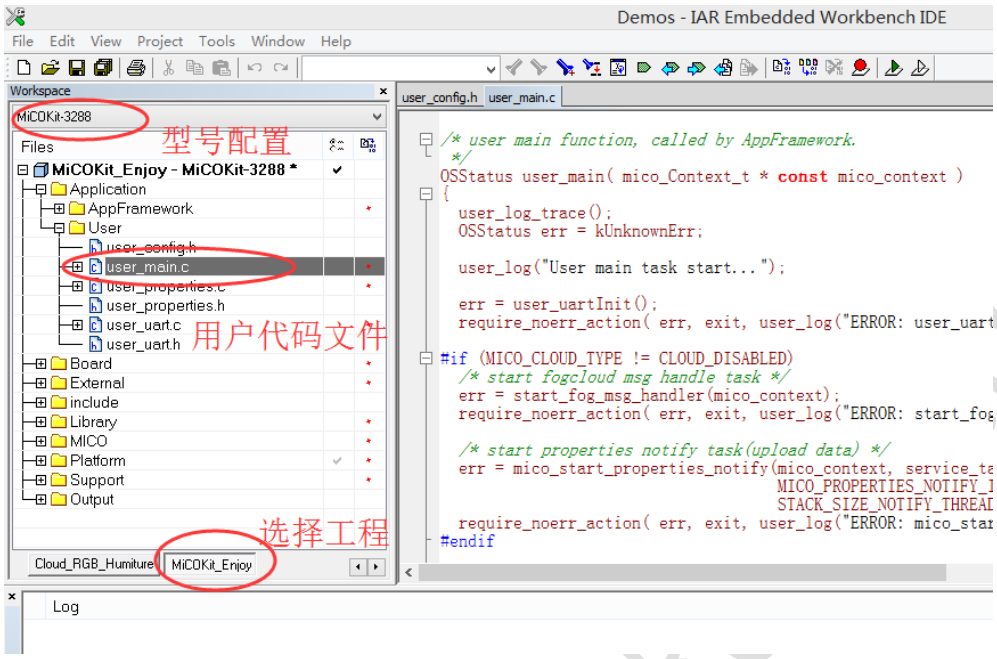
（2）使用 IAR 打开\Projects\STM32F4xx\Demo.eww 工程环境。

Figure 2. 工程环境



（3）选择 MiCOKit_Enjoy 工程，配置选择 MiCOKit-3288，如下图中标出。

Figure 3. 配置工程



（4）查看（修改）用户代码：

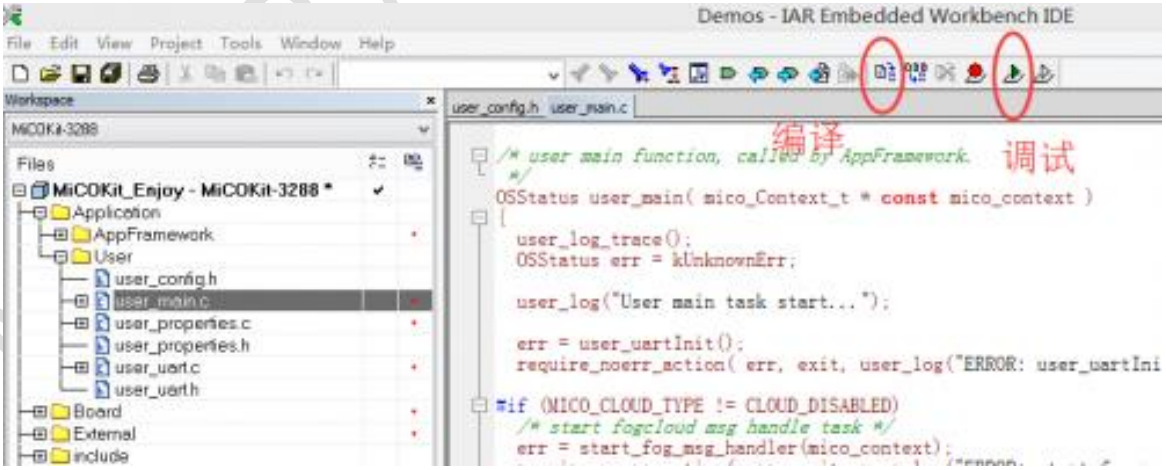
工程中 Application/User/下的代码。

（5）编译、调试、下载。

详细的编译，调试，下载方法，请参考：

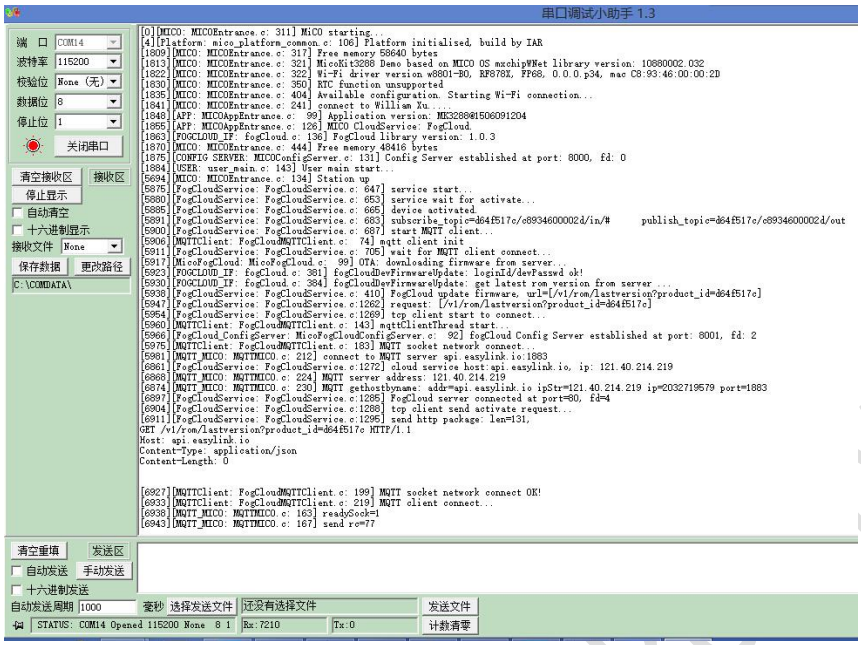
开发者网站 mico.io 中 wiki 中心 – MiCO SDK 开始学中：第一个“Hello World”程序。

Figure 4. 编译、下载、调试



（6）连接好 USB，PC 上配置好串口，查看设备运行 LOG：

Figure 5. 查看设备运行 LOG



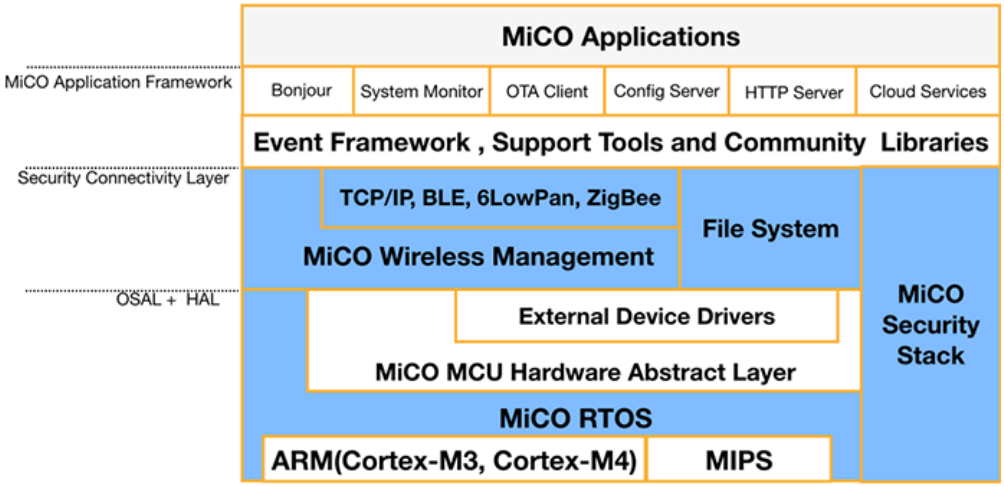
（7）手机 APP 登陆、设备配网、设备控制（详细操作方法见《MiCOKit 用户手册》）。

3. MiCO

MiCO(Micro-controller based Internet Connectivity Operating System 是一个面向智能硬件优化设计的、运行在微控制器上的、高度可移植的操作系统和中间件开发平台。MiCO 作为独立的系统，拥有开放架构，还包括了底层芯片驱动、无线网络协议、射频控制技术、安全、应用框架等，能够帮助 IoT 设备开发者降低软件开发难度，快速形成可以量产的产品。

MiCO 内含一个面向 IoT 设备的实时操作系统内核， 特别适合运行在能量受限的微控制设备上。此外，MiCO 还包含了网络通信协议栈，安全算法和协议，硬件抽象层，编程工具等开发 IoT 必不可少的软件功能包，MiCO 系统内核框架结构如下图所示。

Figure 6. MiCO 系统内核



MiCO 提供 MCU 平台的抽象化， 使得基于 MiCO 的应用程序开发不需要关心 MCU 具体件功能的实现，

通过 MiCO 中提供的各种编程组件快速构建 IoT 设备中的软件，配合 MiCOKit 开发套件实现快速产品原型开发。

MiCO 的关键特性

- 支持多种网络协议栈，并持续增加中：Bluetooth low energy，以太网，Wi-Fi，ZigBee，6LoWPAN；
- 全自动高效功耗管理；
- 支持多种 MCU 体系结构：Cortex-M 系列，MIPS 等，提供 MCU 平台级的抽象化，标准外设驱动接口；
- 完整的网络安全算法和协议栈，支持常用的传输层安全协议 TLS；
- 方便易用的应用程序框架，使 MiCO 应用安全地直达云端；
- 支持多种 MCU 常用的开发调试环境，可以进行硬件仿真。提供命令行接口和标准调试信息输出，方便实现运行中的调试；
- 提供完整的设备量产技术，如引导程序，量产烧录，远程网络升级服务，生产测试程序等；
- 基于 MiCO 系统开发的 IoT 设备软件已经成功地运行在大量的商品上，是一个被证明了的，安全，稳定，可靠的软件平台。

MiCO 详细资料请查看 MiCO 开发者网站：http://mico.io/?page_id=31

4. FogCloud

FogCloud 是专门为智能硬件提供后台支持的云服务平台，实现了硬件产品与智能手机及云端服务互连，为多种行业提供云端解决方案。目前庆科云可以提供包括设备云端互联、MiCO 软件 OTA 升级、阿里智能云、微信等第三方公有云接入等支持服务。MiCOKit 接入 FogCloud 工作架构图：

Figure 7. MiCOKit 接入 FogCloud 工作架构

设备接入 FogCloud 流程：

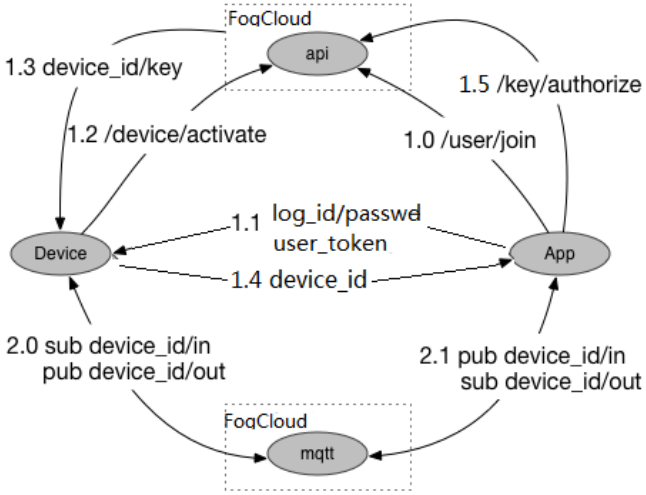
（1）申请产品 ID 和 KEY；

（2）设备使用产品 ID/KEY 向 FogCloud 激活，获取设备 ID 和 KEY；

（3）设备使用设备 ID/KEY 连接 FogCloud 消息服务器，收发消息；

（4）设备从云端注销（删除设备）。

Figure 8. 设备接入 FogCloud 流程



FogCloud 详细接入流程请参考 FogCloud Wiki: http://api.easylink.io/wiki/

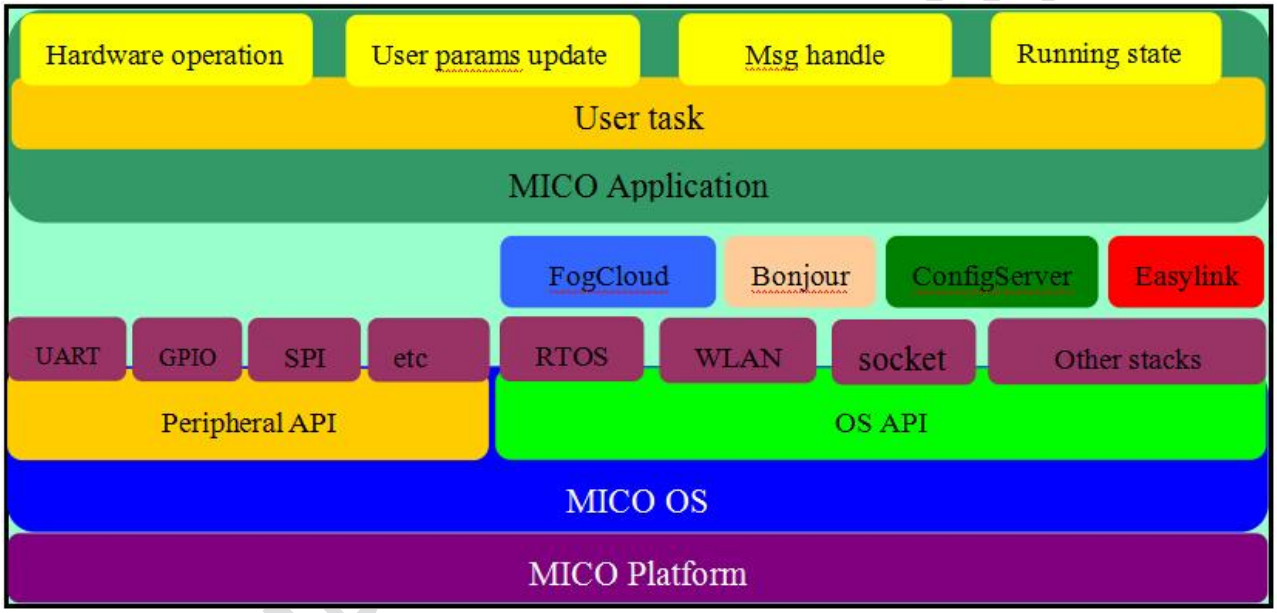
5. AppFramework

MiCOKit 基于 MiCO 操作系统，集成 FogCloud 云接入功能，给开发者提供了一套简单易用的固件应用程序框架，用户只需要修改控制具体硬件模块的代码即可测试相应的功能，无需关心设备网络连接、云接入、手机 APP 软件界面的修改等工作，大大节约了固件开发者的开发工作。

MiCOKit App Framework 默认接入 FogCloud 云 采用抽象的方法将设备上的各个模块抽象成服务(service)，将每个模块的各个功能点抽象成属性（ property ）；设备通过设备描述表的形式将它所具有的功能告诉 FogCloud 云端以及配套的手机 APP，手机 APP 根据设备描述表中的内容来自动生成设备控制界面，并且通过读写各个属性值来控制设备上的功能模块。

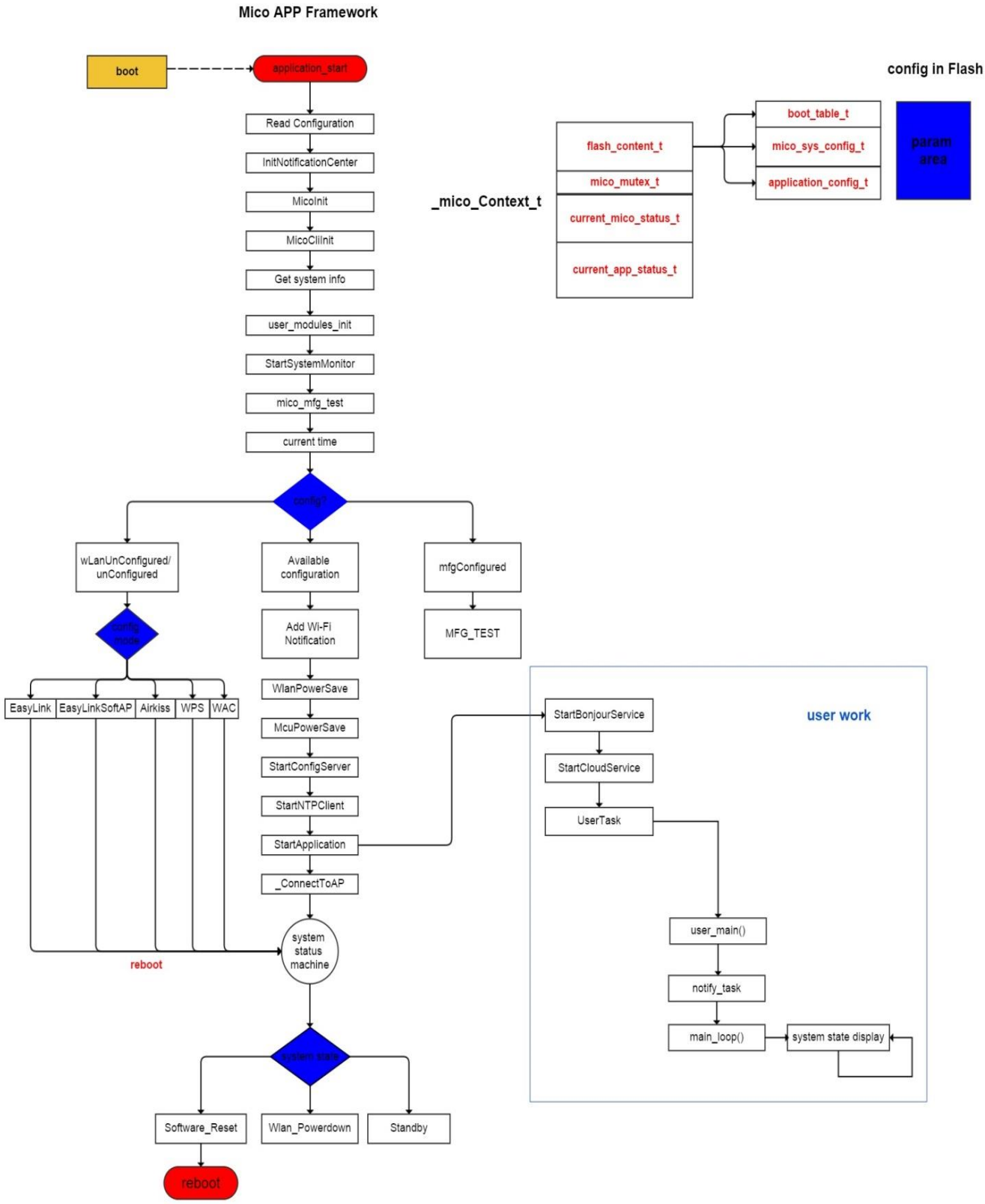
5.1. MiCOKit 固件结构图

Figure 9. MiCOKit 固件结构图



5.2. 固件流程图

Figure 10. 固件流程图



5.3. MiCOKit 设备抽象

（1）设备描述表

将 MiCO 设备所具有的功能模块（如开关、LED、串口等外设）抽象成可访问的服务（service）；将每个模块所具有的功能（开关的状态、LED 灯的亮度值等）抽象成可读写的属性（property）。每一个 service/property 都分配一个固定的 iid，作为访问标识。APP 从设备获取到该描述表之后，就可以描绘出整个设备所具有的功能模块，展示给用户，并且可以使用 iid 来读写设备的每一个属性，从而完成对设备的控制。

设备描述表在固件中的由 servcie_table 数组来定义，示例如下：

Figure 11. 设备描述表定义

```
const struct mico_service_t    service_table[]  = {

  [0] = {

    .type = "public.map.service.dev_info",       // service 1: dev_info (uuid)

    .properties  = {

      [0] = {

        .type = "public.map.property.name",     // device name uuid

        .value = &(g_user_context.config.dev_name),

        .value_len = &(g_user_context.config.dev_name_len),

        .format = MICO_PROP_TYPE_STRING,

        .perms = (MICO_PROP_PERMS_RO | MICO_PROP_PERMS_WO),

        .get = string_get,                          // get string func to get device name

        .set = string_set,                         // set sring func to change device name

        .notify_check = NULL,                      // not notifiable

        .arg = &(g_user_context.config.dev_name),                  // get/set string pointer (device name)

        .event = NULL,                              // not notifiable

        .hasMeta = false,                          // no max/min/step

        .maxStringLen = MAX_DEVICE_NAME_SIZE,  // max length of device name string
```

```
        },
        [1] = {
          .type = "public.map.property.manufacturer",              // device manufacturer uuid
          .value = &(g_user_context.config.dev_manufacturer),
          .value_len = &(g_user_context.config.dev_manufacturer_len),
          .format = MICO_PROP_TYPE_STRING,
          .perms = MICO_PROP_PERMS_RO,
          .get = string_get,                                  // get string func to get manufacturer
          .set = NULL         ,                               // set sring func to change manufacturer
          .notify_check = NULL,                               // not notifiable
          .arg = &(g_user_context.config.dev_manufacturer),
          .event = NULL,                                      // not notifiable
          .hasMeta = false,
          .maxStringLen = MAX_DEVICE_MANUFACTURER_SIZE,       // max length of device
manufacturer
        },
        [2] = {NULL}                                          // end flag
      }
    },
```

设备描述表上传给 APP 时采用 JSON 格式，使用一个 services 对象数组表示设备的模块列表；每一个 service 对象中使用一个 properties 对象数组表示该模块所具有的所有属性；每一个属性中使用不同的字段表示该属性的特征。

设备描述表结构示例如下：

Figure 12. 设备描述表示例

```
{
 "services": [
   {
     "type": "UUID",                    // service1 UUID
     "iid": <integer>,                  // service iid
     "properties": [
       {  // property 1
         "type": "UUID",                // property UUID
         "iid": <integer>,              // property iid
         "value": <value>,              // property value
         "format": "data_type",          // property value type
         "perms": [                     // property permission
           "pr",                        // can  read
           "pw",                        // can  write
           "ev"                         // will  notify
         ],
         "maxValue": <value>,           // max value for int/float
         "minValue": <value>,           // min value for int/float
         "minStep": <value>,            // min step value for int/float
         "maxStringLen": <integer>      // max string length in byte
         "unit": "unit string"           // property data unit
       },
       {  // property 2
          ...
       }
```

```
        ...
      ]
},
{  // service2
  ...
},
...
  ]
}
```

用户要添加新功能模块，连接好相应的硬件后，只需要在固件的设备描述表中添加相应的 services 和 properties 及相关属性的 get/set/notify_check 操作即可告知 APP 新添加的模块。

设备描述表字段说明如下表：

Table 1. 设备描述表字段说明

字段	作用	值
type	service/property 的唯一标识，表示不同类型的模块或属性	唯一的字符串，如“public.map.service.dev_info”
value	属性的值	固件中为属性值变量的指针，如：
value_len	属性值的长度	固件中为属性值长度变量的指针，如：
format	属性值的数据类型	枚举类型：MICO_PROP_TYPE_INT，MICO_PROP_TYPE_FLOAT，MICO_PROP_TYPE_BOOL，MICO_PROP_TYPE_STRING
perms	属性的访问权限	MICO_PROP_PERMS_RO，MICO_PROP_PERMS_WO，MICO_PROP_PERMS_EV 可组合
get	属性值的读方法，如读取传感器数据	函数指针：int (*get)(struct mico_prop_t *prop, void *arg, void *val, uint32_t *val_len);由用户定义函数功能
set	属性值的写方法，如设置 LED 灯开关。	函数指针：int (*set)(struct mico_prop_t *prop, void *arg, void *val, uint32_t val_len);由用户定义函数功能
notify_check	属性通知条件检测方法，	函数指针：

字段	作用	值
	判断属性变化是否需要上报云端。	int (*notify_check)(struct mico_prop_t *prop, void *arg, void *val, uint32_t *val_len); 由用户定义函数功能
arg	额外的参数，传给 get/set/notify_check 函数	如传递 user_context 的指针给 get/set/notify_check，方便获取 user 的其他参数
event	属性通知开关，根据此变量值确定该属性是否需要主动上报	bool 型变量指针 ,指向属性的通知开关变量，如：&(user_context.config.adc_event)
hasMeta	属性的元数据标志，表示该属性是否具有最大值、最小值、最小间隔的限制	bool 值，true: 具有最大值、最小值、最小间隔；false 则没有这些限制
maxValue	int/float 型属性的最大值	int/float
minValue	int/float 型属性的最小值	int/float
minStep	int/float 型属性的最小间隔	int/float
maxStringLe	字符串的最大长度	长度值
unit	属性值的单位	字符串， 如“%”，“degree”

（2）属性操作

每一个属性都需要用户根据不同的硬件自定义属性的操作方法，即 get/set/notify_check 方法。AppFramework 调用相应的 get/set 方法读取硬件数据或者操作硬件，调用 notify_check 方法检测属性值是否需要上报给云端。其中：

- **get 方法：**

  读取硬件的属性值，AppFramework 会调用该方法获取硬件状态；

  读取成功，返回 0；

  读取失败，返回非零值。

- **set 方法：**

  根据属性值操作相应的硬件，AppFramework 会调用该方法操作硬件；

  操作成功，返回 0；

  操作失败，返回非零值。

- **notify_check：**

  如果该模块的某一个属性是具有主动通知特性的(perms 字段包含 ev)，则需要定期检查是否需要主动上报该属性值；AppFramework 会调用该方法判断是否满足主动上传属性值的条件。

满足通知条件（例如：属性值发送了变化），返回 1；

否则，返回 0。

6. 设备网络配置

使用手机 APP 对 MiCOKit 进行 Wi-Fi 网络配置以及云端激活、绑定方法请参考文档：《MiCOKit 网络配置协议》。

7. 云端数据通信

MiCOKit 和手机 APP 通过 FogCloud 的数据通信协议参考文档：《MiCOKit 云数据通信协议》

8. 版本更新说明

日期	修改人	版本	更新内容
2015-7-29	Jenny Liu	V1.0	1. 初始版本
2015-9-8	Jenny Liu	V1.1	1. 格式修订