ESP量产平台固件烧写方案

# 方案背景及需求

为了满足我司 ESP8285/ ESP32 内嵌flash IC ，SIP 芯片 等 固件烧写需求，在此提出如下固件烧写方案：

**面向客户：**

1. 采用ESP8285/ SIP 芯片，ESP32 内嵌flash 方案的客户直接烧写IC 或直接线上烧写模组；

烧写原理及步骤简介（详见流程简图）：

1. Host MCU通过UART download 一个带SPI driver 的bin文件到ESP IC Ram 中（ESP\_cmm\_download\_driver.bin）；
2. ESP\_cmm\_download\_driver.bin 运行，通过SPI 与主机台MCU 进行交互，Host从SD中读取固件bin发送到目标模块，完成固件的烧写。

**优势分析：**

原始的UART 下载方式，速度慢，稳定性差，

当前设计采用SPI 进行固件传输，效率高，下载速度快；

**量产可操作性分析：**

将如下设计方案，会做成一整套的下载解决方案，我司提供硬件所有相关环境，客户在工厂只需要采用我司提供的烧写机台，即可完成固件烧写。

1. 烧写机台

采用ESP-WEOVER-KIT开发板，带一块显示屏，可以清晰的显示出下载的状态，有助于工厂操作人员清晰的判断下载成功与否；

1. 烧写配置存放SD 中，产线部署简单，稳定

烧写所有配置文件与烧写的bin 文件，均放置在SD 卡中，烧写机台从SD 卡中读取烧写文件进行烧写，减少产线配置机台的风险。

就如下方案，请各位在此邮件讨论并提出意见，我们一起把这个解决方案做好。

# 方案设计

本章会介绍本方案的总体设计，分成两大部分，先介绍的是方案的流程设计，包括主函数设计和各个功能模块的设计。第二部分会介绍所用的两个主要通信协议的设计——uart和spi。

## 方案设计及流程图

本节将首先介绍方案总体的流程设计，给出功能模块划分、主函数流程图等进行说明，然后按照具体功能依次介绍各个功能的流程设计，并设计一些输入与输出。在方案实施环节，将会按照设计的各个功能编写对应的几个函数。

### 方案总体设计



图 2‑1 方案程序模块及流程

如图 2‑1所示，平台程序功能主要分成4部分：模块初始化、烧录状态复位、烧录工作执行和对烧录结果的操作。

每个部分都会分成几个步骤来实现：

* 模块初始化：
  + 主要是硬件及外设驱动的初始化操作，主要的硬件外设有：控制用的gpio、uart、spi-master、lcd、sd卡。
  + 屏幕显示初始化
* 烧录状态复位
  + 通过gpio电平来判断是否有模块接入
  + 需要读取sd卡中的烧录配置文件，来初始化烧录参数
* 烧录工作执行步骤（spi）
  + 与目标模块所带的rom程序建立uart通信连接
  + 通过uart口将一个带spi驱动程序bin文件发送到目标模块的ram
  + 与目标模块建立spi通信连接（本机为master）
  + 从SD中读取要发送的flash固件，并使用spi将固件发送到目标模块的flash中。
* 烧录结果记录
  + 通过gpio空控制目标模块重启，运行flash程序，然后通过uart通信，来判断固件烧录是否成功
  + 将结果记录到SD卡中，并显示到LCD屏上。
* 结果同步到服务器：将每次操作的结果发生到服务器，但目前不是必须要实现的功能，会另外开启一个独立的task，等待单次操作完成信号再执行上次操作。
  + 连接到AP
  + 连接到服务器
  + 将结果上传到服务器



图 2‑2 程序主函数流程图

图 2‑2是方案程序所设计的主函数的执行流程图，根据所分的模块和功能大致预分成5个主要流程函数来调用，并通过函数的返回结果来判断和设置程序状态机，从而完成固件烧写方案。

另外操作过程中的各个状态也会显示到LCD屏上。下面做一下具体的划分描述：

### 硬件驱动初始化



图 2‑3 硬件初始化流程

### 烧录状态复位



图 2‑4 烧录复位流程

这里设计用一个gpio口的高低电平来判断是否有目标模块接入，在模块接入前程序会block在这里。

对于烧录的初始化，需要读取SD卡中的烧录参数，判断参数是否符合当前烧录版本，并根据参数执行之后的烧录操作。这样就能实现，用户通过修改SD卡中的配置文件，自动识别烧录不同的固件。

### 固件烧录操作

目前烧录固件的操作可以先使用uart来进行，因为uart在板子上可以实现较高的速率，等完成之后，再使用spi比较两者的效率，移植也比较简单。

如果使用spi烧录固件，则操作分成两大步骤：



图 2‑5 uart通信操作

1. 首先要通过uart连接将spi驱动固件发送到目标模块的RAM，此部分的操作流程主要参考之前我们flash下载工具操作的流程。



图 2‑6 spi通信操作

1. 在发送spi驱动成功后，就要通过建立spi连接，然后通过spi将flash固件发送到目标模块的flash；此过程的流程主要参照uart的连接操作流程，希望尽量做到一直，只是调用的发送接口不同。

### 结果记录



图 2‑7 烧录结果操作

## 通信协议设计

本节主要介绍所使用的通信接口的协议，为了通用性和可移植性行考虑，尽量会按照flash下载工具所使用的进行设计，包括uart和spi两部分。最终目标是使两部分的协议的区别只是在于发送的调用接口不同。

传输协议采用SLIP（serial Line Internet Protocol）的封装格式。

* 每个数据包都以0x0C开始和结束
* 如果0x0C出现在数据包内，就将0x0C替换成两个字节0xDB 0xDC；如果出现0xDB则替换为0xDB 0xDD。
* 在数据帧里，数据包有数据头和长度不定的数据体组成，如图 2‑8所示。
* 所有多字节字段的存储模式均为小端模式。

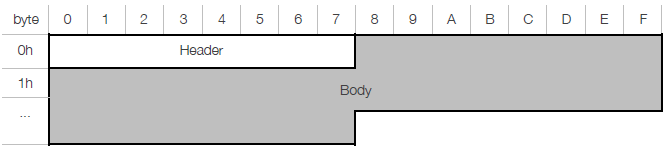


图 2‑8 数据包格式

### 数据头

表 2‑1 数据头格式

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 字节 | 请求 | 应答 |
| **Type** | 0 | 始终为0x00 | 始终为0x01 |
| Command | 1 | 操作代码详细信息请参考表格 | |
| Data size | 2~3 | 数据体的大小 | |
| Checksum/Response | 4~7 | Payload(数据体中16字节之后的固件数据)的异或校验。 | 响应数据 |
| Body | 8~n | 取决于操作 | |
| Status | 8 | - | 状态标识，成功（0）或失败（1） |
| Error | 9 | - | 成功（null）或失败（错误码） |

表 2‑2 操作代码

|  |  |  |
| --- | --- | --- |
| 代码 | 名称 | 说明 |
| 02 | Flash Download Start | 擦除Flash中的数据。   * **Word0**：擦除扇区的数量，每个扇区4096字节 * **Word1**：发送数据包的数量 * **Word2**：发送数据包的大小，如0x400 * **Word3**：偏移地址 |
| 03 | File Packet Send | 发送数据   * **Word0**：发送数据包的大小（填0x400） * **Word**1：发送数据包的序列号 * **Word2：0x0** * **Word3：0x0** |
| 04 | Flash Download stop | 停止发送数据 |
| 08 | Sync Frame Send | Sync\_framep[36] = {0x07, 0x07, 0x12, 0x20, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55}; |

### 数据体

数据体格式如图 2‑9所示。

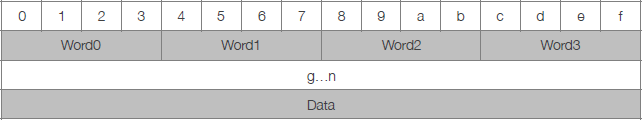


图 2‑9 数据体格式

数据体的前16个字节（word0~word3）是描述数据体的。不同的数据指令，其描述也不同。

需要设置的参数有：下载模式（ram、flash）、芯片类型、spi配置参数

1. 从SD卡中读取image下载模式（ram、flash）、芯片类型、需要执行的操作等信息
2. 根据下载模式设置参数
   * RAM：读取image的前8字节
     + 固件是否有效（magic，sequence>16?）
     + 有几个sequence
   * 读取SD中的配置文件
     + Flash的spi参数等
3. 根据芯片类型读取固件
   * Esp32：
     + 继续读取16字节额外头部信息，并进行验证
     + 依次读取各个segment
     + 读取checksum并进行验证image有效性
   * ESP8266：需要判断magic进行对应操作
     + 0xe9：读取正常固件
       - 依次读取各个segment
     + 0xea：读取ota固件：需要处理一个额外的irom image。。。
4. 根据参数segments数发送固件
   * 对于每个segment：
     + 读取offset、size和data
     + 调用开始load命令（begin）
     + 分片发送sequence的各个数据段，需要一个seq
     + 调用完成load命令（finish）

后续工作计划：

1. 完成SD卡读写操作、文件读写操作 （√）
2. SD卡操作移植到工程里面 （）
3. 完成从SD卡读取配置文件 （）
4. 完成image->ram的下载 （）
5. 完成image->flash （）