ESP8266EX

常见问题

版本 1.7

关于本手册

本文介绍 ESP8266EX 的常见问题以及解答。

发布说明

日期	版本	发布说明
2016.08	V1.0	首次发布。
2016.09	V1.1	第 3 章增加问题“如何优化 ESP8266 应用的内存使用？”。
2016.10	V1.2	• 第 1 章增加问题“如果应用使用的是不支持 FOTA 的固件，要将 eagle.irom0.text.bin 下载到哪里？”。 • 第 3 章增加问题“如何让 ESP8266 上电后快速连接 AP？”。
2016.10	V1.3	第 7 章增加问题“调用 wifi_softap_set_config() 时，函数返回成功，但为何无法修改 ESP8266 的 SoftAP SSID 和密码？”。
2016.11	V1.4	第 3 章增加以下两个问题： • 为什么 ESP8266 进入启动模式（2，7）并触发看门狗复位? • ESP8266 上电时打印的 boot 模式信息代表什么？如何改变 boot 模式?
2018.08	V1.5	格式更改。
2018.10	V1.6	增加第 7 章有关“如何修改默认上电校准方式？”的问题。
2019.03	V1.7	第 3 章增加有关 “ESP8266 启动打印 ***ets_main.c***” 的问题

文档变更通知

目录

1. 应用

1.1. 为什么云端升级需要 2 个 BIN 文件？“user1.bin”和“user2.bin”有什么区别？

user1.bin 和 ***user2.bin*** 是 2 个不同的 BIN 文件。生成 ***user1.bin*** 和 ***user2.bin*** 时，必须使用相同的 Flash 和 boot 设置，以保证 OTA 升级成功。2 个 BIN 文件是互补的，运行 ***user1.bin*** 的时候，升级是下载 ***user2.bin***；运行 ***user2.bin*** 的时候，升级是下载 ***user1.bin***。这样可以保证升级过程中，如果有掉线的情况发生，设备还是可以正常运行。

1.2. 如何生成“user1.bin”和“user2.bin”？

编译环境下，执行 ***gen_misc.sh*** 分别得到 ***user1.bin*** 和 ***user2.bin***。步骤如下：

1. 使用正确的 Flash 和 boot 配置，编译生成 ***user1.bin***。
2. 执行 make clean，以便清除之前的残余信息。
3. 使用相同的 Flash 和 boot 配置，编译生成 ***user2.bin***。

1.3. 如果应用使用的是不支持 FOTA 的固件，要将 eagle.irom0.text.bin 下载到哪里？

对于 Non-OS SDK 和 RTOS SDK，固件 BIN 文件的位置取决于合适的链接脚本内容。如果应用中使用的是不支持 FOTA 的固件，则用户代码包含在 ***eagle.irom0.text.bin*** 中。该 BIN 文件在 Flash 中的位置是由 ***SDK/ld*** 中的链接脚本 ***eagle.app.v6.ld*** 决定的。

> ⚠ **注意：**
>
> 在所有版本 *ESP8266 SDK* 中，*eagle.irom0.text.bin* 的默认位置并不都是一样的。用户可以确认在自己的 *ESP8266 SDK* 版本中，该 *BIN* 文件在 *Flash* 中的位置。如下图所示：

```
MEMORY
{
  dport0_0_seg :                    org = 0x3FF00000, len = 0x10
  dram0_0_seg :                     org = 0x3FFE8000, len = 0x14000
  iram1_0_seg :                     org = 0x40100000, len = 0x8000
  irom0_0_seg :                     org = 0x40210000, len = 0x5C000
}
```

图中标蓝的数字表示 ***eagle.irom0.text.bin*** 在 Flash 中的地址。在上图的例子中，该地址为 0x10000。

1.4. 云端升级失败有哪些原因?

云端升级的详细介绍参考文档《*ESP8266 FOTA 云端升级指南* 》。

请先检查以下问题：

- 确认使用了正确大小的 Flash。
- 确认是否烧录了 ***blank.bin*** 做初始化。
- 确认 ***user1.bin*** 和 ***user2.bin*** 下载到了正确的地址。
- 确认生成 ***user1.bin*** 和 ***user2.bin*** 使用了相同的 Flash、boot 配置。

1.5. 如何通过我自己的服务器进行云端升级?

如果通过客户自己的服务器升级，请确认服务器满足下面的要求。

1. 发送 HEAD 指令到云端服务器，询问待升级的 BIN 文件长度，服务器回复的 HTTP 包头中要求带有 BIN 文件的长度信息。
2. 根据上述方法查询到的 BIN 文件长度，在 ESP8266 模块的 Flash 待升级区域，擦除该指定长度 （`spi_flash_erase_sector`），等待下载。
3. 发送 GET 指令，从服务器下载 BIN 文件，写入到 Flash 的待升级区域。

1.6. 如何使用我自己的云服务器进行云端升级?

客户可以使用自己的云服务器，但是要能支持 HTTP 请求，并可以支持设备控制功能。

云端的 API，请见：*http://iot.espressif.cn/#/api-zh-cn/*。

1.7. ESP8266 如何和云端服务器进行交互?

可以使用标准 HTTP 协议连接云服务器。比如，ESP8266 使用如下的 HTTP 请求。

```
GET /your-bin-file-path.bin HTTP/1.1
Host: yourdomain.com
Connection: keep-alive
Cache-Control: max-age=0
Accept:
text/html, application/xhtml+xml, application/xml; q=0.9, image/webp, */*;
q=0.8
User-Agent:
```

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/39.0.2171.95 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language:
en-US, en;q=0.8, ja; q=0.6,zh-CN; q=0.4,zh-TW; q=0.2
```

⚠ **注意：**

HTTP 协议规定请求以"\r\n\r\n"为结束，所以在通讯过程中请不要使用这样的组合。

客户可以通过下面的方法来 debug 服务器：

```
telnet <sitename> <port_number>
```

请事先确认 telnet 在您的 PC 上是可以使用的，或者使用其他的支持 telnet 的终端软件也可以达到同样的目的，比如 TeraTerm。

1.8. SmartConfig 配网配不上有哪些原因?

请做以下检查：

- APP 版本是否支持 SDK 版本或 SmartConfig 版本。
- 手机连接的路由器不能是单 5G 路由（双频路由器除外）。
- SmartConfig 过程中不要调用其他 API。
- 使用 AT 时，设备没有获得 IP 之前，不要调用 smartconfig_stop。

如以上排除，请把连接失败和成功的 log 发给我们技术做支持分析。

1.9. SmartConfig 支持的 APP 对应的版本是什么?

调用 `smartconfig_start()` 接口，会有"SC version: vX.X.X"版本信息打印。这是 ESP-TOUCH 模块的版本号。下面是非 OS 对应的 SmartConfig 版本和 APP 版本。

SDK 版本	SmartConfig 版本	APP 版本
sdk v1.2.0	smartconfig v2.4	app v0.3.4.x
sdk v1.3.0	smartconfig v2.5	app v0.3.4.x
sdk v1.3.0	smartconfig v2.5.1	app v0.3.4.x
sdk v1.4.0	smartconfig v2.5.2	app v0.3.4.x
sdk v1.5.0	smartconfig v2.5.3	app v0.3.4.x
sdk v1.5.4	smartconfig v2.5.4	app v0.3.4.x

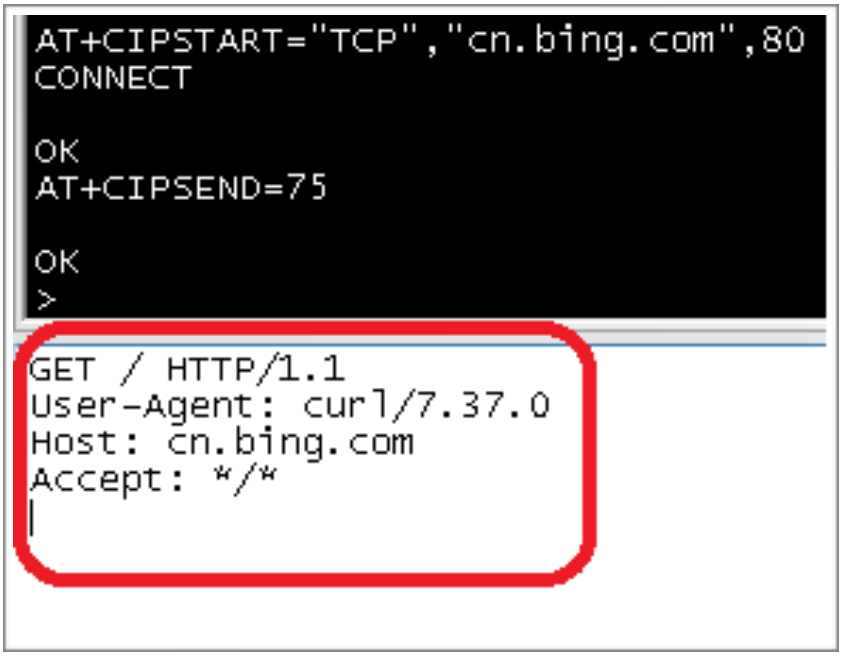
1.10. ESP8266 支持 HTTP 服务端吗?

支持。ESP8266 在 SoftAP 和 Station 模式下都可以作服务端。

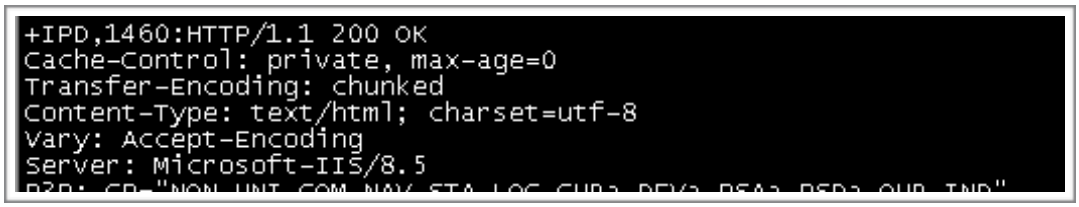
- 在 SoftAP 模式下，ESP8266 的服务端 IP 地址是 192.168.4.1。
- 如果 Station 模式，服务端的 IP 地址为路由器分配给 ESP8266 的 IP。
- 如果是基于 SDK 二次开发，那么需使用 espconn 结构体和相关 API。
- 如果是使用 AT 指令，需使用 `AT+CIPSERVER` 开启服务端。

1.11. 如何通过 AT 指令发 HTTP 包?

1. AT 指令配置 SoftAP + Station 模式：`AT+CWMODE=3 // set softAP+station mode`
2. AT 指令连接路由：`AT+CWJAP="SSID","password" // ESP8266 station connect to router`
3. 创建 TCP 连接，按照 HTTP 包的格式发送数据，如下图红框标注，请注意，HTTP 包中的换行符 [(0x0d 0x0a) or (CR LF)] 是必须的，不能省去。



4. 收到 HTTP 包的回复。



1.12. 如何在 AT+ 指令中定义自己的函数？如何在函数之间传递参数？

在 Non-OS SDK 中的 AT 示例（*\ESP8266_NONOS_SDK\examples\at\user\user_main.c*）中有提供如何实现一条自定义的 AT 指令“AT+TEST”。

结构体 at_funcationType 用于定义一条指令的四种类型，例如指令名称“AT+TEST”。

- 类型 at_testCmd：测试指令，对应指令为 AT+TEST=？，AT 示例中注册的实现回调为 at_testCmdTest，测试指令可以设计为返回参数的取值范围；注册为 NULL，则无测试指令。
- 类型 at_queryCmd：查询指令，对应指令为 AT+TEST？，AT 示例中注册的实现回调为 at_queryCmdTest，查询指令可以设计为返回当前值；注册为 NULL，则无查询指令。
- 类型 at_setupCmd：设置指令，对应指令格式为 AT+TEST=parameter1,parameter2,.....，AT 示例中注册的实现回调为 at_setupCmdTest，设置指令可以设计用于设置参数值；注册为 NULL，则无设置指令。
- 类型 at_exeCmd：执行指令，对应指令为 AT+TEST，AT 示例中注册的实现回调为 at_exeCmdTest，执行指令可以设计用于执行某项操作；注册为 NULL，则无执行指令。

1.13. 微信中的近场发现中使用的 Product ID 是从哪里来的？

需要建立一个 device ID，比如在客户的微信号下，建立了一个 ID = 1234 的设备。

通过如下命令

```
curl -q “https://api.weixin.qq.com/cgi-bin/token?
grant_type=client_credential&appid=wx0f74f861897f84b5&secret=your_WeChatnam
e_32characters_please”
```

//注：目的是得到 access token。（“your_WeChatname_32characters_please”代表客户的微信 API 应用秘钥，申请获得，应该为 32 个字符。）

返回：

```
{"access_token":"L2_2V1E98Vk-
jTXenXDZjDT0GaudUn_VGTRa7O98hdfT0lTZa2B7nj6YvXN01gssQa3ZraRgjALuCvxd-
SamuPR885KJabaw1EYLA0kns-Yglr4ryolEhHb-QcnWMaNqSEDjACANZY","expires_in":
7200}
```

```
curl -d '{"device_num":"1","device_list":
[{"id":"1234","mac":"18fe3497d500","connect_protocol":"4","auth_key":"00000
000000000000000000000000000","close_strategy":"1","conn_strategy":"1","cryp
t_method":"0","auth_ver":"0","manu_mac_pos":"-2","ser_mac_pos":"-2"}],"op_t
ype":"0", "product_id": 5157}' "https://api.weixin.qq.com/device/
```

```
authorize_device?access_token=L2_2V1E98Vk-
jTXenXDZjDT0GaudUn_VGTRa7O98hdfT0lTZa2B7nj6YvXN01gssQa3ZraRgjALuCvxd-
SamuPR885KJabaw1EYLA0kns-Yglr4ryolEhHb-QcnWMaNqSEDjACANZY"
```

//注：建立一个 ID = 1234 的设备。

这样，您只需要使用 AT 指令 `AT+CWSTARTDISCOVER=“gh_9e2cff3dfa51","1234",1` 即可（“`gh_9e2cff3dfa51`”是您的微信公众号名字）。

1.14. ESP8266 如何添加自定义 AT 命令，自定义 AT 命令字段和参数段长度限制是多少?

客户可以基于 ***ESP8266_NONOS_SDK\examples\at*** 示例代码，在 ESP8266 自带 AT 命令的基础上，添加客户自定义的 AT 命令。

关于自定义 AT 命令，SDK 限制整条 AT 命令数据长度最大 128 字节（含结束符“\r\n”），不单独限制命令段和参数段。

例如：`AT+CMDTEST=param1,param2,param3,….paramN\r\n`

则：`strlen(“AT+CMDTEST=param1,param2,param3,....paramN\r\n”)<=128 bytes`

相关 SDK 及参考资料请至乐鑫官网下载：*ESP8266 SDK 和 Demo*。

2. 云平台

2.1. 在乐鑫的云平台上，设备的生命周期是怎样的?

1. 烧录 master-device-key，出厂。
2. 到达终端用户，使用 ***Airkiss/ESP_TOUCH*** 让设备联网，同时传递随机字符（token，App 产生）作为权限标识，设备上网之后调用 ***/v1/device/activate***，同时把 token 传递给云端。
3. 终端用户使用 App 并且调用 ***/v1/device/authorize*** 接口（使用之前产生的随机 token），获得这个设备的所有权（成为 owner，获得对应的 owner key）。
4. 终端用户对设备的拥有本质上是对 device key 的拥有，对于每一个设备的控制，是通过对应的 device key 来操作的。
5. 终端用户使用 ***/v1/user/devices*** 列出拥有的设备以及对应的 device key，然后使用对应的 device key 做具体的操作。
6. 终端用户是 owner 的权限下，可以调用 ***/v1/device/share*** 接口分享设备给他人，对应的用户使用 ***/v1/device/authorize*** 接口得到授权。

3. 系统

3.1. ESP8266 的看门狗是什么作用?

为了提供系统稳定性，以应对多冲突的操作环境，ESP8266 集成了 2 级看门狗机制，包括软件看门狗和硬件看门狗。默认 2 个看门狗都是打开的。

3.2. 看门狗的超时间隔是多少? 触发超时事件会有什么现象?

硬件看门狗中断时间为 0.8*2048 ms，即 1638.4 ms，中断后处理时间为 0.8*8192 ms，即 6553.6 ms。其中中断处理后时间为硬件看门狗中断发生后，需要进行喂狗操作的时间，如果超过该时间，即会触发硬件看门狗复位。因此，在仅有硬件看门狗的情况下，一个程序段如果运行时间超过 6553.6 ms，即有可能触发硬件看门狗复位，若超过 8192 ms 则一定会触发复位。

软件看门狗建立在 MAC timer 以及系统调度之上，中断时间为 1600 ms，中断后处理时间为 1600 ms。因此，在有软件+硬件看门狗的情况下，一个程序段如果运行时间超过 1600 ms，即有可能会触发软件看门狗复位，若超过 3200 ms 则一定会触发复位。如果我的应用不需要看门狗，如何关闭看门狗?

3.3. 如果我的应用不需要看门狗，如何关闭看门狗?

当前 SDK 仅支持关闭软件看门狗，支持同时喂软硬件看门狗。可以通过如下方式防止执行时间过长的用户程序导致看门狗复位：

1. 如果一个程序段运行时间在触发软件看门狗和触发硬件看门狗复位之间，则可通过 `system_soft_wdt_stop ()` 的方式关闭软件看门狗，在程序段执行完毕后用 `system_soft_wdt_restart ()` 重新打开软件看门狗。
2. 可以通过在程序段中添加 `system_soft_wdt_feed ()` 来进行喂软硬件狗操作，防止软硬件看门狗复位。

3.4. 如果我要在程序里面引入 10 秒的延迟，怎么做最好?

看门狗不支持无限循环。如果客户使用循环做延迟或者进入一个事件太长时间，就会触发硬件看门狗重启。推荐使用 callback 和 timer 的 API 做延迟。

如果要轮询事件，推荐使用中断和 timer 的 API 来做。大多数事件都是关联到 callback 上的，所以大多数情况下，轮询都是可以避免的。

3.5. 对于 Non-OS SDK，memory leak 问题如何 debug?

可通过定义 `MEMLEAK_DEBUG` 宏启用 memory leak debug 功能，代码中调用 `os_malloc`，`os_zalloc`，`os_calloc`，`os_realloc`，`os_free` 可将调用的文件以及调用的对应行数记录在内存管理链表中，在有需要的地方通过调用 `system_print_meminfo()` 可打印出 heap 区内存分配情况。步骤：

1. 修改用户工程目录的 MakeFile，在 `CONFIGURATION_DEFINES` 后加宏定义：

   `-DMEMLEAK_DEBUG`

   如：`CONFIGURATION_DEFINES = -DMEMLEAK_DEBUG`

2. 在用户代码，如 ***user_main.c*** 中，增加如下代码：

```
#include “mem.h"
bool ICACHE_FLASH_ATTR check_memleak_debug_enable(void)
{
return MEMLEAK_DEBUG_ENABLE;
}
```

3. 在有可能内存泄露的地方调用 `system_print_meminfo()`，建议仅在关键代码位置加入此函数进行 debug。

3.6. 对于 RTOS SDK，memory leak 问题如何 debug?

暂不支持该功能。可以参考 FreeRTOS 官网的 debug 方法。

3.7. 如何优化 ESP8266 应用的内存使用?

通过以下四种方法可以优化 ESP8266 应用的内存使用，减少应用的内存占用空间。

1. 将字符串放到 Flash 中：

- 有些字符串可以放在 Flash 中，特别是长字符串，例如 HTML 请求和响应模板。

  比如，一个字符串原来是用 define 定义的：

```
#define test_string    "hello world"
```

  现在可以定义成如下：

```
static const char test_string[] ICACHE_RODATA_ATTR = "hello world";
```

- 当用 `ICACHE_RODATA_ATTR` 定义字符串常量时，需要对数据内容进行四字节对齐读取。由于 Flash 中的数据需要四字节对其读取，所以定义一个宏获取对齐后的字符串长度：

```
#define GET_ALIGN_STRING_LEN(str)    ((strlen(str) + 3) & ~3)
```

使用字符串时，动态分配一个新的数组对象，读写 Flash 中的数据。然后用 `os_memcpy` API 来复制数据内容：

```
unsigned int str_len = GET_ALIGN_STRING_LEN(test_string);
char *tmp_string = (char *)os_malloc(str_len);
os_memcpy(tmp_string, test_string, str_len);
```

- 在用户的应用代码里使用 `tmp_string` 进行操作，而不使用 `test_string`。此方法除了减少应用的 RAM 占用空间，也能解决由于对 Flash 中的数据进行非对齐读取时，在应用中引起的 exception。

  .....

- 当用户代码中，无需再使用通过以上方法读取的数据，需要释放之前分配的内存空间

  ```
  os_free(tmp_string);
  ```

> ⚠ **注意：**
>
> 如果不释放之前分配的内存空间，重复分配内存将会减少核心功能所需的内存，导致 *API* 出现功能异常或失败。

2. 把 const 数据放到 Flash：

- uint32 类型的数组可以直接放到 Flash，比如：

  ```
  const uint32 array[4] ICACHE_RODATA_ATTR = {0x11111111, 0x22222222,
  0x33333333, 0x44444444};
  ```

  可以直接使用 array[0]。

- 对于 uint8 和 uint16 类型的数组，要注意读取数据的时候要四字节对齐，比如：

  ```
  const uint8 array[7] ICACHE_RODATA_ATTR = {0x01, 0x02, 0x03, 0x04, 0x05,
  0x06, 0x07};
  ```

- 如果需要按字节读取 char 数组当中的元素，可从软件上进行处理，先按四字节读取，然后再按偏移取当中的一个字节。如果直接读取 array[0]，会导致 crash。

- 对于数据结构，通常做法是分配比用户需要读取的结构更大的内存，从 Flash 四字节读取数据到内存。如同方法 1，在代码中依然使用对象指针。修改代码操作内存中的数据结构，而不是读取数组。

3. 将调试字符串放到 Flash 中：

现在默认的 `printf` 打印的字符串都还是放在 RAM 区，占用部分内存。如果用户无需频繁打印日志文件，或者调试字符串太长，可以使用优化的 `os_printf` 把打印的字符串放到 Flash 而不是 RAM 中。

4. 避免使用全局数组变量：

全局数组变量会在应用的整个生存期中占用不必要的内存。为减少全局数组变量的使用，乐鑫提供了动态内存分配 API。在基于事件的编程中，请使用 `os_malloc` 和 `os_free` 来动态分配所需的内存空间。但注意，我们不建议过于频繁地分配和释放大小不等的内存空间。

3.8. 发生“fatal exception”问题如何处理?

可以在（*.S）文件中找出对应的地址，添加打印以便定位问题。

```
Fatal exception (28):
epc1=0x4025bfa6, epc2=0x00000000, epc3=0x00000000, excvaddr=0x0000000f,
depc=0x00000000
```

比如使用的是 ***user1.bin***，那么就在 ***user1.S*** 中找到 0x4025bfa6 地址，并查明对应的函数。

如果使用的是 ***flash.bin*** 和 ***irom0text.bin***，可以在 ***eagle.S*** 中查找出错的地址。

3.9. ESP8266 总共有几个 timer?

ESP8266 有 2 个 timer。一个硬件的 timer，一个软件的 timer。

API `os_timer` 是 DSR 处理，不能产生中断，但是可以产生任务。任务会按照普通等级排队。

硬件 timer 能产生中断和任务，中断能触发任务，任务按照普通等级排队。

3.10. 使用 timer 中断是否有特定条件?

请参考 SDK 的 API 参考：《*ESP8266 Non-OS SDK API 参考*》和《*ESP8266 RTOS SDK API 参考*》。一般情况，使用 Non-OS SDK 时，硬件中断回调里面不要有声明为 `ICACHE_FLASH_ATTR` 的功能。同时中断回调里不要占用 CPU 太长时间。

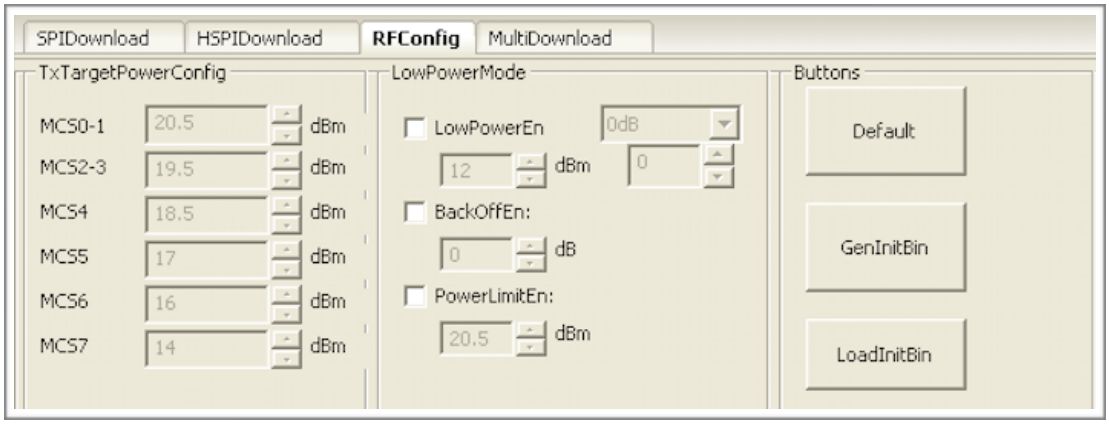
3.11. 如何调整 Tx Power?

“`system_phy_set_max_tpw`” 用于设置 RF Tx Power 最大值，单位：0.25 dBm。

目前 Flash download tool 中已开放给客户自行配置并生成 ***esp_init_data_default.bin***。其中关于 Tx power 的调整如下所示：

- ***LowPowerEn***：同时设置每个模式下的 Tx Power。
- ***BackOffEn***：同时设置每个模式下 TX Power 需要减小的值。
- ***PowerLimitEn***：限制 Tx Power 的最大值。

- 设置确认之后点击 ***GenInitBin***，并替换原先的 ***esp_init_data_default.bin***。



3.12. 为什么 ESP8266_Non-OS_SDK 中有的函数前面添加了“ICACHE_FLASH_ATTR”宏?

对于 ESP8266_Non-OS_SDK：

添加了“ICACHE_FLASH_ATTR”宏的函数，将存放在 IROM 中，CPU 仅在调用到它们的时候，将它们读到 cache 中运行；没有添加“ICACHE_FLASH_ATTR”宏的函数，将在一开始上电运行时，就加载到 IRAM 中运行；由于空间有限，我们无法将所有代码都一次性加载到 IRAM 中运行，因此在大部分函数前添加“ICACHE_FLASH_ATTR”宏，放在 IROM 中。

请注意，不要在中断处理函数中调用带有“ICACHE_FLASH_ATTR”宏的函数，否则可能与 Flash 读写操作冲突。

对于 ESP8266_RTOS_SDK：

函数默认存放在 IROM 中，无需再添加“ICACHE_FLASH_ATTR”宏。中断处理函数也可以定义在 IROM 中。如果开发者需要将一些频繁调用的函数定义在 IRAM 中，在函数前添加“IRAM_ATTR”宏即可。

3.13. 为什么编译 Non-OS SDK 时会发生 IRAM_ATTR 错误?

如果需要在 IRAM 中执行功能，就不需要加“ICACHE_FLASH_ATTR”的宏，那么该功能就是放在 IRAM 中执行。

3.14. 为什么编译的时候会发生“irom0_0_seg”错误?

它表示代码量太大，IROM 区域存放不下了。

我们可以在 SDK_v0.9.5（及之后）的软件版本中，尝试如下步骤，解决这个问题:

1. 使用默认设置，编译生成 ***eagle.flash.bin*** 和 ***eagle.irom0text.bin***。

   (1) 如果 `size of eagle.flash.bin + size of eagle.irom0text.bin >= 236KBytes`：很抱歉，您的代码量太大了，只能换大些的 Flash。

   (2) 如果 `size of eagle.flash.bin + size of eagle.irom0text.bin < 236KBytes`：

   请继续步骤 2。

2. 在路径 ***SDK/ld*** 下修改文件***"eagle.app.v6.new.512.app1.ld"***。

```
irom0_0_seg :                         org = 0x40201010, len = 0x2B000
```

根据步骤 1 中编译的***"eagle.irom0text.bin"***大小，改写上述 len 的值。

示例：如果***"eagle.irom0text.bin"***大小为 179 KB，则可修改配置如下：

```
irom0_0_seg :                         org = 0x40201010, len = 0x2D000
```

3. 重新编译 ***user1.bin*** 选择 boot_v1.2+。

补充说明：

代码中，

- 函数前未加 `ICACHE_FLASH_ATTR` 的，编译到 IRAM 中，最大 32 KB；
- 函数前加了 `ICACHE_FLASH_ATTR` 的，编译到 IROM 中；

因为 RAM的空间有限，因此做了这两个部分的区分：

- IRAM 中的代码，会在上电初始就完整加载到 RAM 中；
- IROM 中的代码是用到的时候才从Flash 加载到 cache 中执行。

3.15. ESP8266 有 main 吗?

ESP8266 没有 main，程序入口为 `user_init`。

3.16. 操作指针有什么需要注意的?

内存必须 4 字节对齐读取，指针做转换时请确保为 4 字节对齐，否则转换失败，不能正常使用。例如，请勿直接指针转换 `float temp = *((float*)data)`；而是使用 `os_memcpy (memcpy)` 实现。

3.17. RTOS SDK 和 Non-OS SDK 有何区别?

主要差异点如下：

Non-OS SDK

Non-OS SDK 主要使用定时器和回调函数的方式实现各个功能事件的嵌套，达到特定条件下触发特定功能函数的目的。Non-OS SDK 使用 espconn 接口实现网络操作，用户需要按照 espconn 接口的使用规则进行软件开发。

RTOS SDK

1. RTOS 版本 SDK 使用 freeRTOS 系统，引入 OS 多任务处理的机制，用户可以使用 freeRTOS 的标准接口实现资源管理、循环操作、任务内延时、任务间信息传递和同步等面向任务流程的设计方式。具体接口使用方法参考 freeRTOS 官方网站的使用说明或者 *USING THE FREERTOS REAL TIME KERNEL - A Practical Guide* 这本书中的介绍。
2. RTOS 版本 SDK 的网络操作接口是标准 lwIP API，同时提供了 BSD Socket API 接口的封装实现，用户可以直接按照 socket API 的使用方式来开发软件应用，也可以直接编译运行其他平台的标准 Socket 应用，有效降低平台切换的学习成本。
3. RTOS 版本 SDK 引入了 cJSON 库，使用该库函数可以更加方便的实现对 JSON 数据包的解析。
4. RTOS 版本兼容 Non-OS SDK 中的 Wi-Fi 接口、SmartConfig 接口、Sniffer 相关接口、系统接口、定时器接口、FOTA 接口和外围驱动接口，不支持 AT 实现。

3.18. 哪些接口需要在 user_init 中调用，否则容易出现问题，或者不生效?

1. `wifi_set_ip_info`、`wifi_set_macaddr` 仅在 `user_init` 中调用生效，其他地方调用不生效。
2. `system_timer_reinit` 建议在 `user_init` 中调用，否则调用后，需要重新 arm 所有 timer。
3. `wifi_station_set_config` 如果在 `user_init` 中调用，底层会自动连接对应路由，不需要再调用 `wifi_station_connect` 来进行连接。否则，需要调用 `wifi_station_connect` 进行连接。
4. `wifi_station_set_auto_connect` 设置上电启动时是否自动连接已记录的路由；例如，关闭自动连接功能，如果在 `user_init` 中调用，则当前这次上电就不会自动连接路由，如果在其他位置调用，则下次上电启动不会自动连接路由。

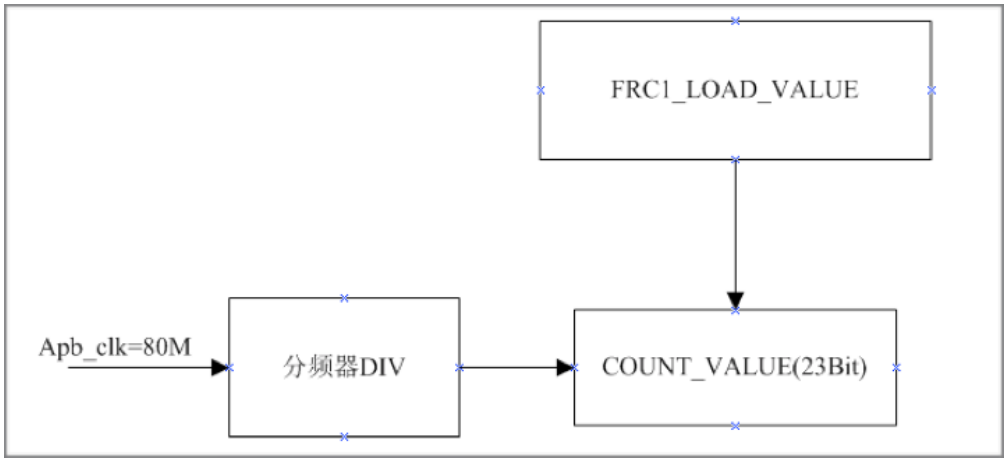
3.19. Light-sleep 如何通过 GPIO 或网络事件唤醒?

在 Light-sleep 模式下，CPU 在暂停状态下不会响应来自外围硬件接口的信号与中断，因此需要配置通过外部 GPIO 信号将 ESP8266 唤醒，唤醒过程小于 3 ms。

```
wifi_station_disconnect();
wifi_set_opmode(NULL_MODE); // set WiFi mode to null mode
wifi_fpm_set_sleep_type(LIGHT_SLEEP_T);
wifi_fpm_open();
PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTCK_U, FUNC_GPIO13);
gpio_pin_wakeup_enable(13, GPIO_PIN_INTR_LOLEVEL);//建议低电平唤醒
wifi_fpm_set_wakeup_cb(ssc_fpm_wakup_call);
wifi_fpm_do_sleep(FPM_SLEEP_MAX_TIME);
```

3.20. ESP8266 FRC1 的 hw_timer 如何使用?

1. 模型：



(1) FRC1 的参考时钟为 80 M，分频系数可以配置为 1 分频，16 分频，256 分频，不同的分频，影响每个 tick 的时长。

(2) FRC1 为递减型 timer，当 `COUNT_VALUE` 的值减到 0 后会触发中断。每一个 tick，`COUNT_VALUE` 的值都会减 1。

(3) FRC1 可以配置为自动填装模式和非自动填装模式。

- 自动填装模式：当发生中断后，`COUNT_VALUE` 会自动取 `FRC1_LOAD_VALUE` 的值，放到本身，做自减操作。
- 非自动填装模式：当发生中断后，`COUNT_VALUE` 会切到最大值 0x7fffff 开始计。

(4) FRC1 的中断可以配置为 FRC1 中断源和 NMI 中断源。

NMI 中断被称为 CPU 不可屏蔽中断。NMI 中断在 ESP8266 对应 LEVEL3 的中断，其他中断对应 LEVEL1 的中断，NMI 中断可以打断任意比其优先级低的中断。

2. 关于 SDK HW_TIMER 的注意事项

SDK 中 hw_timer 的分频系数为 16，每个 tick 的时长为 0.2 µs。hw_timer_arm 函数配置的参数单位为 µs，最大值为 1677000 µs。

3.21. 如何让 ESP8266 上电后快速连接 AP?

ESP8266 与某个 AP 连接后，会将该 AP 的信道信息存储在 RTC memory 中。

- 当软件复位 ESP8266，或 ESP8266 从 Deep-sleep 模式中唤醒之后，ESP8266 会从 RTC memory 中读取 AP 的信道信息，并尝试连接该信道中的 AP。
- 但如果上电启动或硬件复位 ESP8266，RTC memory 会被清空。因此，ESP8266 会扫描所有的信道，这会占用一些时间。

用户上电启动或硬件复位 ESP8266 时，可通过以下方式存储 AP 的信道信息，以避免 ESP8266 重新扫描所有信道寻找上次连接的 AP。这将有助于减少 ESP8266 启动后的连接时间。

- 在 ESP8266 与 AP 连接后，调用函数 wifi_get_channel 来读取当前 AP 的信道信息，然后将该信息存储到 SPI Flash 中。在写入 AP 的信道信息之前，请确保该信道信息有效。
- ESP8266 上电或硬件复位时，用户固件将从 Flash 中读取之前存储的信道信息。通过调用函数 WRITE_PERI_REG(0x600011f4, 1 << 16 | channel) 将该 AP 的信道信息写入 RTC memory 中。之后便可以从 RTC memory 获取信道信息，加快 ESP8266 与 AP 的连接。
- 使能自动连接功能后，ESP8266 会从 RTC memory 中读取 AP 的信道信息，并尝试连接该信道中的 AP。

只有信道信息会被存储在 RTC memory 中。当调用函数 wifi_station_set_config 时，配置的其它信息（比如 SSID 和密码）已存储在 Flash 中。

> ⚠ **注意：**
>
> 如果应用要求 *ESP8266* 频繁上电或硬件复位，建议客户使用片外 *RTC memory* 来备份信道信息。由于 *Flash* 内存的写入周期有限，不建议对 *Flash* 内存频繁地写入。

3.22. 为什么 ESP8266 进入启动模式（2,7）并触发看门狗复位?

请确保 ESP8266 启动时，strapping 管脚处于所需的电平。如果外部连接的外设使 strapping 管脚进入到错误的电平，ESP8266 可能进入错误的操作模式。在无有效程序的情况下，看门狗计时器将复位芯片。

因此在设计实践中，建议仅将 strapping 管脚用于连接高阻态外部器件的输入，这样便不会在上电时强制 strapping 管脚为高/低电平。

3.23. ESP8266 上电时打印的 boot 模式信息代表什么？如何改变 boot 模式?

ESP8266 上电时会判断 boot strapping 管脚的状态，并决定 boot 模式。例如，ESP8266 上电时打印的 boot 模式信息如下：

```
ets Jan  8 2013,rst cause:1, boot mode:(3,2)
```

其中打印的 boot mode 的第一位数字（3）代表当前的 boot 模式。

Boot 模式由 strapping 管脚的 3 位值 [GPIO15，GPIO0，GPIO2] 共同决定。如下表所示：

Strapping 管脚的 3 位值/[GPIO15，GPIO0，GPIO2]	Boot 模式
7 / [1，1，1]	SDIO HighSpeed V2 IO
6 / [1，1，0]	SDIO LowSpeed V1 IO
5 / [1，0，1]	SDIO HighSpeed V1 IO
4 / [1，0，0]	SDIO LowSpeed V2 IO
3 / [0，1，1]	Flash Boot
2 / [0，1，0]	Jump Boot
1 / [0，0，1]	UART Boot
0 / [0，0，0]	Remapping

3.24. 为什么 ESP8266 启动时打印 *ets_main.c*，并且无法正常运行?

ESP8266 启动时打印 ***ets_main.c***，表示没有可运行的程序区，无法运行；遇到这种问题时，请检查烧录时的 bin 文件和烧录地址是否正确。

4. 硬件

4.1. ESP8266 电压电流需求?

ESP8266 的数字部分的电压范围是 1.8 V ~ 3.3 V。

模拟部分的工作电压是 3.0 V ~ 3.6 V，最低 2.7 V。

模拟电源峰值 350 mA。

数字电源峰值 200 mA。

注意：选择的 SPI Flash 工作电压也需要与 GPIO 的电压匹配。

CHIP_EN 还是工作在 3.0 ~ 3.6 V，使用 1.8 V GPIO 控制时需要注意电平转换。

4.2. 设计 ESP8266 的供电时，需要注意哪些问题?

请注意如下几点：

1. 如果是使用 LDO 变压，请确保输入电压和输出电压要足够大。
2. 电源轨去耦电容器必须接近 ESP8266 摆放，等效电阻要足够低。
3. ESP8266 不能直连 5 V 电压。
4. 如果是通过 DC-DC 给 ESP8266 供电，必要时要加上 LC 滤波电路。

4.3. ESP8266 上电时电流很大，是什么原因?

ESP8266 的 RF 和数字电路具有极高的集成度。上电后，RF 自校准会需要大电流。模拟部分电路最大的极限电路可能达到 500 mA；数字电路部分最大电流达到 200 mA。一般的操作，平均电流在 100 mA 左右。

因此，ESP8266 需要供电能达到 500 mA，能够保证不会有瞬间压降。

4.4. 可以使用锂电池或者两节 AA 纽扣电池直接给 ESP8266 供电吗?

两节 AA 纽扣电池可以给 ESP8266 供电。锂电池放电时压降比较大，不适合直接给 ESP8266 供电。ESP8266 的 RF 电路会受温度及电压浮动影响。不推荐不加任何校准的电源直接给 ESP8266 供电。推荐使用 DC-DC 或者 LDO 给 ESP8266 供电。

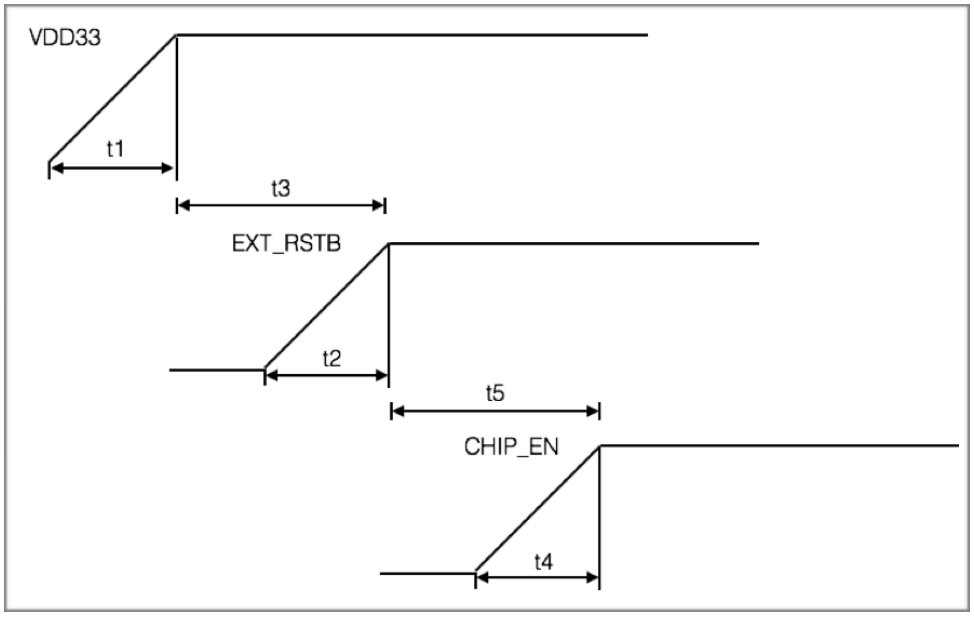
4.5. SPI Flash 上电时，是否有特殊需求?

SPI Flash 用于存储用户的程序和数据。为了保证兼容性，SPI Flash 的电压应该和 GPIO 的电压相匹配。

4.6. 上电时序是怎样的，boot 模式是如何选择的?

CHIP_EN 上电时序要求：CHIP_EN 芯片使能管脚，内部无上拉，高电平有效。CHIP_EN 的上电要晚于或同时与系统电源 3.3 V 上电。一般 CH_EN 有外接 RC 电路，延时大概 在 µs 级即可。CHIP_EN 拉高大概 60 ms 后，设备判断 boot mode {GPIO15, GPIO0, GPIO2}，之后 UART 即可通讯。

EXT_RSTB：外部复位管脚，内部有上拉，悬空即为高电平。EXT_RSTB 为电平触发，低电平触发芯片复位。如果是外部给 ESP8266EX 的 reset 信号，则最低要求（0.25 VIO, 100 µs）。



	描述	最小值	最大值	单位
t1	VDD33 上升时间	10	2000	µs
t2	EXT_RSTB 上升时间	0	2	ms
t3	EXT_RSTB 电平 在 VDD33 电平为高后上升	0.1	-	ms
t4	CHIP_EN 上升时间	0	2	ms
t5	CHIP_EN 电平 在 EXT_RSTB 电为高后上升	0.1	-	ms

4.7. ESP8266 的 RAM 的使用结构是怎么的?

ESP8266 的 RAM 总共 160 KB。

- IRAM 空间为 64 KB：

  前 32 KB 用作 IRAM，用来存放没有加 `ICACHE_FLASH_ATTR` 的代码，即 .text 段，会通过 ROM code 或二级 boot 从 SPI Flash 中的 BIN 中加载到 IRAM；

  后 32 KB 被映射作为 iCache，放在 SPI Flash 中的，加了 `ICACHE_FLASH_ATTR` 的代码会被从 SPI Flash 自动动态加载到 iCache。

- DRAM 空间为 96 KB：

  对于 Non-OS_SDK，前 80 KB 用来存放 ***.data/.bss/.rodata/heap***，heap 区的大小取决于 ***.data/.bss/.rodata*** 的大小；还有 16 KB 给 ROM code 使用。

  对于 RTOS_SDK，96 KB 用来存放 ***.data/.bss/.rodata/heap***，heap 区的大小取决于 ***.data/.bss/.rodata*** 的大小。

5. 外设

5.1. ADC 的性能参数有几个通道？采样率和有效位数是多少？

通道： 1

采样率：

- 停止 Wi-Fi 的情况下，能达到 每秒 100000 次。
- Wi-Fi 正常工作的情况下，能达到每秒 1000 次。

有效位数：

内部 ADC 有效位数为 12 位。

`system_adc_read()`//API 返回值的有效位数是 10 位。

5.2. 从哪里可以得到 ADC 的寄存器“bitmap”信息？

ADC 是和内部 RF 电路高度集成的，所以 bitmap 和寄存器信息没有公开，如有特殊需求请与技术支持联系。

5.3. ADC 的精度如何？

ESP8266 连接路由器后，单 STA 模式会进入 modem-sleep，导致芯片内部电流发生变化，参考值变化，因此 ADC 采集异常。

用户如果需要测量的非常准确，可以用 `system_adc_fast_read` 的函数，但是测量之前需要关闭 RF，Wi-Fi 连接会断开。如果需要测试比较准确，数值相差 1，或 2，可以配置 Wi-Fi 为 non-sleep 模式 `wifi_set_sleep_type(NONE_SLEEP_T)`；建议该用户这样配置。

如果对精确性要求不高，可以允许模块进入 sleep 模式，功耗较低。

5.4. 内部 ADC 的用途是什么？

内部 ADC 可以用于温度检测和粗略地测量外部设备电流。由于 ADC 容易受噪声影响，所以推荐只在低精度的需求时使用。比如熔断机制。

5.5. (u8 tx_addr, u8 tx_cmd, u8 tx_rep) 这三个参数是什么意思？

`tx_addr` 是发送地址；

`u8 tx_cmd` 是发送指令；

`u8 tx_rep` 是重复发送的次数。

5.6. 为什么 ESP8266 上电时会出现乱码？如何修改波特率？

如果使用的是 26 MHz 晶振，ESP8266 UART0 上电后的波特率是 74880，所以上电时会有乱码。

客户可以在 ***user_main.c*** 里面修改 UART 配置，比如：

```
void ICACHE_FLASH_ATTR
uart_init(UartBautRate uart0_br, UartBautRate uart1_br)
{
    // rom use 74880 baut_rate, here reinitialize
    UartDev.baut_rate = uart0_br;
    uart_config(UART0);
    UartDev.baut_rate = uart1_br;
    uart_config(UART1);
}
```

5.7. 如何使能 UART 流控？

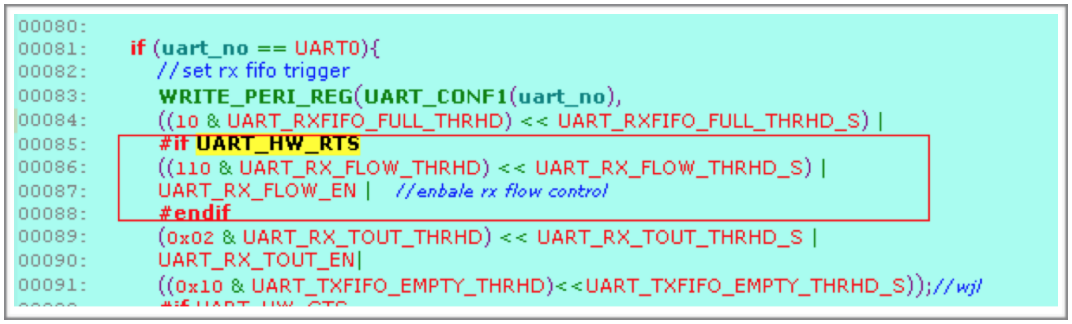
1. UART 通信时，如需配置 UART 通信的数据格式，请参考 ***SDK/driver_lib/driver/*** 路径下的 ***uart.c*** 文件。
2. UART 通信，如需配置硬件流控，请执行下面两个步骤：

   (1) 请在 ***uart.h*** 中将下面的宏置 1。

   ```
   #define UART_HW_RTS   1 //set 1: enable uart hw flow control RTS, PIN
   MTDO, FOR            UART0
   #define UART_HW_CTS  1    //set1: enable uart hw flow contrl CTS , PIN
   MTCK, FOR            UART0
   ```

   (2) 配置硬件流控的门限值

   截图中红色标注部分为硬件流控的门限值，在 RXFIFO 中字节数大于 110 后，RTS 会被拉高。



5.8. 如何配置信息打印到 UART1 上?

UART1 只有 Tx 功能，可以在 UART0 用于通讯时，做打印 log 用途。

请参考如下代码：

```
void ICACHE_FLASH_ATTR
uart_init_new(void)
{
// Wait for FIFOs to be emptied
    UART_WaitTxFifoEmpty(UART0);
    UART_WaitTxFifoEmpty(UART1);
// Configure UART settings
    UART_ConfigTypeDef uart_config;
    uart_config.baud_rate    = BIT_RATE_74880;
    uart_config.data_bits     = UART_WordLength_8b;
    uart_config.parity          = USART_Parity_None;
    uart_config.stop_bits     = USART_StopBits_1;
    uart_config.flow_ctrl      = USART_HardwareFlowControl_None;
    uart_config.UART_RxFlowThresh = 120;
    uart_config.UART_InverseMask = UART_None_Inverse;
    UART_ParamConfig(UART0, &uart_config);


    UART_IntrConfTypeDef uart_intr;
    uart_intr.UART_IntrEnMask = UART_RXFIFO_TOUT_INT_ENA |
UART_FRM_ERR_INT_ENA | UART_RXFIFO_FULL_INT_ENA;
    uart_intr.UART_RX_FifoFullIntrThresh = 100;
    uart_intr.UART_RX_TimeOutIntrThresh = 2;
    uart_intr.UART_TX_FifoEmptyIntrThresh = 20;
    UART_IntrConfig(UART0, &uart_intr);
// Set UART1 for printing
    UART_SetPrintPort(UART1);
// Register interrupt handler
    UART_intr_handler_register(uart0_rx_intr_handler);
    ETS_UART_INTR_ENABLE();
}
```

5.9. SDIO 是否支持 SD 卡?

ESP8266 是 SDIO Slave，不支持 SD 卡。

5.10. SDIO 最高速度能支持到多少?

SDIO 时钟能到 50 MHz，理论最高速度是 200 Mbps。

5.11. 为什么上电时会有 LED 灯闪一下的情况?

要看灯的驱动是如何设计的。如是低电平灯亮，并且在上电的时候将 IO 强制拉为低电平，那么在上电的瞬间可能会出现灯闪一下。是因为除了 Flash 相关的 IO 和 GPIO4，GPIO5，其他 IO 上电后上拉默认使能。

解决方法：

1. 上电的瞬间，`user_init` 中将上拉关闭。
2. 如第一条无效的，需要我们提供相应的 ***boot.bin***。在该 BIN 被搬到 RAM 的过程里，IO 的上拉就会被关闭。这比 `user_init` 生效要早。

5.12. 使用 PWM 时，发现最开始时有窄波，是什么原因?

这个是精度较高的 PWM 的调节方式，PWM 的精度可以达到 22222 深度。精度的调节主要靠后面的窄波。注意这种方式的 PWM Duty 无法配置为 100%。

5.13. 发现 PWM 的变化缓慢，是什么原因?

客户采用了 ***SDK example/IOT_demo*** 中的渐变 API。如 `light_set_aim` 或 `light_set_aim_r` 这些 API 使用的是渐变方式。不会立即生效，需要渐变的过程。

如用户需要 PWM Duty 设置后立即生效，需要调用接口 `pwm_set_duty`，需要注意调用 `pwm_set_duty` 后要调用 `pwm_start` 此次设置才能生效。

5.14. GPIO 可以直接连 5 V 吗?

不可以。GPIO 只能承受 3.6 V。需要通过降压电路，否则会造成 GPIO 损坏。

5.15. 哪里能找到 GPIO 的 register 和 bitmap 信息?

请参考文档《*ESP8266 技术参考*》。

5.16. 如何编程 GPIO?

对于Non-OS SDK，比如需要把 MTDO 配置成输入，同时下降沿触发中断：

```
void ICACHE_FLASH_ATTR gpio_init(void)

{

 //GPIO Alternate Function
    PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDO_U,FUNC_GPIO15);
    GPIO_DIS_OUTPUT(GPIO_ID_PIN(15));                                   //
Configure it in input mode.
    ETS_GPIO_INTR_DISABLE();                                            //Close
the GPIO interrupt
//Register the interrupt function
    ETS_GPIO_INTR_ATTACH(GPIO_INTERRUPT,NULL);

    gpio_pin_intr_state_set(GPIO_ID_PIN(15),GPIO_PIN_INTR_NEGEDGE);     //
Falling edge trigger
    ETS_GPIO_INTR_ENABLE() ;                                            //
Enable the GPIO interrupt
}
```

对于 RTOS SDK，实现相同的功能。

```
{

    GPIO_ConfigTypeDef gpio_in_cfg;                                     //
Define GPIO Init Structure
    gpio_in_cfg.GPIO_IntrType = GPIO_PIN_INTR_NEGEDGE;                  //
Falling edge trigger
    gpio_in_cfg.GPIO_Mode = GPIO_Mode_Input;                            //Input
mode
    gpio_in_cfg.GPIO_Pin  = GPIO_Pin_15;                                //
Enable GPIO
    gpio_config(&gpio_in_cfg);                                          //
Initialization function
    GPIO_REG_WRITE(GPIO_STATUS_W1TC_ADDRESS, BIT(GPIO_UART_IN_NUM));
    gpio_intr_handler_register(interrupt_GPIO_UART);                    //
Register the interrupt function
    _xt_isr_unmask(1 << ETS_GPIO_INUM);                                 //
Enable the GPIO interrupt

}
```

> ⚠ **注意：**
>
> *Non-OS SDK* 和 *RTOS SDK* 的实现方法稍有不同。

5.17. HSPI 每个数据包的大小最大是多少?

数据包每次发送最大 64 字节数据，四字节对齐，在 memory map 上是连续的。数据可以先传输到 LSB 或者 MSB 中，低位组在前或高位组在前，所以缓存是很灵活的。合理的使用 buffer，可以稳定实现 90% 的时钟速度。

5.18. 对于多设备同时连接到 ESP8266 的情况，HSPI 是如何同时驱动设备的?

与 I2C 接口不同，HSPI 时钟不需要配合最慢的连接设备。HSPI 时钟可以及时的配置为使能的设备。`HSPI_CS0` 管脚可以自动选择。对于 LCD 类型的设备，需要频繁使用 CS 管脚。

如果是连接高速设备，需要使用源端接电阻。

5.19. 如何使用 64 字节的数据缓存?

- 使用函数

  ```
  CLEAR_PERI_REG_MASK(SPI_USER(spiNum), SPI_USR_MISO_HIGHPART );
  ```

- 使能 MISO 的高位传输。

5.20. 如何配置 (H)SPI 接口?

请参考 Non-OS SDK 下，***example/peripheral_test***。

5.21. 哪些 API 会保存到 Flash?

```
wifi_station_set_auto_connect
wifi_station_ap_number_set
wifi_set_phy_mode
wifi_softap_set_config
wifi_station_set_config
wifi_set_opmode
system_restart_enhance
system_restore
system_upgrade_reboot
```

5.22. 系统参数是如何保存的?

SPI Flash 的最后三个扇区被定义为系统参数区，其中前两个扇区用于交替保存系统参数，最后一个扇区用来保存使用前面两个扇区的 flag。这样设计的目的是保证在擦写系统参数区时，即使在擦写任意一个扇区时意外掉电，导致该扇区中数据异常的情况下，也不会导致系统的参数异常。

5.23. Flash 任何位置都可以随意读写吗?

读写操作都需要四字节对齐。我们推荐对 block 操作，避免频繁的小数据读写。

5.24. 可以在所有的 ESP8266 上执行同样的 Flash 读写操作吗?

对应不同的 Flash 大小，有着不同的 Flash map，所以对应不同的固件，Flash map 可能不一样，比如对应 8 Mbit 的 Flash，读写 0x100000 地址就是非法的。

5.25. 可否提供 Flash 擦写例证?

uint32 sector 是开始的 sector，uint 32 cnt 是擦除的 sector 数目。

```
#define FLASH_WRITE_LEN_BYTE (1024*4)
#define FLASH_WRITE_CONTENT (0X10)
void flash_sector_rw_test(uint32 sector,uint32 cnt)
{
    char* w_data=(char*)os_malloc(FLASH_WRITE_LEN_BYTE);
    uint32 flash_operate=0;
    uint32 i=0;
    uint8 status=0;
    os_printf("Test Sector is 0x%x\n",sector);
    if(NULL==w_data){
        os_printf("Memory little\n");
        return;
    }
    os_memset(w_data,FLASH_WRITE_CONTENT,FLASH_WRITE_LEN_BYTE);
    for(i=0;i<cnt;i++){
        if(spi_flash_erase_sector(sector+i)==0)
        {
            os_printf("erase sector0x%x ok\n",sector+i);
        }
        else{
            os_printf("Err:erase sector0x%x err\n",sector+i);
        }
    }
    for(i=0;i<cnt;i++){
```

```
        if(spi_flash_write((sector+i)*(FLASH_WRITE_LEN_BYTE),
(uint32*)w_data,FLASH_WRITE_LEN_BYTE)==0)
        {
            os_printf("write sector 0x%x ok\n",sector+i);
        }
        else{
            os_printf("Err:write sector 0x%x err\n",sector+i);
        }
    }
   for(i=0;i<cnt;i++){
           os_memset(w_data,0x00,FLASH_WRITE_LEN_BYTE);
        if(spi_flash_read((sector+i)*(FLASH_WRITE_LEN_BYTE),
(uint32*)w_data,FLASH_WRITE_LEN_BYTE)==0)
        {
            uint32 j=0;
            for(j=0;j<FLASH_WRITE_LEN_BYTE;j++){
             if(*(w_data+j)!=FLASH_WRITE_CONTENT){
                 os_printf("Err Flash read w_data[%d]=0x%x\n",j,w_data[j]);
                 //status=1;

            }
           if(*(w_data+j)==FLASH_WRITE_CONTENT&&j==FLASH_WRITE_LEN_BYTE-1)
           {
                os_printf("Sector0x%x Test Ok\n",sector+i);
                        }
    }
        }
        else{
            os_printf("Err:read sector0x%x err\n",sector+i);
        }
   }
    os_free(w_data);
    w_data=NULL;
}
```

5.26. 如何判断 Flash 是否支持 QIO 或 DIO 模式?

判断Flash是否支持四线：

1. QE 在状态寄存器的 BIT(9)。
2. 写状态寄存器的格式为 01H+StatusReg1+StatusReg2。
3. 有读 Flash 的如下命令:
   - 若是选择 QIO，必须支持 EBh 命令。
   - 若是选择 QOUT，必须支持 6 Bh 命令。

判断 Flash 是否支持两线：

有读 Flash 的如下命令:

- 若是选择 DIO，必须支持 BBh 命令
- 若是选择 DOUT，必须支持 3 Bh 命令。

特殊的 Flash：ISSI Flash 可以支持 QIO 模式。

5.27. 为什么透传过程会丢包?

因为没有设置硬件流控。如果需要避免丢包，请设置硬件流控。透传功能使用的是 TCP 协议，每包数据是 1460（取决于协议栈），只要网络良好，buffer 空间没有被消耗完，就可以不停地传输数据。对于透传，串口接收数据间隔超过约 20 ms，就会认为数据接收结束，将已经接受的数据传输到网络。如果网络不好，就可能会丢弃一些数据，因此，为避免这种情况，可以将串口设置为流控模式。

5.28. ESP8266 有几个 UART?

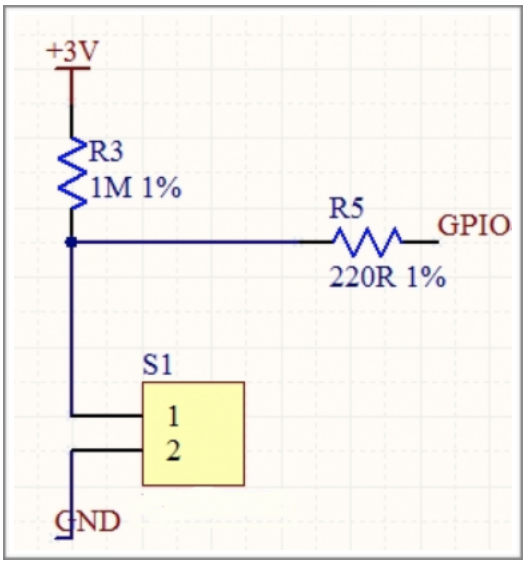
ESP8266 有两个 UART，其中 UART0 有 TX、RX，可做数据传输；UART1 由于 RX 脚被 SPI-Flash 占用，只能使用 TX，可以做串口调试信息打印。

5.29. GPIO 电平状态是怎样的?

除了 XPD_DCDC，GPIO 可以配置上拉。关于 GPIO 的上电 IO 口默认状态为：除了 SDIO 6 根线、GPIO4、GPIO5、GPIO16 上电 IO 默认无上拉，其他的 GPIO 口均有上拉。由于是内部配置上拉，所以如需下拉，需外部加下拉方式或者加一个三级管的反相电路。

⚠ **注意：**

GPIO 不能到 *5 V*。*GPIO4/5* 外接 *1 M* 电阻不能上拉到高电平；需 *100 K* 电阻。



5.30. 如何屏蔽上电打印?

U0TXD 默认上电有系统打印，对此敏感应用可通过 UART 的内部引脚交换功能，在初始化的时候，调用 `system_uart_swap` 函数，将 U0TXD、U0RXD 分别与 U0RTS（MTDO/GPIO15），U0CTS（MTCK/GPIO13）交换来屏蔽该上电的系统打印。

交换后，硬件上的下载管脚还是使用 U0TXD + U0RXD，通信时需要将 MTDO 对应接到 MCU 的 RXD，MTCK 对应接到 MCU 的 TXD。

6. 协议

6.1. TCP / UDP 的包长是多少?

单包数据，TCP 单包 1460 字节，UDP 单包 1472 字节。

7. RF

7.1. 如何修改默认上电校准方式?

- 上电时 RF 初始化默认采用部分校准的方案

  esp_init_data_default.bin 中第 115 字节为 ***0x01***，RF 初始化时间较短 。

- 不关注上电启动时间，可修改使用上电全校准方案
  - 使用 NONOS SDK 及 RTOS SDK 3.0 以前的版本：

    a）在 ***user_pre_init*** 或 ***user_rf_pre_init*** 函数中调用 ***system_phy_set_powerup_option(3)***；

    b）修改 ***phy_init_data.bin*** 中第 115 字节为 ***0x03***。

  - 使用 RTOS SDK 3.0 及以后版本：

    a）在 ***menuconfig*** 中关闭 ***CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE***；

    b）如果在 ***menuconfig*** 中开启了 ***CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION***，修改 ***phy_init_data.bin*** 中第 115 字节为 ***0x03***；

    如果没有开启 ***CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION***，修改 ***phy_init_data.h*** 中第 115 字节为 ***0x03***。

- 继续使用上电部分校准方案，若需在业务逻辑中增加触发全校准操作的功能
  - 使用 NONOS SDK 及 RTOS SDK 3.0 以前的版本：

    擦除 RF 参数区中的内容，触发全校准操作

  - 使用 RTOS SDK 3.0 及以后版本：

    擦除 NVS 分区中的内容，触发全校准操作

8. Wi-Fi

8.1. 设备开启 SoftAP + Station 模式下，连接的路由是 192.168.4.X 网段时，为什么会失败?

ESP8266 SoftAP 默认 IP 地址是 192.168.4.1。

ESP8266 如果要连接 192.168.4.X 的路由时，不能分辨是要连接自己本身的 SoftAp 还是外部路由，所以会造成错误。

8.2. 路由配置是正确的，但是发生找不到路由，连接失败，为什么?

如果 SSID 和密码配置是正确的，可能的原因有 2 个。

1. 推荐使用英文字符，不要使用中文。
2. 需要注意 `bssid_set` 的设置，如果不需要指定路由的 MAC 地址，那么需配置 `stationConf.bssid_set = 0`。

8.3. 调用 wifi_softap_set_config() 时，函数返回成功，但为何无法修改 ESP8266 的 SoftAP SSID 和密码?

使用函数 `wifi_softap_set_config()` 时，如果 API 从回调函数内部调用，ESP8266 SoftAP 的配置有时候会修改失败。例如，当应用程序试图在 SoftAP 事件的回调函数内，从 SoftAP 模式切换到 Station 模式时，可能出现这种情况。

为确保 `wifi_softap_set_config()` 所做的修改立即生效，请使用 `system_os_task()` API 创建一个更改 SoftAP 设置的任务。在调用任何 SoftAP API 之前，请确保 ESP8266 已成功切换到 SoftAP 模式。例如：

```
LOCAL void ICACHE_FLASH_ATTR
some_callback_function (void)
{
  unsigned char res;
  os_event_t *testQueue;

  res = wifi_set_opmode_current (0x02);         // 确保 ESP8266 处于 SoftAP
模式。
  os_printf ("\r\nSet op mode returned: %d", res);

  testQueue = (os_event_t *)os_malloc(sizeof(os_event_t)*4);
```

```
  system_os_task (set_ap_config, USER_TASK_PRIO_1, testQueue, 4);

  ap_server_setup (AP_PORT);                    // 继续设置服务器等。
}
void set_ap_config (os_event_t *e)
{
   struct softap_config ap;

   wifi_softap_get_config(&ap);                 // 先获得之前的设置。

   os_memset(ap.ssid, 0, 32);
   os_memset(ap.password, 0, 64);

   os_memcpy(ap.ssid, "SSIDhere", 8);
   os_memcpy(ap.password, "PASSWDhere", 10);

   ap.authmode = AUTH_WPA2_PSK;
   ap.ssid_len = 0;                             // 或者 SSID 的实际长度。
   ap.max_connection = 1;                       // 允许接入 Station 的最大
数量。
   wifi_softap_set_config (&ap);                // 更新 ESP8266 SoftAP 设
置。
}
```

8.4. ESP8266 SoftAP + Station 模式下网络断开或丢包的情况?

虽然 ESP8266 支持 SoftAP + Station 共存模式，但是 ESP8266 实际只有一个硬件信道，由 ESP8266 Station 与 SoftAP 接口共用。因此在 SoftAP + Station 模式时，ESP8266 SoftAP 会动态调整信道值与 ESP8266 Station 一致。这个限制会导致 ESP8266 SoftAP + Station 模式时一些行为上的不便，用户请注意。例如：

- 情况一
  - 如果 ESP8266 Station 连接到一个路由（假设路由信道号为 6）；
  - 通过接口 `wifi_softap_set_config` 设置 ESP8266 SoftAP；
  - 若设置值合法有效，该 API 将返回 true，但信道号仍然会自动调节成与 ESP8266 Station 接口一致，在这个例子里也就是信道号为 6。
- 情况二
  - 调用接口 `wifi_softap_set_config` 设置 ESP8266 SoftAP（例如信道号为 5）；
  - 其他 Station 连接到 ESP8266 SoftAP；

- 将 ESP8266 Station 连接到路由（假设路由信道号为 6）；
- ESP8266 SoftAP 将自动调整信道号与 ESP8266 Station 一致（信道 6）；
- 由于信道改变，之前连接到 ESP8266 SoftAP 的 Station 的 Wi-Fi 连接断开。

- 情况三
  - 其他 Station 与 ESP8266 SoftAP 建立连接；
  - 如果 ESP8266 Station 一直尝试扫描或连接某路由，可能导致 ESP8266 SoftAP 端的连接断开，或者 UDP 丢包，ping 丢包等情况。

因为 ESP8266 Station 会遍历各个信道查找目标路由，意味着 ESP8266 其实在不停切换信道，ESP8266 SoftAP 的信道也因此在不停更改。这可能导致 ESP8266 SoftAP 端的原有连接断开，或者 UDP 丢包，ping 丢包等情况。

这种情况，用户可以通过设置定时器，超时后调用 `wifi_station_disconnect` 停止 ESP8266 Station 不断连接路由的尝试；或者在初始配置时，调用 `wifi_station_set_reconnect_policy` 和 `wifi_station_set_auto_connect` 禁止 ESP8266 Station 尝试重连路由。

8.5. Wi-Fi 信道是什么？可以自行选择信道吗？

信道指的是 Wi-Fi 使用的指定频段中特定频率的波段。不同国家地区使用的信道数目是不同的。

用户可以参考《*ESP8266 Wi-Fi 信道选择指南*》。

8.6. 如何配置 ESP8266，以便连接到无线路由器？

有关配置连接无线路由器，一般有以下几种方式：

1. smartconfig 一键配置方式，设备在 sniffer 模式扫描特征包的方式。
2. 设备开启 SoftAP，手机连接 SoftAP 后建立稳定的 TCP/UDP 连接后，发送 SSID 和密码。
3. WPS 配置方式，此方式需要设备中增加按键；或连接到设备的 SoftAP 后使用手机软件控制开启 WPS。

9. 工具

9.1. 测试和生产时如何烧录 Flash?

测试时，通过 UART 转 USB 芯片，使用乐鑫提供的 Flash 下载工具进行烧录，具体方法参见文档《*ESP8266 SDK 入门指南*》。

- 下载工具：*http://espressif.com/zh-hans/support/download/other-tools*
- 下载方法：*http://espressif.com/zh-hans/support/download/documents*

生产时，使用 UART 接口烧录 Flash 不方便，使用厂商提供的下载工具可以加快这一过程。用户可以使用乐鑫的 Flash 下载工具把 BIN 文件合成一个完整的固件，通过 Flash 厂商提供的下载器，下载到 Flash 里面，再贴片到模组或产品上。

> 📖 **说明：**
>
> *Flash mode*，*Flash speed*，*Flash size* 选项要在 *combine* 前选好，再执行 *combine* 键，即可得到 ***target.bin***。