## 2024年2月8日星期四

凌晨两点十五分。

本周都在睇乐鑫（ESP）的文档，上网稳环境安装教程，稳官方示例工程。

发现官方示例工程係用esp32作为示例，而我用的是esp32s3。一开始我买的就係esp32，而我一直以为係esp8266，烧录的时候用的是esp8266模式，而模式唔正确係无法匹配进行烧录的，晚上用esp32的示例工程烧录了一整晚，都不明白为什么程序没有被改变，现在明白了。Binary文件这个烧录位置，esp32s3与esp32都并不相同。

当我新建了新的esp32s3工程，并成功编译烧录到板子后，这个USB接口就变成了只能烧录，不能主动退出烧录模式，而debug的log也只从uart接口输出，原本的会闪灯的现在也不闪了。

## 2024年2月9日星期五

凌晨两点五十六分，刚刚结束咗两三铺农。

研究久的这个esp32为何串口输出打印乱码，最后发现係奇偶检验开了，esp32的串口bond rate默认係115200，奇偶校验none，数据位8，停止位1。但这个esp32s3，我之前开咗奇偶检验even（偶），但这个输出的log依旧正常。

我现在为这么快将新的程序烧录入esp32s3感到有些后悔，冇好好的用串口睇清楚到底係调用咗边几个IO控制开发板上的LED，由于冇开发板的原理图，现在感到一头雾水。

打算一边学习esp32的环境，一边写readme文件，作为新手入门的引导。

## 2024年2月10日星期六

凌晨三点二十一分，刚刚花一个钟头写咗一下这个入门readme。

## 2024年2月13日星期二

深夜十一点四十分，今日花咗三到四小时睇文档与进行GPIO的试验。

设置下拉输入模式，检测输入较为顺利，能正常读取到IO的高电平输入。

设置上拉输入模式，将该IO接到板子上的G插针，G插针所对应的应当是GND，但是并没有读取到电位被下拉。

尝试设置中断，中断处理程序中不能打印log，中断中发送任务队列，需要另外创建一个线程任务接收中断发出的任务队列。若在线程任务中移除已注册的中断，似乎该任务也会跟着被释放掉。

## 2024年2月19日星期一

凌晨一点五十三分，昨天下午睇GitHub上的blink示例，但发觉工程实际上唔一样，示例工程的configmenu同我新建工程的cofigmenu的选项并唔一样的。就算develop board上的丝印印着”RGB@IO48”，我亦唔清楚如何正确的将其初始化，我只对其初始化通用输出，设置上拉下拉都未有什么可观察的反应，是否应当将其初始化为LDE\_PWM？

晚上睇API文档，对API进行验证。睇定时器篇。

下午一点四十六分，我仿照之前写的杰里的那个程序结构，写到当前这个项目里。但是没用定时器。DebugLog显示是有的，没有整个OS频繁复位，也就是watchDog没有reset我的MPU，但是仍然会有watchDog timeout。

## 2024年2月20日星期二

我设置了一个2s的定时器，每2s发送一个数据到队列，该队列的数据由我的主任务接收，接收成功则执行任务，接收失败则延时10ms。但实际上我搞不明白，当我队列为空的时候，还是接收成功返回了1。

成功的仿照了以前写杰里项目的程序结构。实践分文件撰写代码，成功编译。

之前修改了对中断处理任务的栈大小为1024，是可用的，但是现在一旦触发中断就导致OS抛异常重启，于是又改回了2048，现在是正常触发且打印log，估计係因为手抖，拔插杜邦线时多次触发中断导致栈溢出了。

这个版本的IDF貌似不支持xTaskList()这个函数，我把宏portUSING\_MPU\_WRAPPERS

打开后报错报了一堆。

## 2024年2月28日星期三

最近两三日都在睇espWIFI的连接。于今日凌晨四点的时候，成功使用ESP-S3的WiFi功能连接到我的AP，但AP上并未有显示新增了信的设备。且连接成功，但使用esp\_wifi\_80211\_tx()这个函数发送一串字符串的时候报错了，由于我使用了ESP\_ERROR\_CHECK宏，然后就一直陷入了重启OS。原来WiFi连接并非网络连接。。。

## 2024年3月8日星期五

我检查咗首次连接上AP后板子的连接情况，确实係首次连接获取到IP后就lost了，或许係需要配置WIFI事件处理函数。但睇ESP-IDF编程指南上的说明，连接到WIFI貌似就係一个简易的配网过程，我并冇设置密码，就唔需要处理密匙安全等级这种较为麻烦的预处理。但貌似我并冇配置联网后的endpoint，且只有一个初始化NET组件得来的基本处理实例。

|  |
| --- |
| const wifi\_prov\_scheme\_t wifi\_prov\_scheme\_console = {  .prov\_start = prov\_start,  .prov\_stop = protocomm\_console\_stop,  .new\_config = new\_config,  .delete\_config = delete\_config,  .set\_config\_service = set\_config\_service,  .set\_config\_endpoint = set\_config\_endpoint,  .wifi\_mode = WIFI\_MODE\_STA  }; |

这个是貌似係配网前要做的准备工作，其中每一个成员都代表了对应处理函数。而我之前将开发板连接到AP，只是做了new cofig这个环节，start在示例中只是打印了一句话。看了一下别人发的视频，我没有接收初始化网络层时返回的实例endpoint，估计通信就是要通过这个endpoint。现在係凌晨四点二十五分，先睡觉了。

## 2024年3月10日星期日

我原本以为係连接上WIFI\_AP后断开咗连接导致的sta\_disconnected事件，逐便在user\_task处添加咗wifi\_connect函数，而事实上WiFi并冇断开，且返回咗connecting，log可见

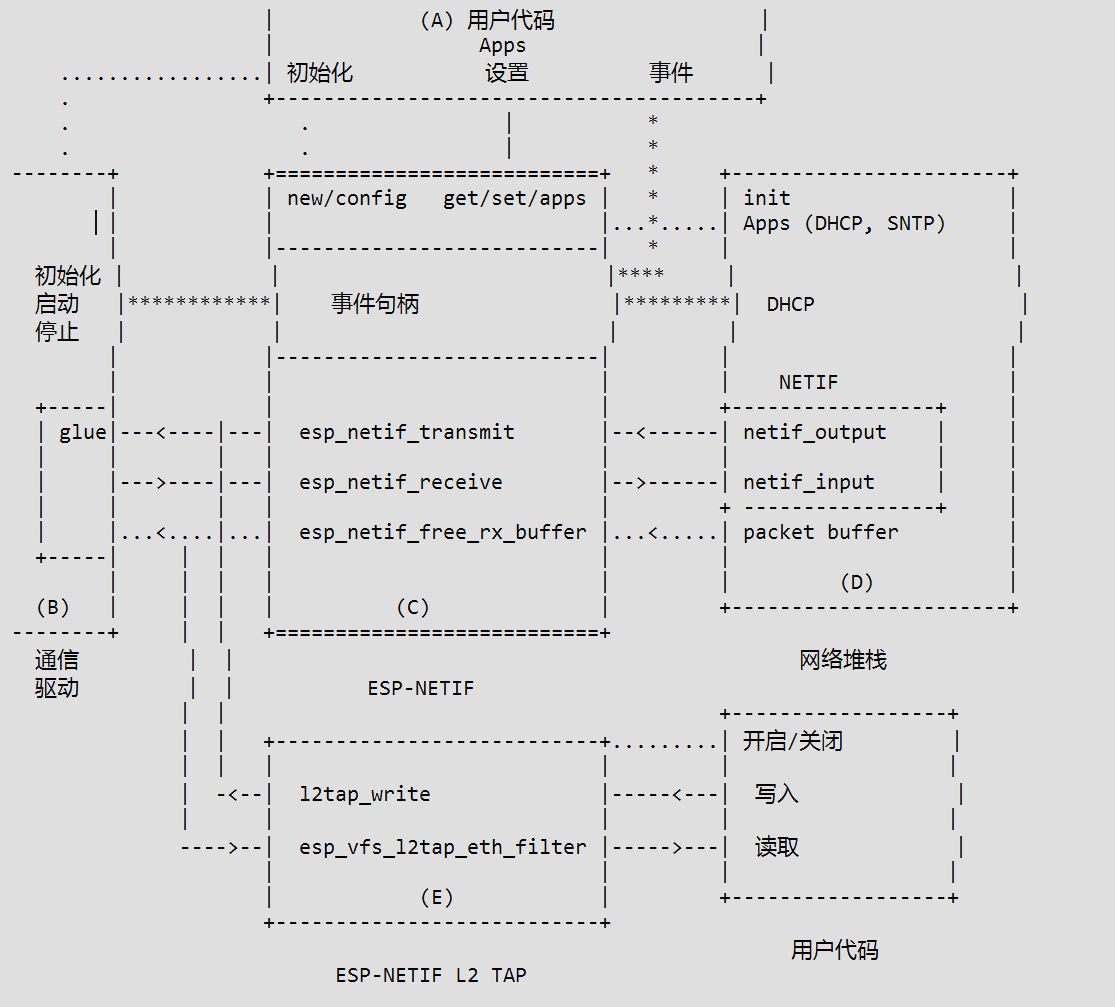
|  |
| --- |
| I (588) wifi:mode : sta (dc:da:0c:4b:32:28)  I (588) wifi:enable tsf  I (588) My\_wifi\_station: WIFI\_EVENT\_STA\_START  E (1498) wifi:sta is connecting, return error  ESP\_ERROR\_CHECK failed: esp\_err\_t 0x3007 (ESP\_ERR\_WIFI\_CONN) at 0x42009268  file: "./main/My\_WIFI\_init.c" line 84  func: WIFI\_STA\_event\_handler  expression: esp\_wifi\_connect()  abort() was called at PC 0x4037c04f on core 0 |

还有，对于WIFI初始化的顺序，不能随意调换，否则会因为某种条件不满足导致OS无限重启，以下是固定顺序

|  |
| --- |
| // Initialize NVS, be must componet ! ! !      esp\_err\_t ret = nvs\_flash\_init();      if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND)      {          ESP\_ERROR\_CHECK(nvs\_flash\_erase()); // erase NVS partition(擦除NVS分区)          ret = nvs\_flash\_init();      }      ESP\_ERROR\_CHECK(ret);      ESP\_ERROR\_CHECK(esp\_netif\_init());                   // 初始化网络层      ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());    // 创建默认事件循环      esp\_netif\_create\_default\_wifi\_sta();                 // 创建默认的 wifi STA 网络接口      wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT(); // 初始化配置, 使用宏将WiFi硬件配置初始化      ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));                // 初始化 wifi |

## 2024年3月20日星期三

乐鑫的Lwip即为TCP/IP协议栈，与用户代码关系如下：



A：由用户代码对lwip进行新建配置

B：初始化通信驱动(nvs\_flash)

C：设定lwip配置，初始化事件

D：初始化lwip堆栈(DHCP——即为获取IP、输入、输出、数据包缓冲区)

E：用户通过lwip层提供的函数进行 读/写

## 2024年3月26日星期二

在沉寂多日后，今日对于这个有了进展，也愿继续钻研下去

[esp-idf/examples/wifi/espnow/main/espnow\_example\_main.c at be06a6f5ffe36f9554cfc91fe2036e0fc85fea60 · espressif/esp-idf (github.com)](https://github.com/espressif/esp-idf/blob/be06a6f5ffe36f9554cfc91fe2036e0fc85fea60/examples/wifi/espnow/main/espnow_example_main.c#L299)

这个 ESP-NOW 貌似就是这个网络层

## 2024年3月27日星期三

今日copy咗GitHub上的esp\_now的样例，还有些地方尚未明白如何配置，如：CONFIG\_ESPNOW\_LMK和CONFIG\_ESPNOW\_PMK，根据注释所写，应当係加密用的数组，但此前尚未使用过这种data加密方式未知如何填写。

2024年3月29日星期五

昨天，项目编译通过了，貌似都成功发送了信息，但这个发送的MAC\_ADDRESS是ff:ff:ff:ff:ff:ff，发送的地址係空的，之所以作这种判断，是因为无返回。然后我更改了地址，是AP的地址，但发送错误了。

问题有点大，昨天还是能成功广播的,今晚改成单播，报错”ESP\_ERR\_ESPNOW\_NOT\_FOUND”，看注释是”peer is not found”，不明白了。就算我把单播改回广播依旧如此。

## 2024年3月30日星期六

esp-now貌似係esp所专有的通信数据结构，并不能用于连接互联网。兜兜转转绕了一大圈，我似乎见到了我想要的[esp-idf/examples/network/eth2ap/main/ethernet\_example\_main.c at fcb76f0671395e4e983ad660edb2a1a6dafccce9 · espressif/esp-idf (github.com)](https://github.com/espressif/esp-idf/blob/fcb76f0671395e4e983ad660edb2a1a6dafccce9/examples/network/eth2ap/main/ethernet_example_main.c)

## 2024年3月31日星期日

似乎又找错了，这次应该是对吧[esp-idf/examples/network/sta2eth/main/sta2eth\_main.c at fcb76f0671395e4e983ad660edb2a1a6dafccce9 · espressif/esp-idf (github.com)](https://github.com/espressif/esp-idf/blob/fcb76f0671395e4e983ad660edb2a1a6dafccce9/examples/network/sta2eth/main/sta2eth_main.c)

## 2024年4月1日星期一

或许我应该从以太网开始睇，睇几日样例，分析样例流程先。。。

## 2024年4月3日星期三

我应该花时间好好睇一下的，一睇就睇到好野了[ESP-NETIF - ESP32-S3 - — ESP-IDF 编程指南 release-v5.2 文档 (espressif.com)](https://docs.espressif.com/projects/esp-idf/zh_CN/release-v5.2/esp32s3/api-reference/network/esp_netif.html?highlight=esp_netif_t#_CPPv417esp_netif_receiveP11esp_netif_tPv6size_tPv)

## 2024年4月7日星期日

于前日和昨日，抄了一下这个https\_request的样例，现在正在分析打印出来的log。从log上来睇，sntp时间同步貌似係失败咗，失败原因未清楚，但这个请求确实是成功了，也成功返回了请求内容，也是请求一次后如代码所写的正常关闭了。不过，我在路由器设置网页上并未有睇到开发板，不知是否因为网络请求只请求了一次的原因。

## 2024年4月8日星期一

目前这个https请求，只能係"https://www.howsmyssl.com/a/check"

其他的请求一律唔可以正常连接

## 2024年4月11日星期四

虽然这个样例名称係”https\_request”，但实际上我感觉唔似HTTPS，而係HTTP，距冇加密。于是我现在睇相关SSL/TLS的HTTPS请求样例。

现在係中午十二点三十四分，写好task了，休息休息下午测试剖析样例。

现在係晚上七点三十八分，午觉睡到下午五点几先起，编译好的binary大小超过咗预设，设置了很长时间将debug\_level配置好先可以正常烧录。但请求失败了。

## 2024年4月13日星期六

看了一晚上HTTPS证书验证，对于编写的两个https请求，一个可以固定访问另一个什么都访问不了的情况，初步怀疑是CA证书问题，正在想办法验证。我又对比了一下在edge中的网页，发现请求方案确实都是用的https，但这个协议不同，一般网页的协议都是”h2”，”chrome-extension”这种，而样例请求的是”http/1.1”，而这个”h2”指的是”http2”，而这个协议栈是否支持”h2”也尚未可知，但我看这个ESP-IDF是能够支持TLS1.3，想来也能支持”h2”才对。又去睇咗下，确实係支持”http2”，那么确实很有可能係证书的问题[ESP-TLS - ESP32 - — ESP-IDF 编程指南 latest 文档 (espressif.com)](https://docs.espressif.com/projects/esp-idf/zh_CN/latest/esp32/api-reference/protocols/esp_tls.html)这是ESP-TLS的链接。一番追本溯源，我怀疑证书有问题，但还不能百分百确定，但感觉证书是空的——所以导致握手失败。

## 2024年4月19日星期五

我决定暂时先将这个放下，实在太痛苦了，而且我冇这个能力，因为唔識的如何去分析这个错误代码0x6c00的成因。先搞一下这个ESP的USB。唔知道为什么使用idf.py命令添加了这个tiny USB组件后编译出错了，问题有点大，这个库多处报错了。

## 2024年4月21日星期日

问题有点大，刚刚编译由于是因为开发板是多核的原因，静态断言报错了，我使得这个报错忽略了，然后烧写好后板子就不跑了，进不了RTOS内核。。。

我新建了一个工程，又可以了，USB能用了。今天复原了板子刚到时的初始代码。

## 2024年4月23日星期二

现在是凌晨零点十五分，昨天把之前写的代码都移植到当前这个新建的项目中了。这个灯闪不能闪太快，会发热，我都担心会不会把开发板烧了，因为试过因为闪太快发热严重之间掉电保护了。然后就是发现调用这个tiny USB的时候，调用这个esp\_tusb\_init\_console再调用esp\_tusb\_deinit\_console可能内存泄漏了，这个esp\_get\_minimum\_free\_heap\_size给我的回报值越来越低，直到内存不足OS重启，现在正在着手检查定位内存是哪里泄漏了。

虽然没有充分的证据证明是调用esp\_vfs\_tusb\_cdc\_unregister的时候释放不完全导致内存泄漏，但似乎有了解决办法，无需接触tusb\_cdc的注册，只需更改std\_stream\_to的方向即可，就不会因为反复”register”&”unregister”，因为这个”register”的时候内存申请多了一点导致”unregister”内存释放不完全，从而内存泄漏。

现在是早上十点二十二分，找到了申请的内存了

esp\_tusb\_init\_console->

esp\_vfs\_tusb\_cdc\_register->

esp\_vfs\_register->

esp\_vfs\_register\_common->

vfs\_entry\_t \*entry = (vfs\_entry\_t\*) malloc(sizeof(vfs\_entry\_t));

但是释放的时候我看着确实有释放内存的操作，可能是记录申请内存的变量不准确。

只修改std\_stream\_to的方向，也就是更改输出方式，不重复注册tusb，确实解决了内存泄漏的问题。

午觉一觉醒来，我修改过的组件被项目自动配置根据hash值重新修改回去了，然后我不死心，跟项目自动配置作对，搞到组件损坏了，删除组件后重新下载才改回来。

## 2024年4月25日星期四

头文件的包含还係最好唔好起头文件中，应当起.c文件中，昨天将之前写的WIFI功能添加进去，编译报错说我未定义引用。。。我把头文件都放.c源文件里头就没问题了。当前我写的这个版本的代码，似乎又出现咗内存泄漏，我reset后OS回报的free heap size: 224160 bytes，运行一晚上后（约六七小时）是222808 bytes，差咗1352 bytes。

8:58----heap: 224184, block: 172032

9:54---- heap:223000, block: 172032

10:40----heap: 223000, block: 172032

11:44----heap: 223000, block:172032

係我大惊小怪了，整个朝早，这个可用区块都冇少，应该係内存碎片化导致的可用内存减少。

写了个调用深度睡眠API的功能，不知道为什么每次调用API以后OS直接就被唤醒了。

## 2024年4月26日星期五

停止了主任务、http\_request的定时器后，也自动唤醒了，我以为是串口监视器的问题，但即便断开了IDF监视器，也依旧如此。调用检查唤醒的API，发现是我配置的这个EXT1中断所导致的唤醒。被唤醒后，定时器又自动继续运行。

遇到了WiFi停止后https\_request在执行导致请求失败，然后wifi\_start抛异常OS重启了。写了异常处理，现在能正常进入深度休眠了，但依旧会被莫名触发唤醒源。