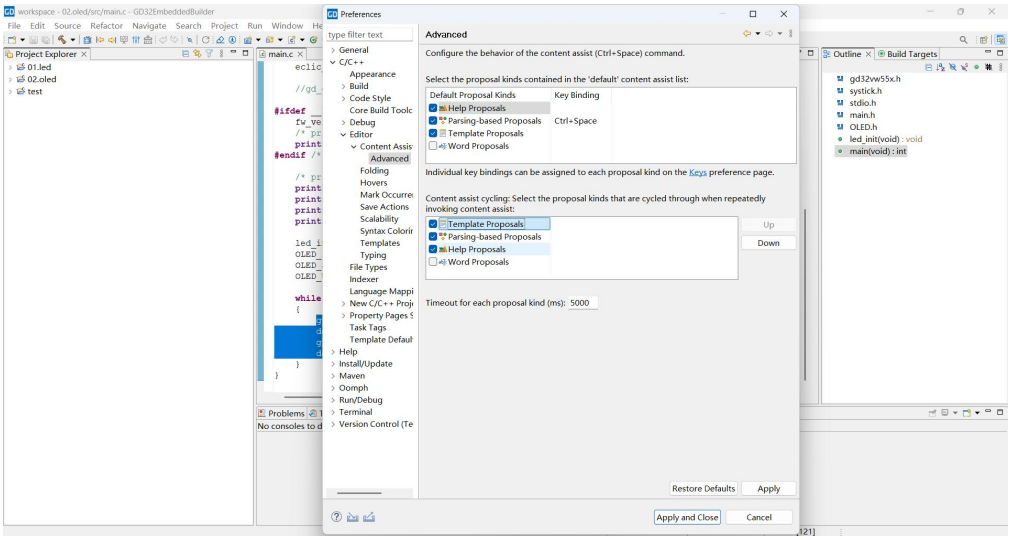
一，开发环境搭建中遇到的问题

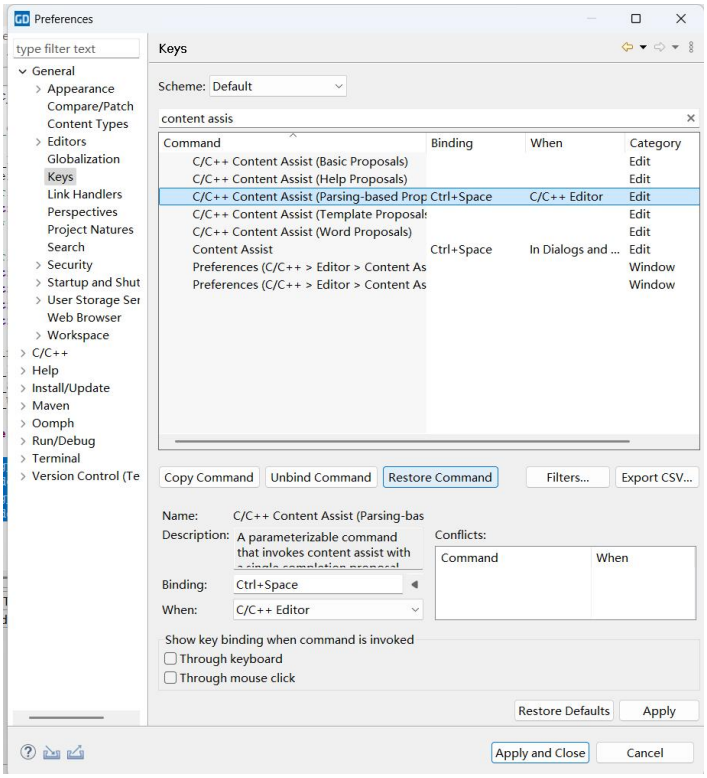
1.代码补全

GD32EmbeddedBuilder 是基于 eclipse 开发的，与 STM32CubeIDE 相同，所以我参考了一个 cubeIDE 代码补全的教程：https://zhuanlan.zhihu.com/p/621198855

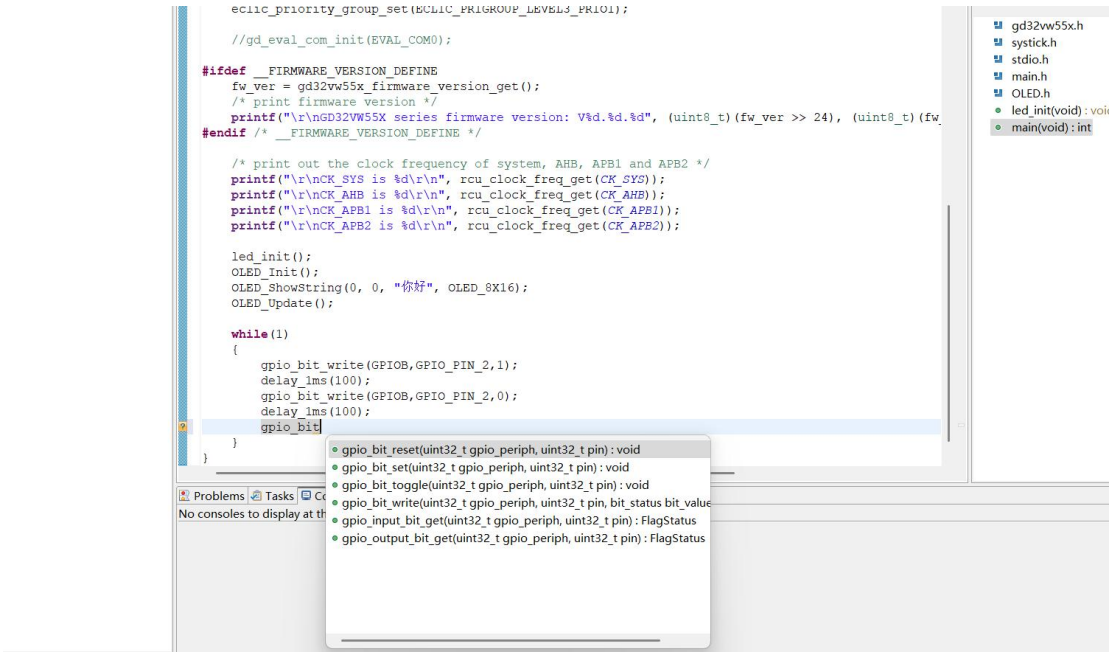
首先，打开程序 Window 菜单下 Preferences 选项，在 C/C++ --> Editor --> Content Assist --> Advanced 设置里勾选上下两部分的 Parsing-based Proposals



然后，我们需要设置相应的快捷键。在 General --> Keys 下面搜索 content assist，设置 C/C++ Content Assist (Parsing-based Proposals)的 Binding 快捷键为你所想设置的，在这里笔者设置成了 Ctrl+Space(空格)，When 设置成 C/C++ Editor。



完成之后我们可以测试一下：输入 gd32 GPIO 操作函数 `gpio_bit` 然后同时按下 `Ctrl` 和空格键，便会弹出自动补全的选项



2.使用 jlink 烧录程序

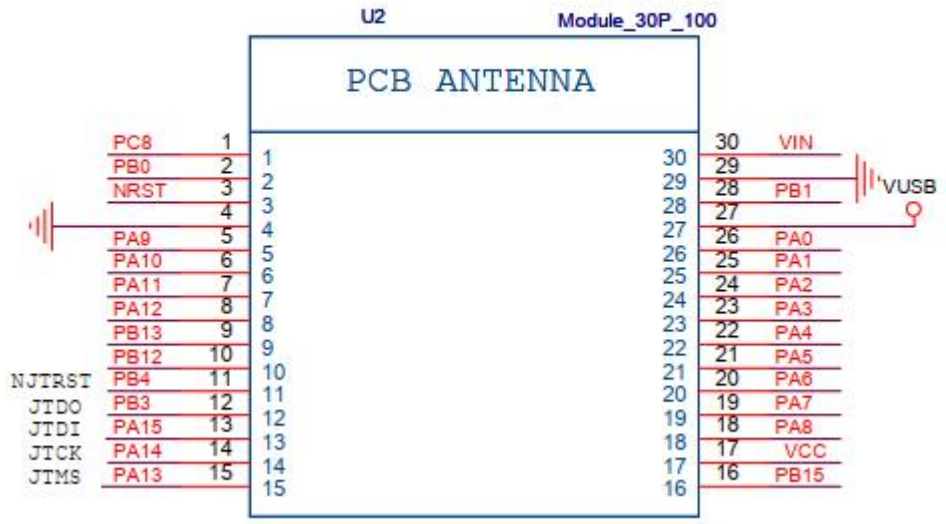
根据网上所说，需要 jlinkv10 及以上版本才可以烧录程序，而我手头上的 Jlinkv9 并没有引出 JTAG 引脚，所以无法进行验证，我使用的是 jlinkob ra4m2 调试器，支持 risk-v 内核调试，我的是在闲鱼上购买的，当然也可以自己制作，嘉立创开源平台上也有开源的电路板与固件。

后记：发现芯片支持两线 cJTAG 引脚烧录，与 arm swd 引脚相同于是使用 jflash 软件进行了测试，发现 jlinkv9 并不支持此接口，遂作罢，同时发现 jlinkob ra4m2 可以使用 cJTAG 引脚成功连接芯片；

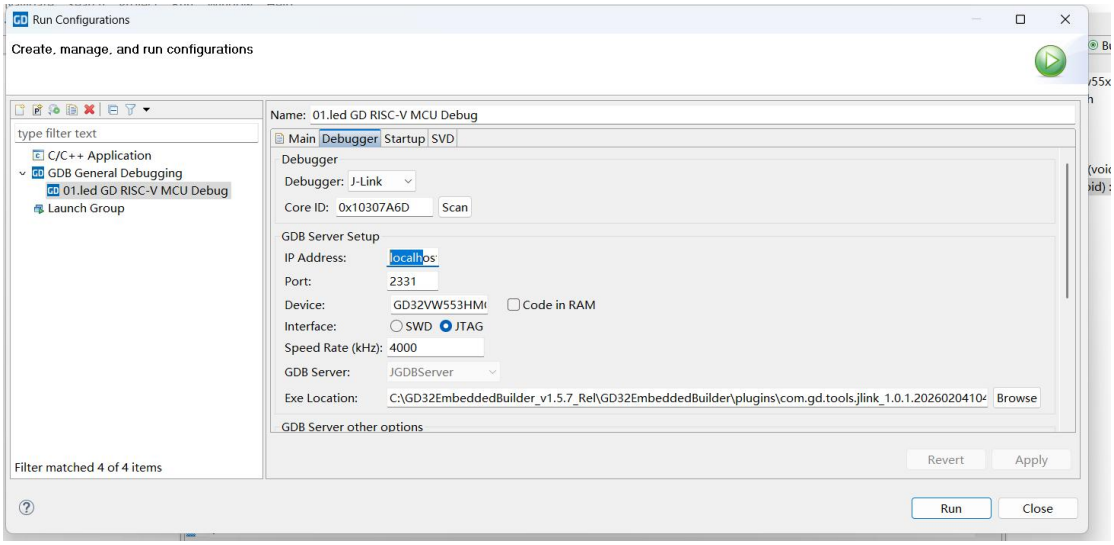
> 默认复位后使用五个引脚的 JTAG 调试，用户可以在不使用 NJTRST 引脚情况下正常使用 JTAG 功能，此时 PB4 可以用作普通 GPIO 功能（NJTRST 硬件拉高）。使用 cJTAG 调试时的引脚分配为：PA14: JTCK、PA13: JTMS，其他引脚（PA15、PB4、PB3）可用作普通 GPIO 功能。如果 JTAG 调试功能都没有使用，这五个引脚都释放作为普通 GPIO 功能。五个引脚具体配置请参考***通用和备用输入/输出接口（GPIO 和 AFIO）***。

```
Connecting ...
 - Connecting via USB to probe/ programmer device 0
 - Probe/ Programmer firmware: J-Link V9 compiled May  7 2021 16:26:12
 - Probe/ Programmer S/N: 69610359
 - ERROR: Debugger tries to select target interface cJTAG.
This interface is not supported by the connected emulator.
Selection will be ignored by the DLL.
```

V2 版本的开发板已经将所有 JTAG 引脚到排针上引出，并在原理图上进行了标注，非常便于我们开发



在 run configuration 中可以将调试器改为 jlink，即可扫描到对应的芯片 id 0x10307A6D,根据 AI 所说应该是芯片内部调试接口（TAP Controller）的 IDCODE。；



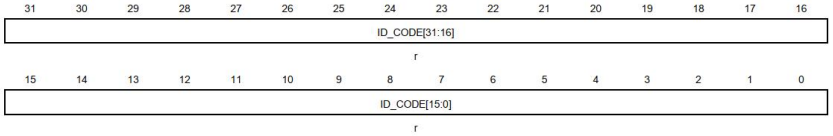
11.4. DBG 寄存器

DEBUG 基地址：0xE0044000

11.4.1. ID 寄存器（DBG_ID）

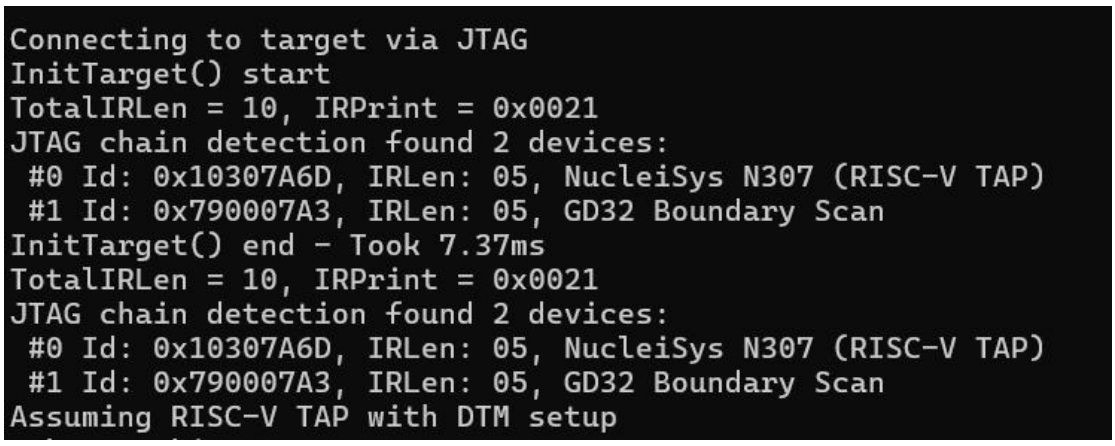
地址偏移：0x00
复位值：0x0000 0000，只读寄存器

该寄存器只能按字（32 位）访问



位/位域	名称	描述
31:0	ID_CODE[31:0]	DBG ID 寄存器 这些位由软件读取，这些位是不变的常数

同时我也进行了深入探究，使用 Jlink 命令行工具连接芯片，发现 AI 所说并不正确，ide 中扫描到的 ID 是来自芯来科技（Nuclei）的 N307 内核 ID，而第二个是用户手册提到的 GD32 的 JTAG id

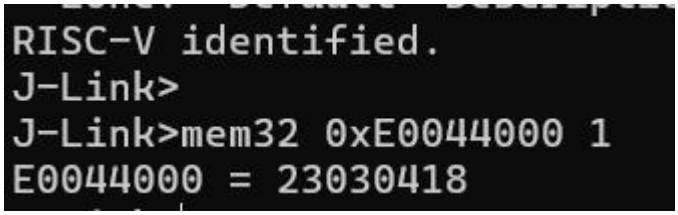


11.2.2. JTAG 链状结构

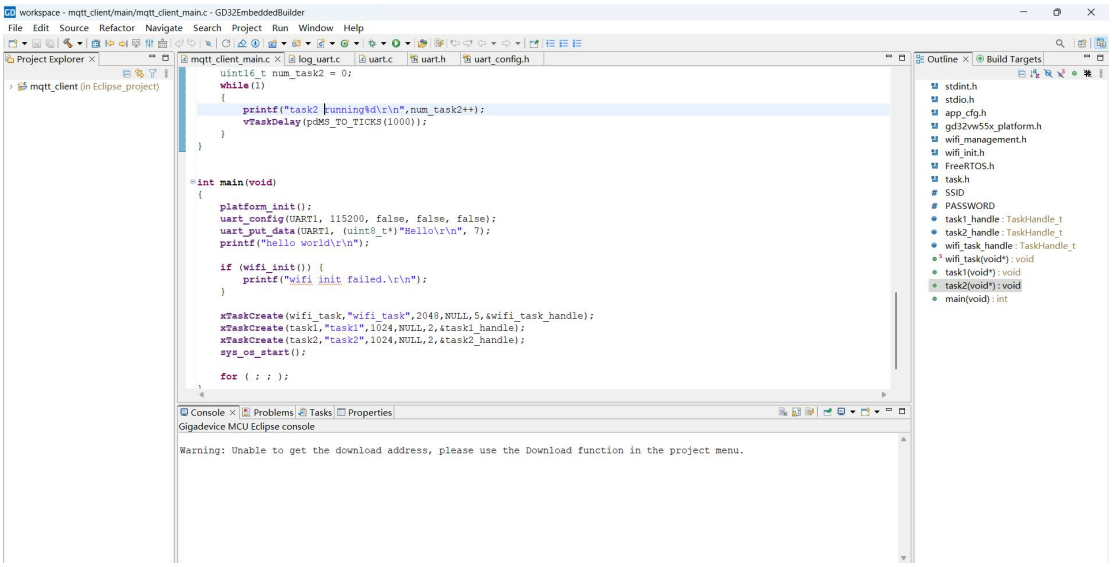
RISC-V 内核的 JTAG TAP 和边界扫描（BSD）TAP 串行连接。边界扫描（BSD）JTAG 的 IR（指令寄存器）是 5 位，RISC-V 内核的 JTAG 的 IR（指令寄存器）也是 5 位。

BSD JTAG ID 代码是 0x790007A3。

而 AI 所说的 IDCODE 寄存器读出来的是另一个值



在开发过程中，我发现 IDE 对 JLink 的兼容性并不是很好。最初我使用的是 iCEasy 商城资料中提供的 IDE 软件，但在尝试使用 JLink 进行烧录时，系统提示未找到目标地址，导致烧录失败。随后，我从 GD32 官网下载了最新版本的 IDE，新建的工程能够正常烧录。不过，在后续测试 Wi-Fi 功能时，当我导入 SDK 中的例程工程，再次遇到了无法烧录的问题，具体情况如下图所示。

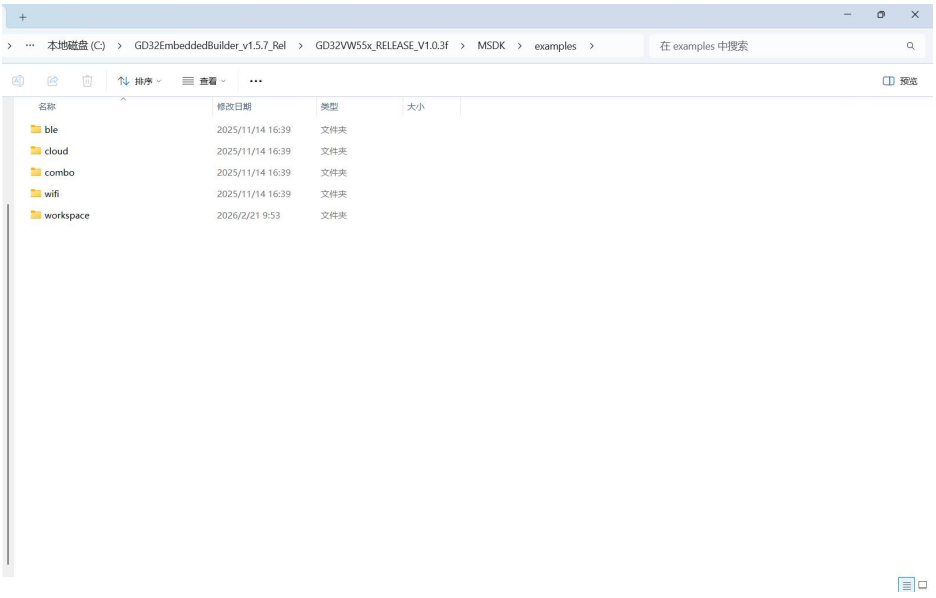


二，使用 WIFI SDK 进行 WIFI 连接测试

1.例程导入

在开发 Wi-Fi 功能时，由于从头新建网络程序需要手动导入大量库文件，较为繁琐，因此我选择直接导入官方 SDK 中的例程进行修改。同时，为了保留原始例程不被破坏，我采用了一个取巧的方法：将所需例程复制到与原始例程同一级目录下（即我新建的工程文件夹所在位置），这样工程中引用的相对路径不会失效，复制后的例程可以直接编译和使用。

如下图所示，原始例程分别存放在各个文件夹的下一级目录中，而我将它们复制出来后，放置在与这些例程同级的自定义文件夹内，从而在保留原例程结构的同时，可以自由修改并确保路径引用正确。



2.程序编写

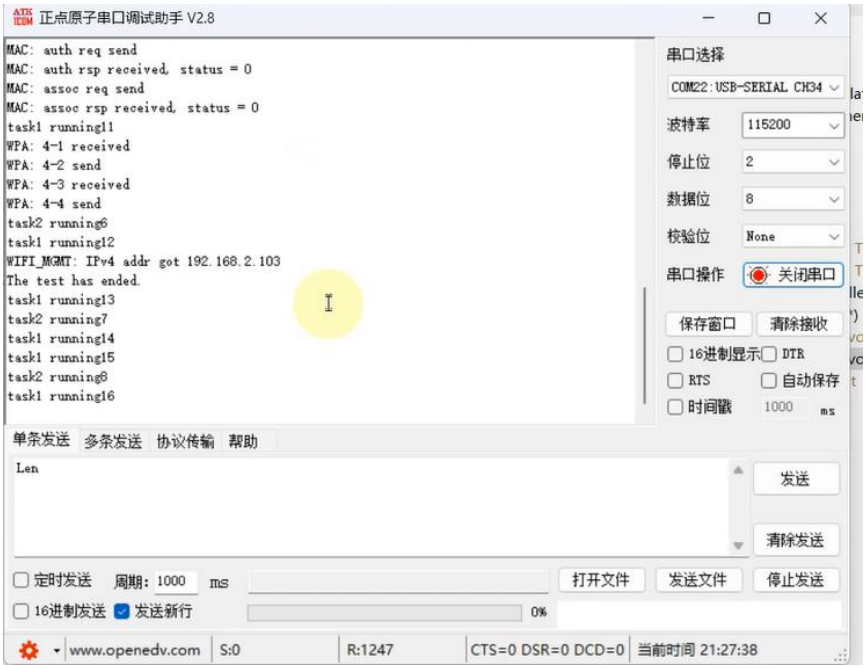
```
#include <stdint.h>
#include <stdio.h>
#include "app_cfg.h"
#include "gd32vw55x_platform.h"
#include "wifi_management.h"
```

```c
#include "wifi_init.h"
#include "FreeRTOS.h"
#include "task.h"


#define SSID "your ssid"
#define PASSWORD "your password"

TaskHandle_t task1_handle;
TaskHandle_t task2_handle;
TaskHandle_t wifi_task_handle;

static void wifi_task(void *param)
{
int status = 0;
char *ssid = SSID;
char *password = PASSWORD;
struct mac_scan_result candidate;

if (ssid == NULL) {
printf("ssid can not be NULL!\r\n");
goto exit;
}

/*
* 1. Start Wi-Fi scan
*/
printf("Start Wi-Fi scan.\r\n");
status = wifi_management_scan(1, ssid); //使用阻塞模式扫描是否存在 ssid 对应的 wifi
if (status != 0) {
printf("Wi-Fi scan failed.\r\n");
goto exit;
}
sys_memset(&candidate, 0, sizeof(struct mac_scan_result));
/*在已完成的扫描结果中查找与指定 ssid 匹配的 AP（接入点）。
* 这里 bssid 为 NULL，
* 因此仅根据 SSID 进行匹配。找到后将 AP 的详细信息（如 BSSID、信道、信号强度等）填充到 candidate 结构体中。*/
status = wifi_netlink_candidate_ap_find(WIFI_VIF_INDEX_DEFAULT, NULL,
ssid, &candidate);
if (status != 0) {
goto exit;
}
```

```
/*
* 2. Start Wi-Fi connection
*/
printf("Start Wi-Fi connection.\r\n");
if (wifi_management_connect(ssid, password, 1) != 0) {
printf("Wi-Fi connection failed\r\n");
goto exit;
}

exit:
printf("The test has ended.\r\n");
vTaskDelete(wifi_task_handle);
}


void task1(void* pvParameters)
{
uint16_t num_task1 = 0;
while(1)
{
printf("task1 running%d\r\n",num_task1++);
vTaskDelay(pdMS_TO_TICKS(500));
}
}
void task2(void* pvParameters)
{
uint16_t num_task2 = 0;
while(1)
{
printf("task2 running%d\r\n",num_task2++);
vTaskDelay(pdMS_TO_TICKS(1000));
}
}


int main(void)
{
platform_init();
uart_config(UART1, 115200, false, false, false);
uart_put_data(UART1, (uint8_t*)"Hello\r\n", 7);
printf("hello world\r\n");

if (wifi_init()) {
```

```
printf("wifi init failed.\r\n");
}

xTaskCreate(wifi_task,"wifi_task",2048,NULL,5,&wifi_task_handle);
xTaskCreate(task1,"task1",1024,NULL,2,&task1_handle);
xTaskCreate(task2,"task2",1024,NULL,2,&task2_handle);
sys_os_start();

for ( ; ; );
}
```

3.现象描述



可以发现已经成功连接到了 WIFI 网络，并且 freertos 的两个测试任务也在正常运行

4.遇到的串口问题说明

在开发过程中，我遇到了串口输出的问题。默认情况下，printf 函数的内容并不会被重定向到开发板的串口，因此需要手动修改相关代码才能输出调试信息。所幸，重定向的工作已经在 log_uart.c 文件中完成。借此机会，我也了解到了不同编译器下的实现差异：在 ARM Keil（即 ARMCC 编译器）中，需要重写 fputc 函数；而在 RISC- V 的 GNU 编译链中，则是通过实现 _write 函数来完成重定向。以下是 log_uart.c 中对应的实现代码：

```
#elif defined(__GNUC__)
int _write(int fd, char *str, int len)
{
    (void)fd;
    int32_t i = 0;

    /* Send string and return the number of characters written */
    while (i != len) {
        while (RESET == usart_flag_get(LOG_UART, USART_FLAG_TBE));
```

```
        usart_data_transmit(LOG_UART, *str);
        str++;
        i++;
    }

    while (RESET == usart_flag_get(LOG_UART, USART_FLAG_TC));

    return i;
}
```

要正常使用串口输出，我们需要在 `uart_config.h` 文件中正确配置 `LOG_UART` 的宏定义。例如，我的开发板上使用的是 UART1，因此需要将 `LOG_UART` 定义为 `USART1`。此外，我还遇到一个问题：串口始终没有输出信息。当时我修改了 `uart.h` 中关于串口引脚及引脚复用的定义，但不确定默认的配置是否与开发板匹配。因此，在使用前建议检查这两个文件中的相关设置，确保引脚分配和复用功能正确无误。