技术文件

技术文件名称：ESC0830 例程指导手册

版　　本：V3.1

目录

版本记录

Version#	Data	Updated by	Details of Change
1.0	2024.10.31	胡晶晶	环境配置、软件 demo 首次使用；
2.0	2024.11.07	胡晶晶	以库的形式提供拟态功能；
2.1	2024.12.09	胡晶晶，刘太昆	新增开发板手册
3.0	2024.12.21	胡晶晶，倪晓波，朱赤丹	新增外设例程、手册，新增调试功能，新增 RTOS
3.1	2025.07.11	胡晶晶	设计大赛精简版

1. 介绍

1.1. 目录结构

目录如下所示：

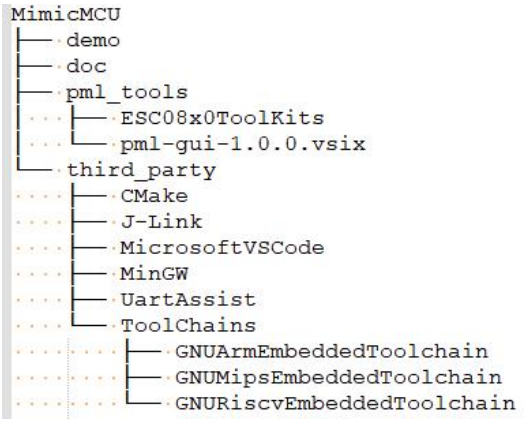

图 1. 目录结构示意图

其中:

- demo 为软件工程示例

内部文件夹可拷贝到任意位置使用；

- doc 为文档目录
- pml_tools 为紫金山实验室自研工具

由紫金山实验室提供的 ESC0830 芯片的编译插件和烧写工具；

- third_party 为三方库

是开发的必要环境，包括 cmake、vscode、mingw、J-link 驱动、串口工具和 3 个功能 cpu 的交叉编译工具链。

1.2. 责任申明

所有三方环境、工具推荐用户自行从各自官网下载、安装，为了更快帮助上手，压缩包中暂时提供安装包或安装程序，请 24 小时内自行删除。紫金山实验室仅持有 demo、pml_tools 和 doc 三个目录的所有权。

2. 安装

2.1. 解压

全文以解压路径 E:\MimicMCU 为例。

2.2. 环境变量

2.2.1. PMLPath

新建环境变量 PMLPath，变量值是解压后的 SDK 路径，本例为 E:\MimicMCU。

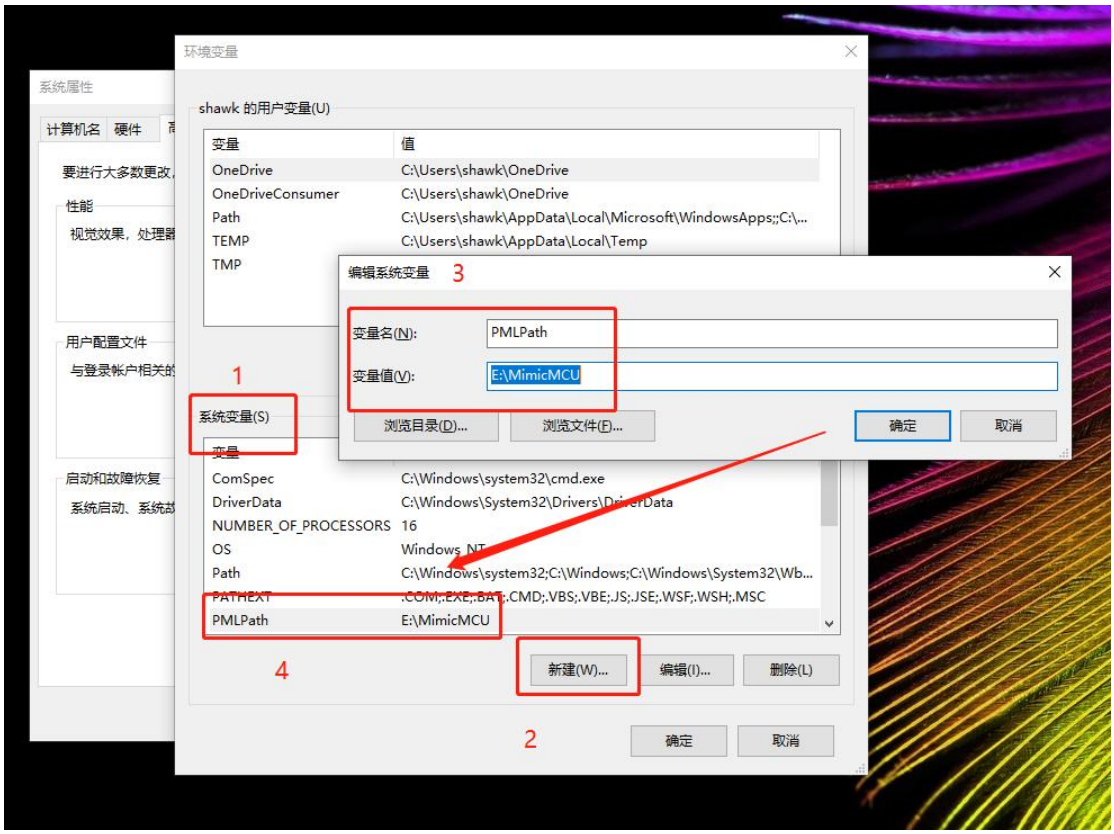

图 2. 新建 PMLPath 环境变量

2.2.2. Path

将 SDK 解压后的子目录中的 cmake 和 mingw 路径加入 Path，本例为：

```
%PMLPath%\third_party\MinGW\bin
```

%PMLPath%\third_party\CMake\bin

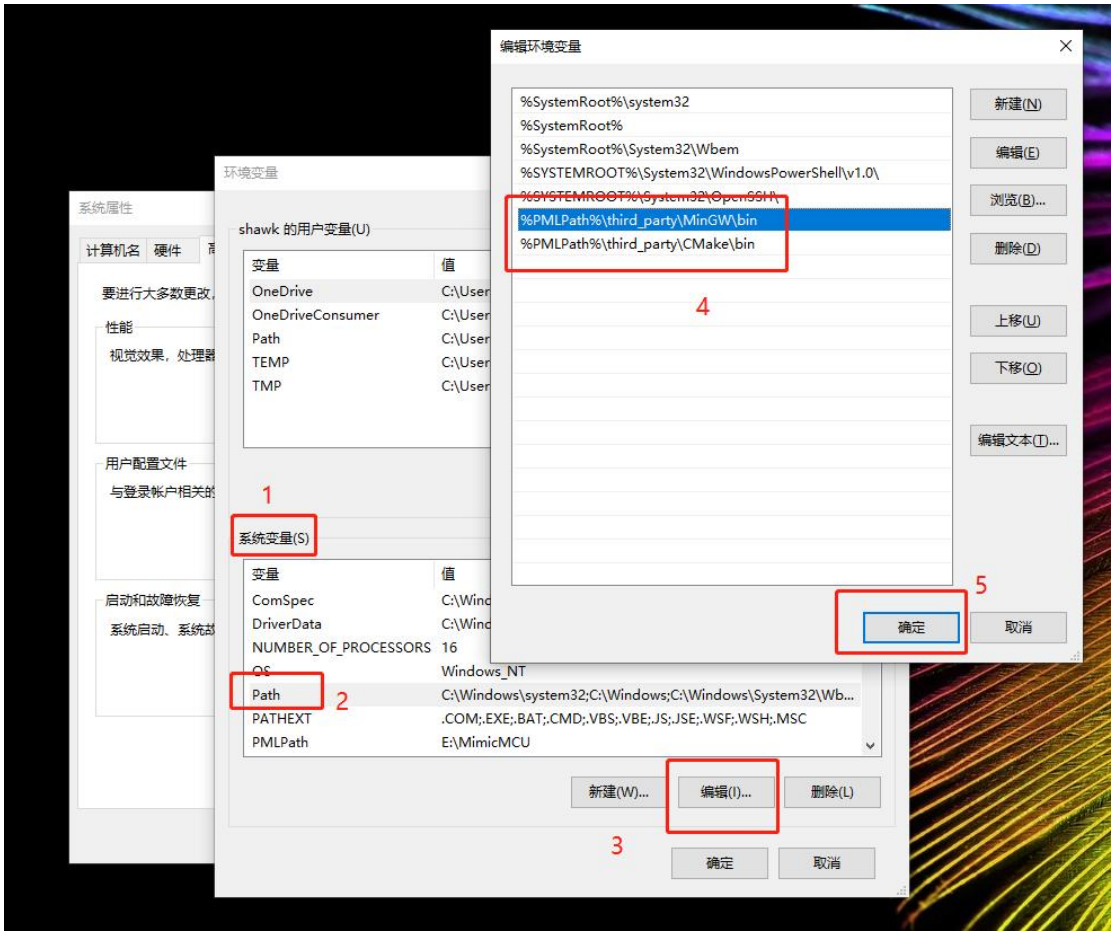

图 3. 编辑 PATH 环境变量

2.3. python

从 Microsoft Store 安装 python3，本文以 python3.9 为例。

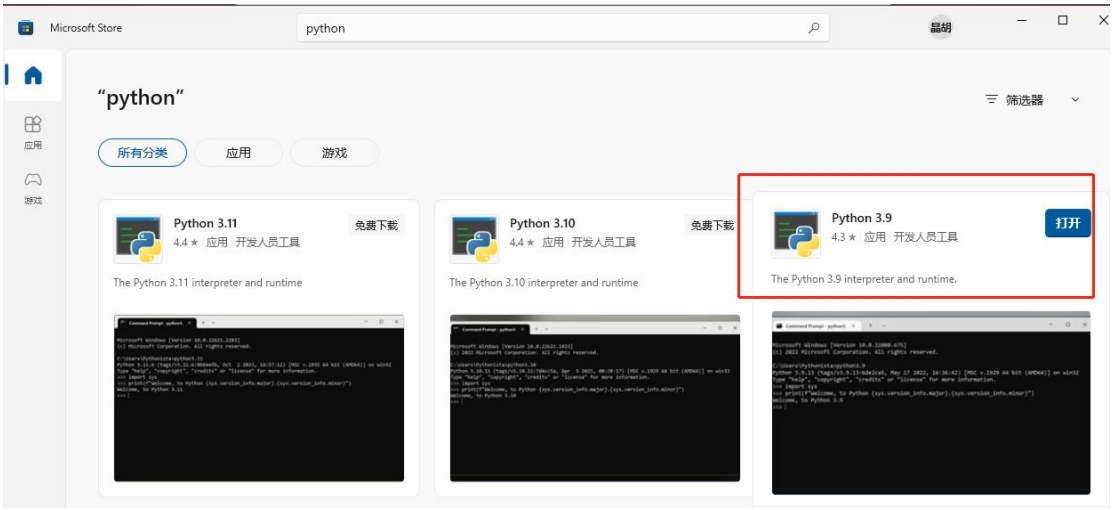

图 4. 安装 python3

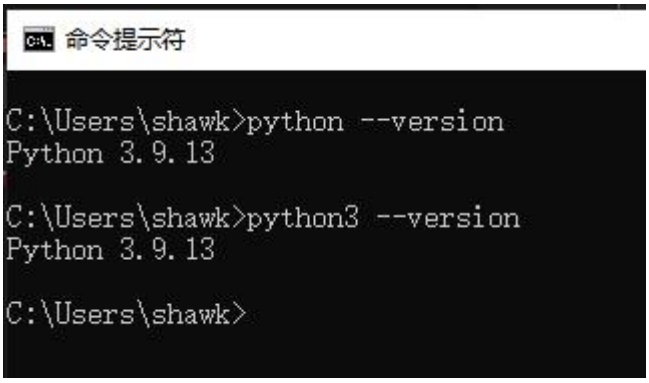

图 5. 检查 python3

注意：安装后确保本机可以在 cmd 使用 python3 命令（如本机同时安装了 python2，python 可能默认指向 python2，所以软件工程中已经做了适配，方案就是直接调用 python3 而非 python）

如果 cmd 中无法直接使用 python3 命令，则建议先找到 python.exe 位置，然后将其复制一份重命名为 python3.exe，再调用即可：

图 6. 确保能调用 python3

2.4. 配置 vscode

将 vscode（本文使用 1.61.2 版本）可执行文件固定到开始屏幕：

E:\MimicMCU\third_party\Microsoft VS Code\Code.exe

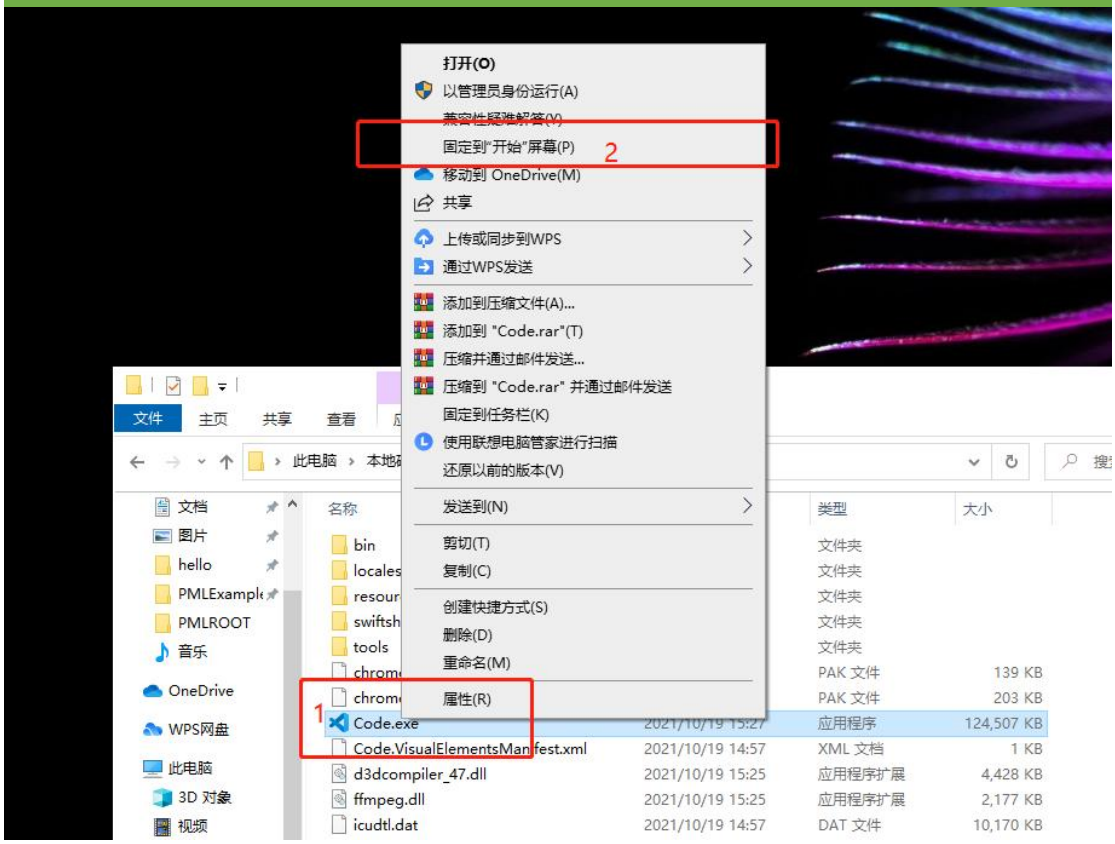

图 7. 固定 VSCode

注意：推荐使用 vscode1.61.2 版本（压缩包内已提供）、1.68.1 版本，已知 1.83.1 版本与本软件工程任务不兼容。

2.4.1. 文件自动保存

从开始打开 vscode 应用，设置文件自动保存（务必）：

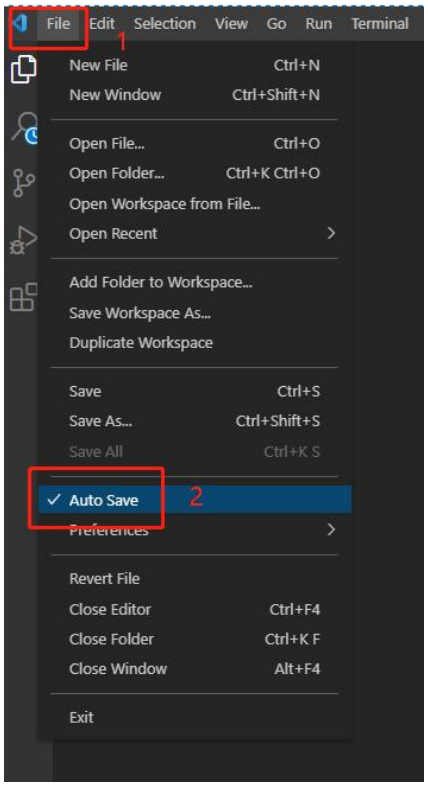

图 8. VSCode 文件自动保存

2.4.2. 使用 powershell

检查并确保 vscode 的默认 terminal 是 powershell（务必）；

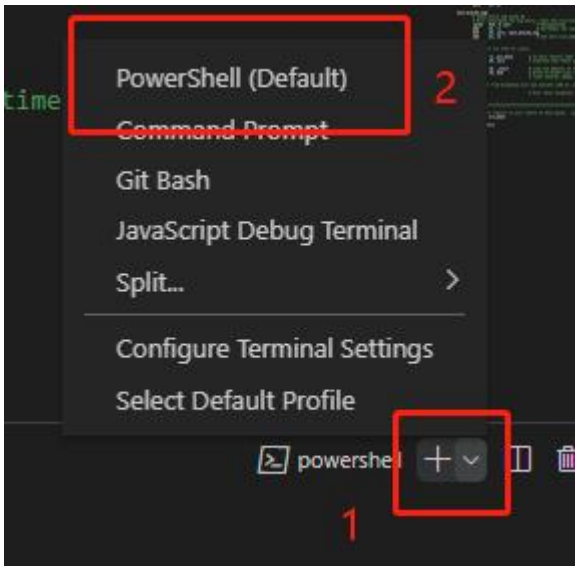

图 9. VSCode 缺省使用 powershell

在 vscode 中 Open Folder 打开软件工程文件夹。打开特定后缀文件名时，vscode 将会提醒安装插件，可按 vscode 推荐安装插件，优化代码阅读体验。也

可以不安装，不影响使用。例如，中文插件、cmake 语法高亮插件、GNU 链接脚本语法高亮插件等。

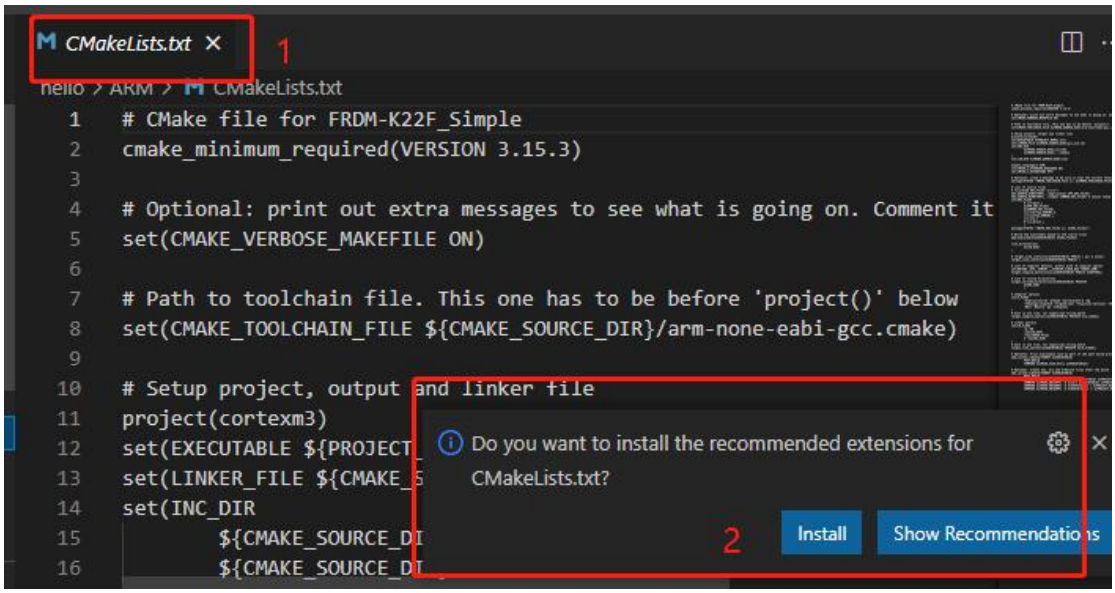

图 10. VSCode 自动推荐扩展

2.4.3. 安装插件

在 vscode 左边侧栏选择 Extensions，安装 SDK 中的自研插件：

```
E:\MimicMCU\pml_tools\pml-gui-1.0.0.vsix
```

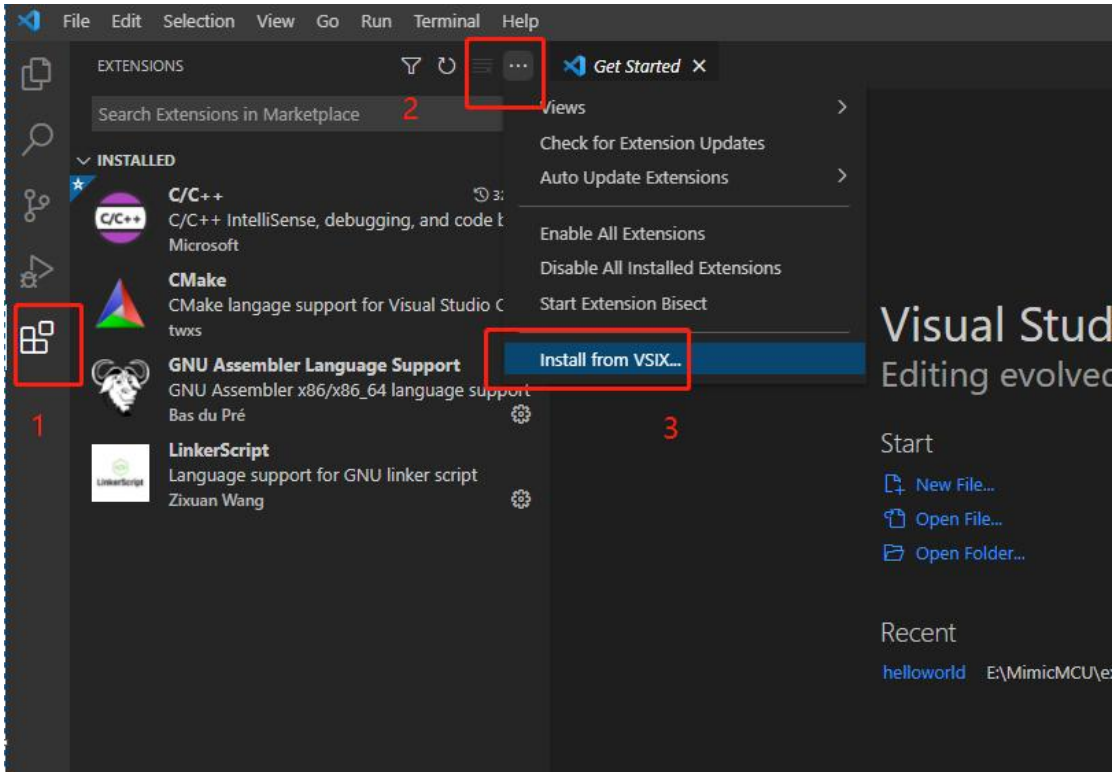

图 11. 安装自研插件

安装插件后，打开任意文件进入编辑器界面，右上角出现新增的 PML 图标，则插件安装成功；

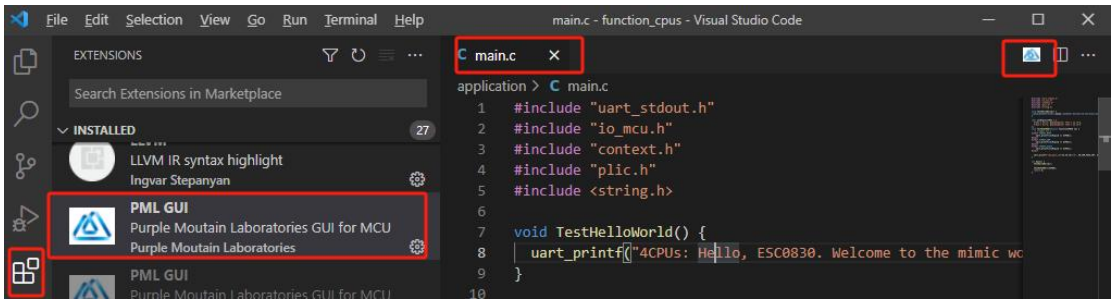

图 12. 自研插件图标

2.5. 安装 J-Link

调试功能使用 J-Link Base 调试器，需要安装响应驱动，例如压缩包中提供的 JLink_Windows_V692.exe。

3. 编译

3.1. 拷贝样例

将 demo 下的样例工程拷贝到任意目录，例如，从：

```
E:\MimicMCU\demo
```

拷贝到：

```
E:\Play\demo
```

3.2. 编译功能 CPU 程序

3.2.1. 编译方法

vscode 中打开 demo\function_cpus 文件夹。

打开任意源文件后，点击编辑界面，右上角出现 PML 插件图标。点击右上角 PML 图标，选择编译架构，全选后 ok，开始编译。

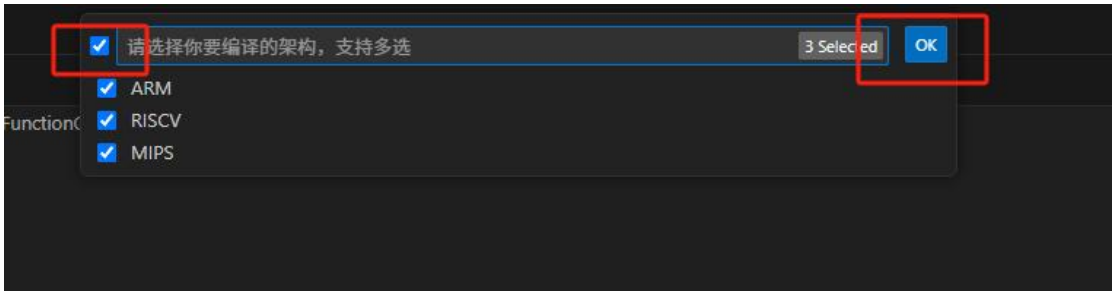

图 13. 功能 cpu 一键编译

3.2.2. 输出路径

生成文件在 demo\output，参考下图红框内的文件，下图黄框内的文件无需改动、已直接提供。

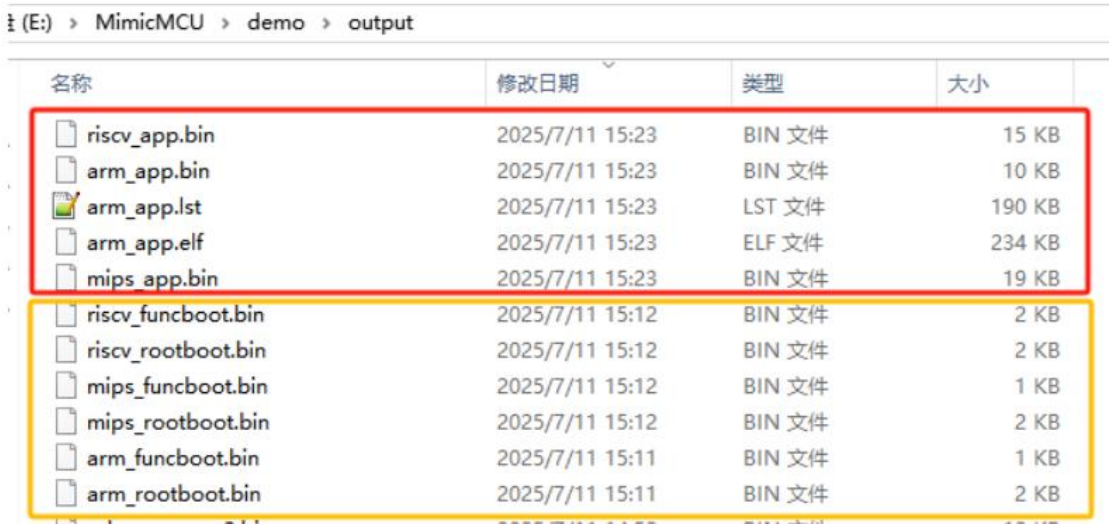

图 14. 功能 cpu 全编译输出路径

3.3. 熟悉调度 CPU 程序

调度 cpu 程序程序固化，demo\output 中提供以下文件供用户直接使用：

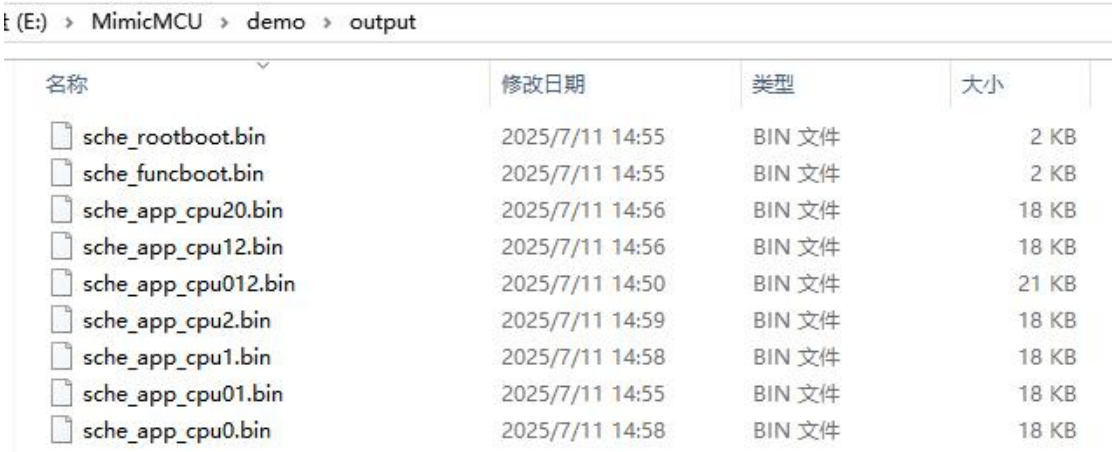

图 15. 调度 cpu 程序

其中 sche_rootboot.bin 和 sche_funcboot.bin 不变，sche_app_cpuxxx.bin 文件用于控制整个 MimicMCU 系统的运行模式，具体来说：

- _cpu0.bin 控制仅启用 mips 功能核，也即配置为 mips 单模模式；类似地，_cpu1.bin、_cpu2.bin 分别控制仅启用 arm 功能核、riscv 功能核；
- _cpu01.bin 控制仅启用 mips 功能核和 arm 功能核，也即配置为 mips+arm 的两模模式；类似地，_cpu12.bin 配置为 arm+riscv 的两模模式、_cpu20.bin 配置为 mips+riscv 的两模模式；
- _cpu012.bin 控制启用所有三个功能核，也即三模模式。

4. 烧写

4.1. 测试板

该描述适用 ESC0830EVBT0.1 开发板，使用方式参考如下文档：

ESC0830 EVB T0.1 开发板使用手册.pdf

ESC0830 EVB T0.1 开发板原理图.pdf

4.2. 烧写测试

前文已经在 output 文件夹准备好 12 个 bin 文件，测试板也已经连接到上位机，本章进行烧写、测试。

4.2.1. 烧写

测试板置于 Flash 烧写模式，打开测试板电源，点击 VSCode 右上角 PML 图标，选择 mcuBoot，弹出烧写工具。

点击工具-->软件配置，选择端口，烧写波特率为 230400：

图 16. 配置烧写波特率

连接设备，读寄存器，显示 c903b，说明烧写通道建立成功：

图 17. 建立烧写通道

按图配置基址（对于 ESC0830 芯片来说固定值如下图）、bin 文件，选中所有核，依次点击“Flash 配置”（对每颗芯片配置一次即可）、“一键烧写”：

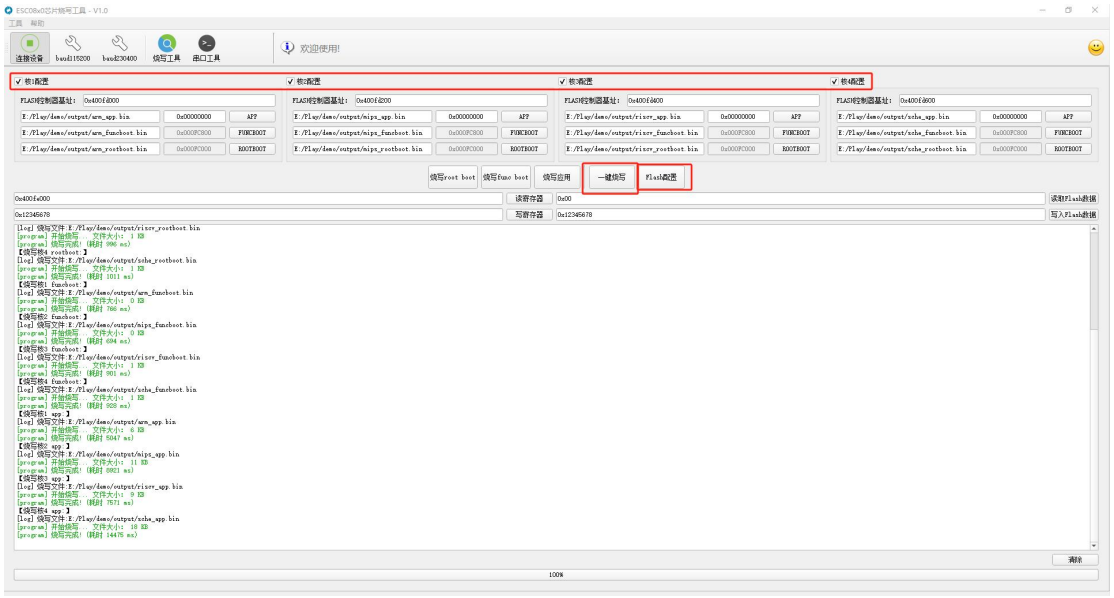

图 18. 一键烧写

4.2.2. 运行

示例中涉及串口 UART0 任务信息和 UART2 告警信息的打印，波特率设置均为 115200bps。测试板置于正常运行模式，连接串口、复位测试板，观察串口结果。

观察串口 UART0：点击烧录工具内置的“串口工具”。

图 19. 串口 UART0 结果

观察串口 UART2：外接 USB 转串口工具，Tx 连接开发板 PD6 引脚、Rx 连接开发板 PD7 引脚，如下图 20 所示。使用 UartAssist V5.0.10.exe 配置并连接。

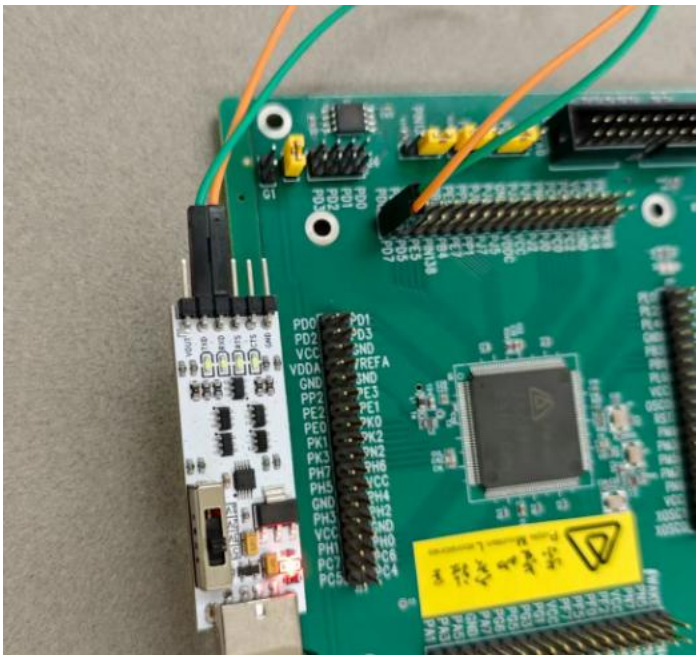

图 20. 串口 UART2 连接

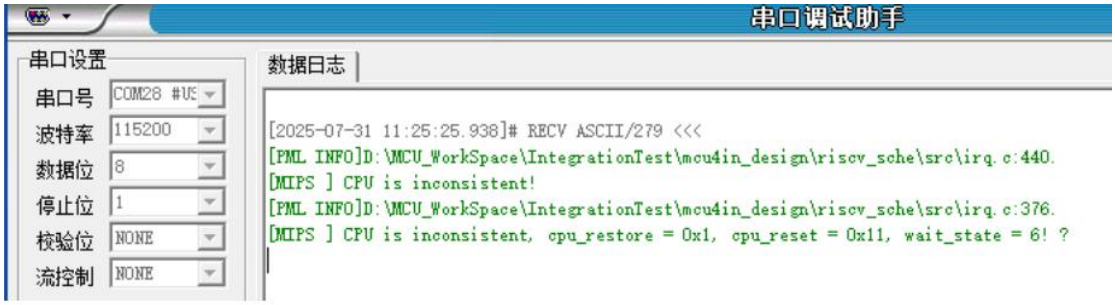

图 21. 串口 UART2 结果

4.2.3. 攻击原理

main.c | work.c | uart_print.h | io_mcu.h | chip.cfg

application > work > work.c > DemoAttack(void)

```c
#define MAX_CALLBACKS 4
callback_t callback_table[MAX_CALLBACKS];
volatile callback_t emergency_stop_callback = 0;

void register_callback(int index, callback_t cb) { callback_table[index] = cb; }

char incMsg[2][100] = {
  {"The battery voltage exceeds the permissible threshold! Stop it now!!!\n"},
  {"The battery voltage is normal.\n"}
};
void legitimate_callback() { pml_print(incMsg[0]); }

void malicious_payload() { pml_print(incMsg[1]); }

void set_emergency_stop_callback(callback_t cb) { emergency_stop_callback = cb; }

#define MAX_SAFE_VOLTAGE (420)  // GB/T 18487.1-2015
unsigned read_battery_voltage() { return 500; }

const FunctionCPUID id = kCPU2;
void DemoAttack(void) {
  set_emergency_stop_callback(legitimate_callback);

  // exploiting callback register vulnerability
  unsigned malicious_addr = (unsigned)malicious_payload;
  int byte_offset = (char*)&emergency_stop_callback - (char*)callback_table;
  int exploit_index = byte_offset / sizeof(callback_t);
#ifdef TARGET_MIPS
  if (id == kCPU0) {
    register_callback(exploit_index, (callback_t)malicious_addr);
  }
#endif
#ifdef TARGET_ARM···
#endif
#ifdef TARGET_RISCV···
#endif

  // trigger 2vs.1
  unsigned voltage = read_battery_voltage();
  if (voltage > MAX_SAFE_VOLTAGE) {
    emergency_stop_callback();
  }
}
```

图 22. 攻击示例源码

如上图 22，demo\function_cpus\application\work 文件夹内的所有源文件都将被编译、所有头文件都可以被包含，参赛队伍必须实现 demo\function_cpus\application\work.h 中申明的 void DemoAttack(void);

示例中在 demo\function_cpus\application\work\work.c 文件中实现的 DemoAttack()利用数组索引漏洞，将数组地址附近的一个函数地址覆盖了，使得调用 emergency_stop_callback()时未受攻击的核执行 legitimate_callback()、受攻击的核执行 malicious_payload()。

根据 UART0 的结果，系统整体仍然按照 legitimate_callback()执行任务，单核攻击未能传递到三模状态的系统外部。

根据 UART2 的结果，系统检测并清洗了出错的核，并发出告警信息。

注意，图 19 中仅红框以内信息受参赛队伍控制，红框以外信息自动生成，用于自动判定系统状态。

4.3. 单步调试 ARM 单核

需自备调试探针，例如 J-Link BASE（见 www.segger.com）。

1. 烧写仅使能 ARM 单核的调度核程序，应用层为 sche_app_cpu1.bin。
2. 烧写 ARM 功能核程序：
3. 置测试板为运行模式，以 115200 波特率打开串口；
4. 将 J-link base 调试器连接测试板，打开 J-Link GDB Server，测试板置于正常运行模式，Target device 选择 Cortex-M3，连接 ARM 单核：

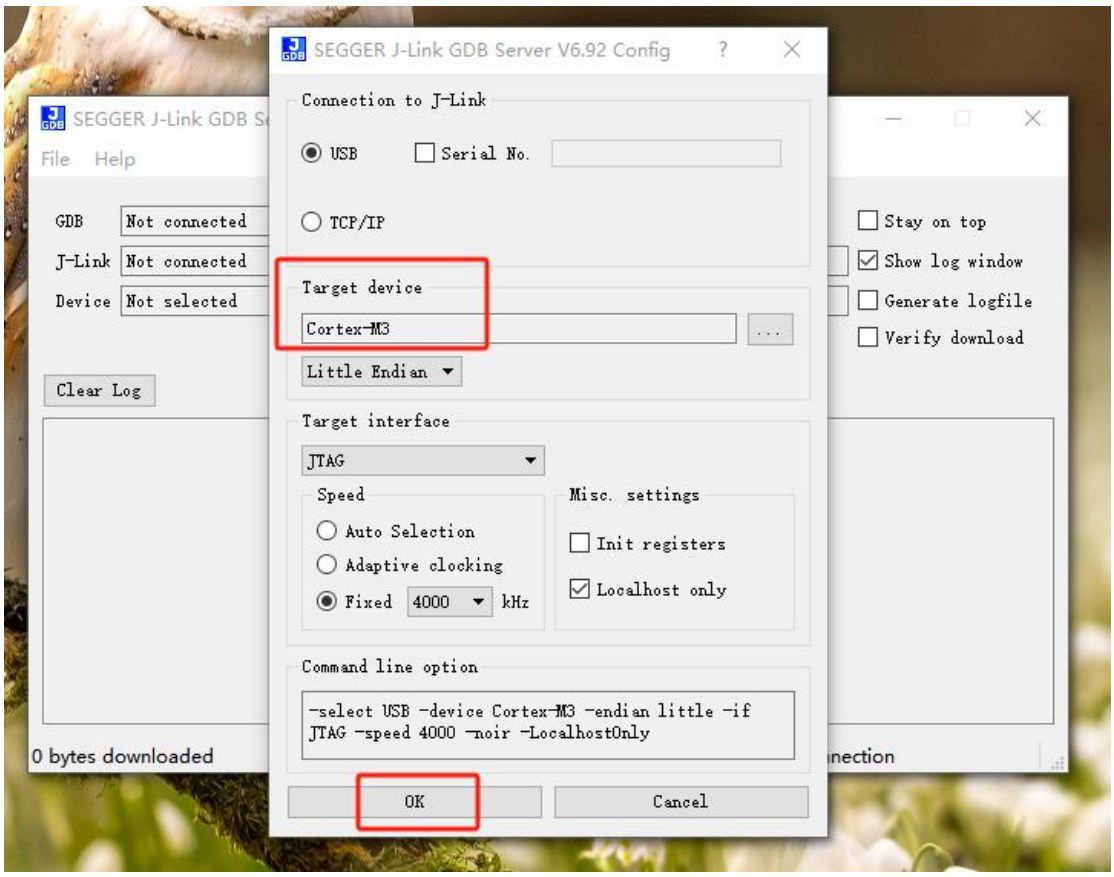

图 23. 连接 ARM 单核

打开 windows 自带 cmd 终端，运行以下命令开始单步调试:

```
xxgdb.exe xx.elf
tar ext:2331
load
b main
c
s
```

调试过程参考视频：ESC0830 单步调试 arm 录屏.wmv