Technical Documentation

ESC0830 Example Program Guide Manual

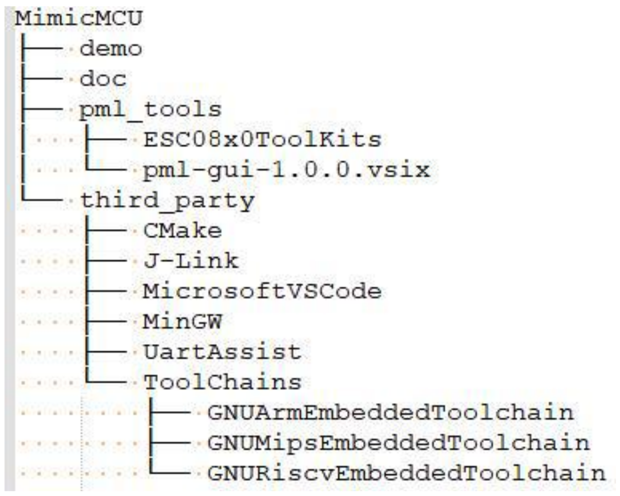
Version: V3.1

Table of Contents

1.Introduction

1.1. Directory Structure

The directory is as follows:



```
MimicMCU
├── demo
├── doc
├── pml_tools
│   ├── ESC08x0ToolKits
│   └── pml-gui-1.0.0.vsix
└── third_party
    ├── CMake
    ├── J-Link
    ├── MicrosoftVSCode
    ├── MinGW
    ├── UartAssist
    └── ToolChains
        ├── GNUArmEmbeddedToolchain
        ├── GNUMipsEmbeddedToolchain
        └── GNURiscvEmbeddedToolchain
```

Figure 1 Schematic Diagram of Directory Structure

Among them:

- **demo** contains example software projects.

  The internal folders can be copied to any location for use.
- **doc** is the documentation directory.
- **pml_tools** consists of self-developed tools by Purple Mountain Laboratories.

  These include compilation plugins and flashing tools for the ESC0830 chip provided by Purple Mountain Laboratories.
- **third_party** contains third-party libraries.

  These are the necessary development environment components, including CMake, VSCode, MinGW, J-Link drivers, serial port tools, and cross-compilation toolchains for the 3 functional CPUs.

1.2. Liability Statement

All third-party environments and tools are recommended to be downloaded and installed by users from their respective official websites. To facilitate a quicker start, installation packages or installers are temporarily provided within the archive. Please delete them within 24 hours.

Purple Mountain Laboratories holds ownership only of the following three directories: demo, pml_tools, and doc.

2.Installation

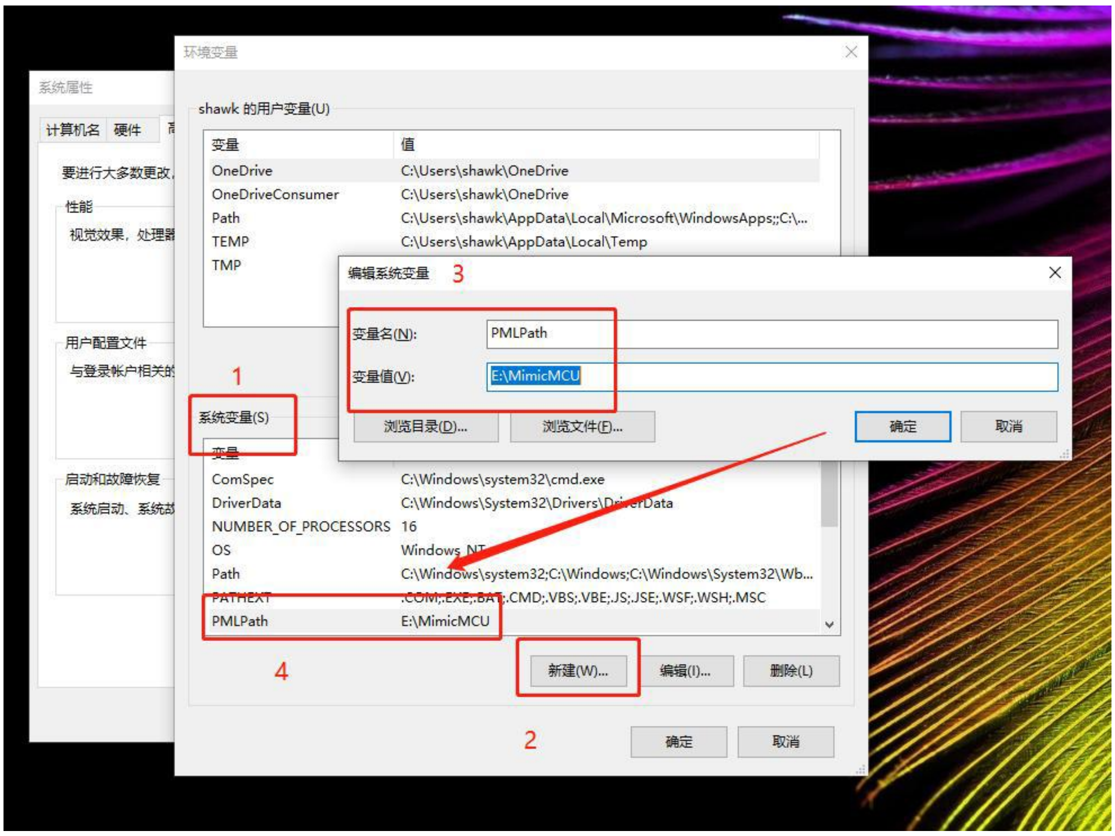
2.1. Unpacking

The entire document uses the unpacked path E:\MimicMCU as an example.

2.2. Environment Variables

2.2.1. PMLPath

Create a new environment variable named PMLPath, with its value set to the path of the unpacked SDK. In this example, it is E:\MimicMCU.



shawk 的用户变量(U)

变量	值
OneDrive	C:\Users\shawk\OneDrive
OneDriveConsumer	C:\Users\shawk\OneDrive
Path	C:\Users\shawk\AppData\Local\Microsoft\WindowsApps;;C:\...
TEMP	C:\Users\shawk\AppData\Local\Temp
TMP	

编辑系统变量

变量名(N): PMLPath

变量值(V): E:\MimicMCU

系统变量(S)

变量	值
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	16
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wb...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PMLPath	E:\MimicMCU

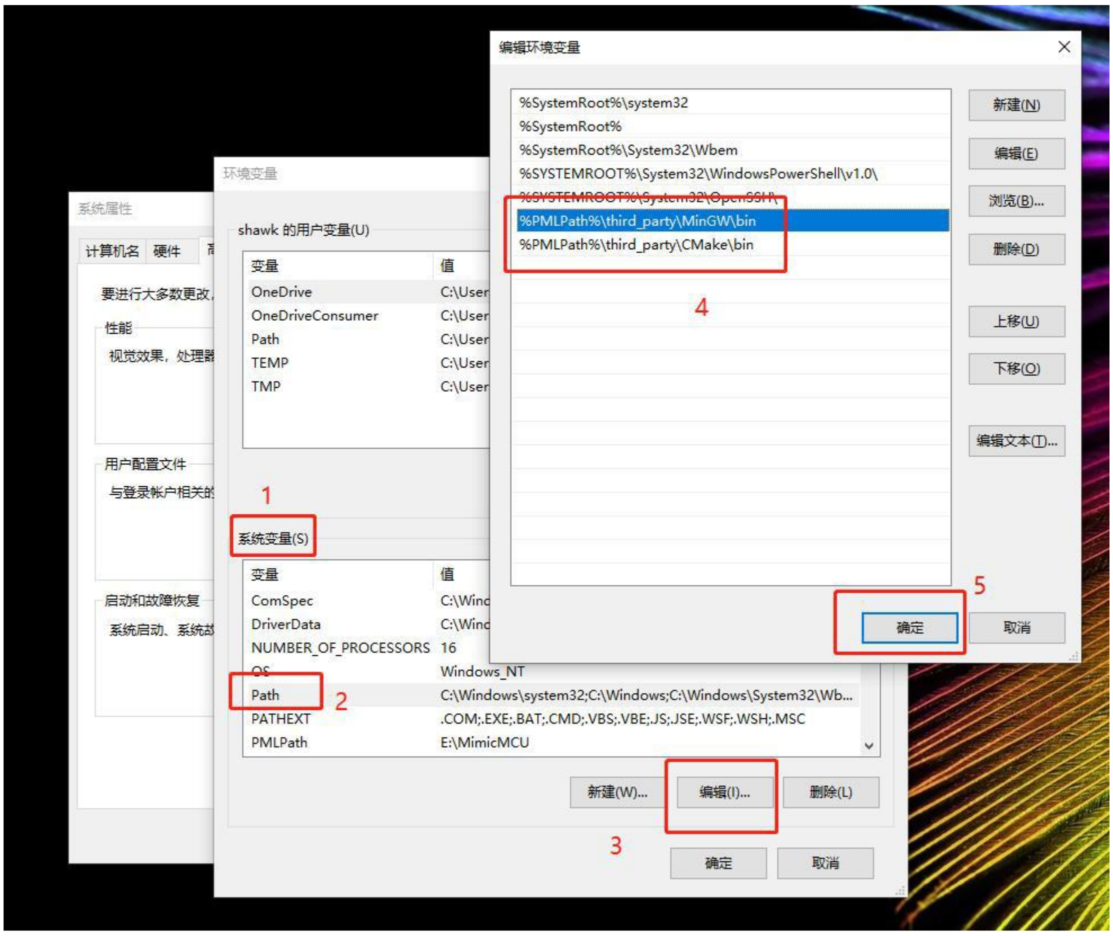
Figure 2 Create a new PMLPath environment variable.

2.2.2. Path

Add the paths for cmake and mingw from the unpacked SDK subdirectories to the Path variable. In this example, they are:

%PMLPath%\third_party\MinGW\bin

%PMLPath%\third_party\CMake\bin



%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
%PMLPath%\third_party\MinGW\bin
%PMLPath%\third_party\CMake\bin

变量	值
OneDrive	C:\User
OneDriveConsumer	C:\User
Path	C:\User
TEMP	C:\User
TMP	C:\User

变量	值
ComSpec	C:\Wind
DriverData	C:\Wind
NUMBER_OF_PROCESSORS	16
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wb...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PMLPath	E:\MimicMCU

Figure 3 Edit the PATH environment variable.

2.3. Python

Install Python 3 from the Microsoft Store. This document uses Python 3.9 as an example.

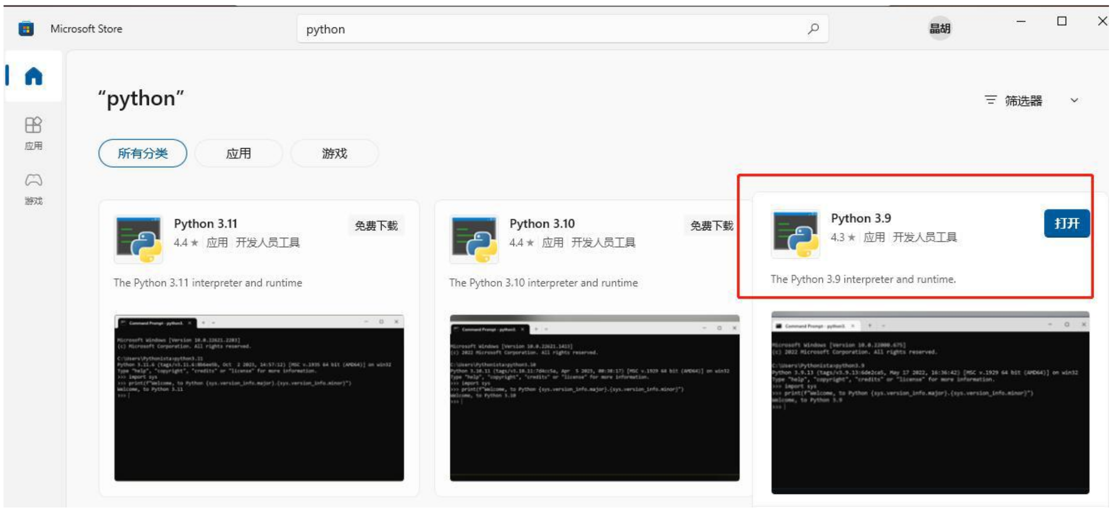
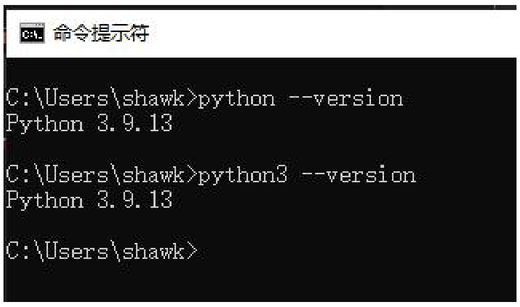


Figure 4 Install Python 3.



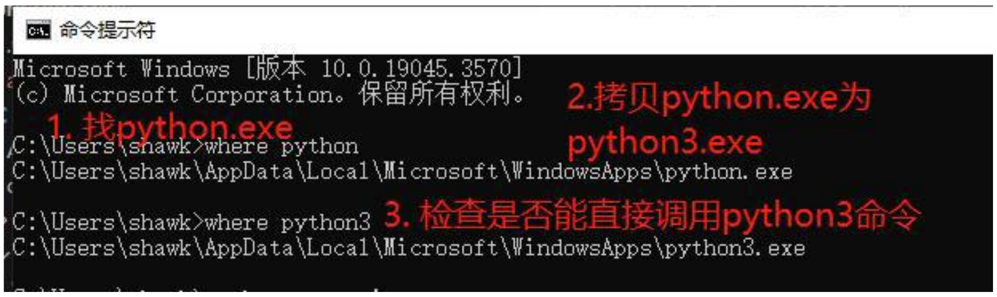
命令提示符

```
C:\Users\shawk>python --version
Python 3.9.13

C:\Users\shawk>python3 --version
Python 3.9.13

C:\Users\shawk>
```

Figure 5 Check Python 3.

Note: After installation, ensure the python3 command is available in the command prompt (cmd) on your computer. (If Python 2 is also installed, the python command may default to Python 2. Therefore, the software project has already been adapted to directly call python3 instead of python.)

If the python3 command is not directly available in cmd, it is recommended to first locate the python.exe file. Then, make a copy of it, rename the copy to python3.exe, and it can be called thereafter.



命令提示符

```
Microsoft Windows [版本 10.0.19045.3570]
(c) Microsoft Corporation。保留所有权利。

C:\Users\shawk>where python
C:\Users\shawk\AppData\Local\Microsoft\WindowsApps\python.exe

C:\Users\shawk>where python3
C:\Users\shawk\AppData\Local\Microsoft\WindowsApps\python3.exe
```

Figure 6 Ensure that python3 can be invoked.

2.4. Configure VSCode

Pin the VSCode executable (this document uses version 1.61.2) to the Start screen:

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

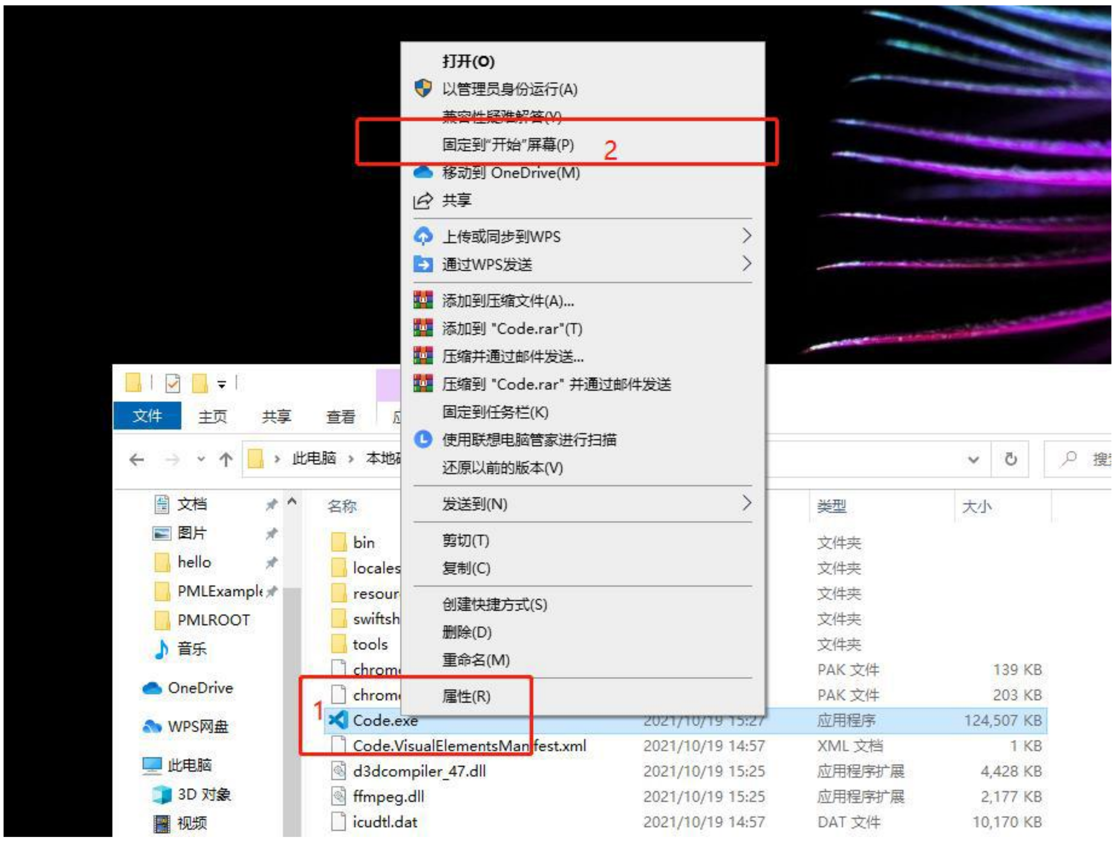


Figure 7 Pin Visual Studio Code

Right-click and select “Pin to Start screen”.

Note: It is recommended to use VSCode version 1.61.2 (already provided in the compressed package) or version 1.68.1. Version 1.83.1 is known to be incompatible with this software project task.

2.4.1. File Auto Save

Open the VSCode application from the Start menu and enable the file auto-save feature (this is essential).

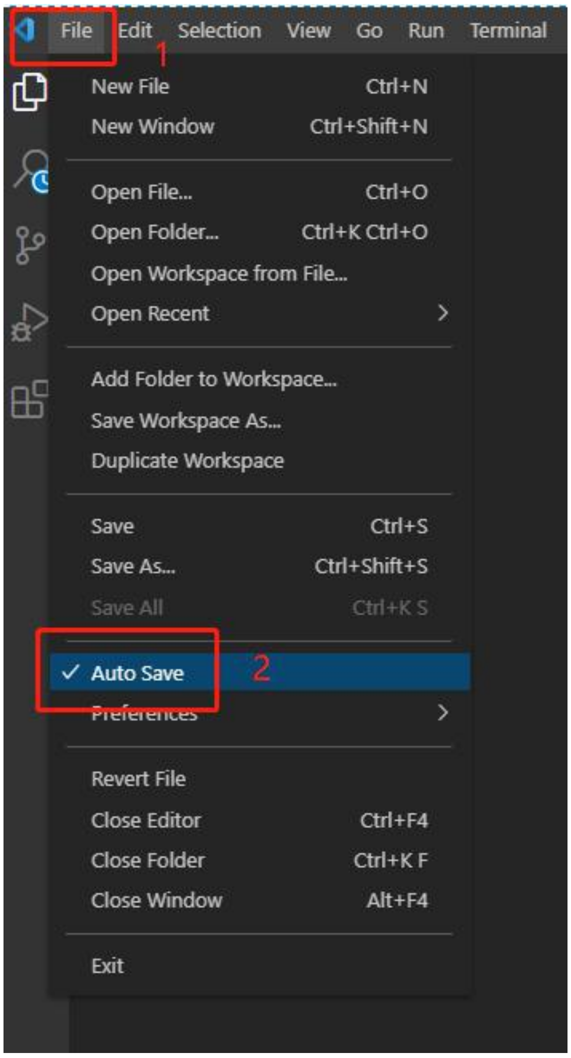


Figure 8 VSCode File Auto Save

2.4.2. Use PowerShell

Check and ensure that the default terminal in VSCode is set to PowerShell (this is essential).

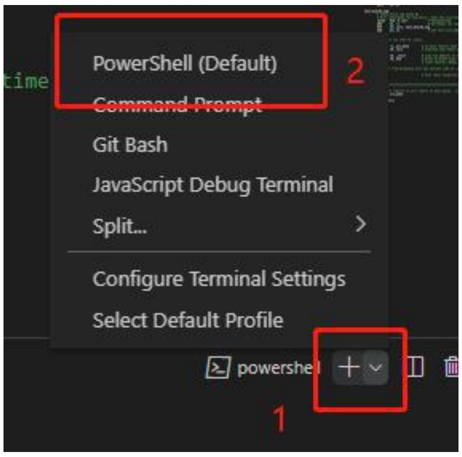
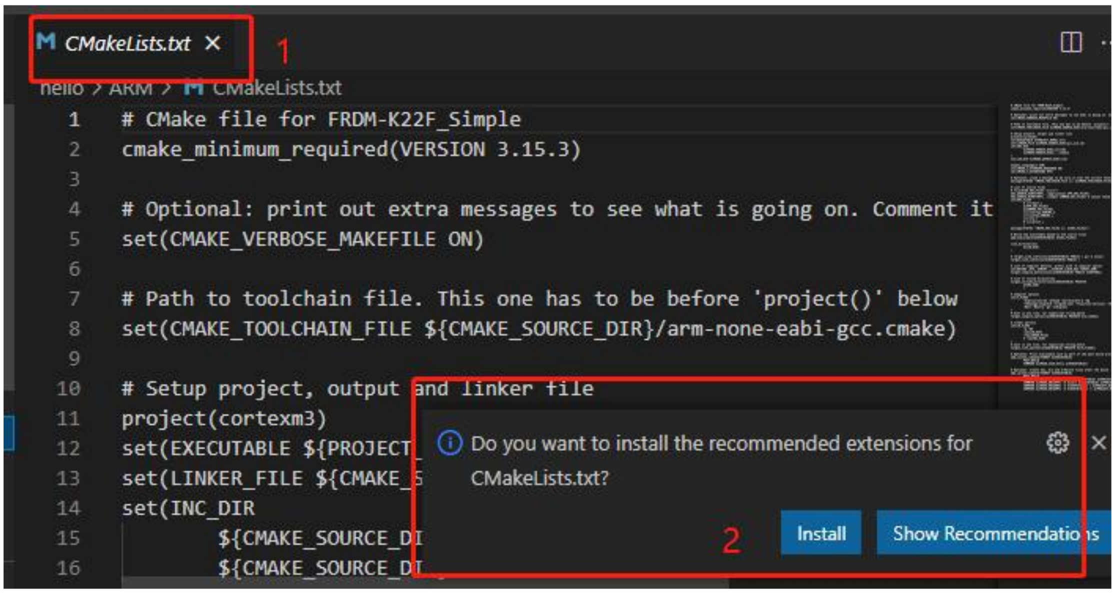


Figure 9 VSCode defaults to using PowerShell.

In VSCode, use "Open Folder" to open the software project folder. When opening files with specific extensions, VSCode may prompt you to install plugins. You can install the recommended plugins to enhance the code reading experience, or skip them without affecting functionality. Examples include the Chinese language pack, CMake syntax highlighting, and GNU linker script syntax highlighting plugins.



```
# CMake file for FRDM-K22F_Simple
cmake_minimum_required(VERSION 3.15.3)

# Optional: print out extra messages to see what is going on. Comment it
set(CMAKE_VERBOSE_MAKEFILE ON)

# Path to toolchain file. This one has to be before 'project()' below
set(CMAKE_TOOLCHAIN_FILE ${CMAKE_SOURCE_DIR}/arm-none-eabi-gcc.cmake)

# Setup project, output and linker file
project(cortexm3)
set(EXECUTABLE ${PROJECT_
set(LINKER_FILE ${CMAKE_S
set(INC_DIR
        ${CMAKE_SOURCE_DI
        ${CMAKE_SOURCE_DI
```

Do you want to install the recommended extensions for CMakeLists.txt?

Figure 10 VSCode Automatically Recommends Extensions

2.4.3. Install Plugins

In VSCode's left sidebar, select "Extensions" and install the self-developed plugins from the SDK:

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

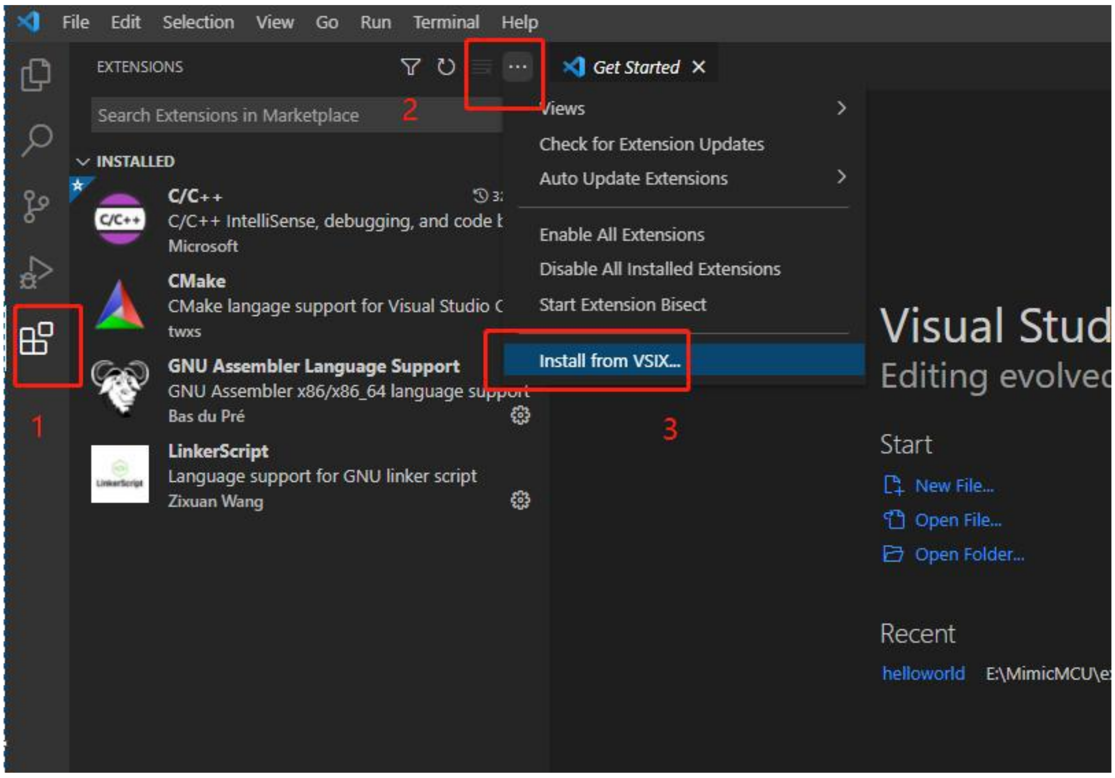
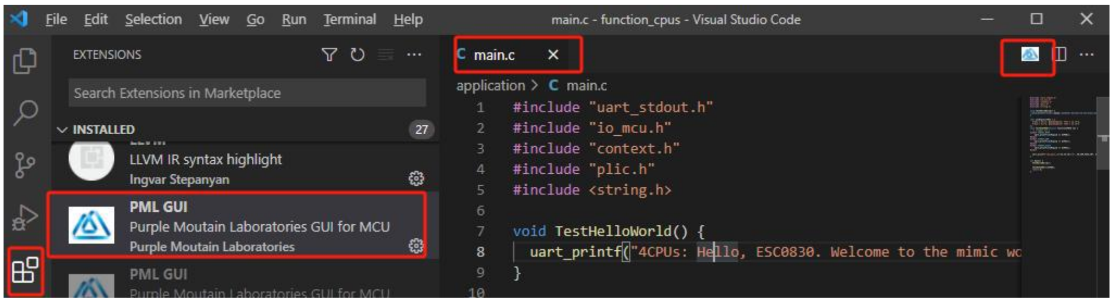


Figure 11 Install Self-developed Plugins

After installing the plugin, open any file to enter the editor interface. If a new PML icon appears in the upper right corner, the plugin installation was successful.


File
Edit
Selection
View
Go
Run
Terminal
Help
main.c - function_cpus - Visual Studio Code
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
27
LLVM IR syntax highlight
Ingvar Stepanyan
PML GUI
Purple Moutain Laboratories GUI for MCU
Purple Moutain Laboratories
PML GUI
Purple Moutain Laboratories GUI for MCU
main.c
application > C main.c

```c
#include "uart_stdout.h"
#include "io_mcu.h"
#include "context.h"
#include "plic.h"
#include <string.h>

void TestHelloWorld() {
  uart_printf("4CPUs: Hello, ESC0830. Welcome to the mimic wo
}
```

Figure 12 Self-developed Plugin Icon

2.5. Install J-Link

The debugging function requires the use of a J-Link Base debugger. The corresponding driver, such as JLink_Windows_V692.exe provided in the compressed package, needs to be installed.

3.Compilation

3.1. Copy the Sample Project

Copy the sample project from the demo directory to any desired location. For example, copy from:

E:\MimicMCU\demo

Copy to:

E:\Play\demo

3.2. Compile Functional CPU Programs

3.2.1. Compilation Method

Open the demo\function_cpus folder in VSCode.

After opening any source file, click on the editor interface. A PML plugin icon will appear in the upper right corner. Click the PML icon, select the compilation architecture, check all options, click OK, and begin the compilation.

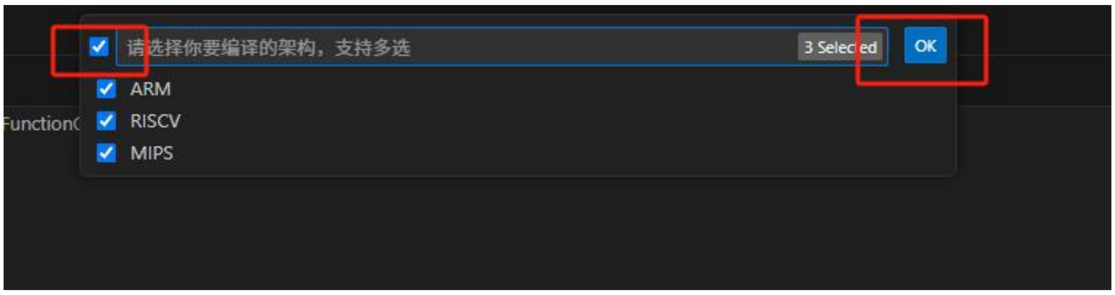
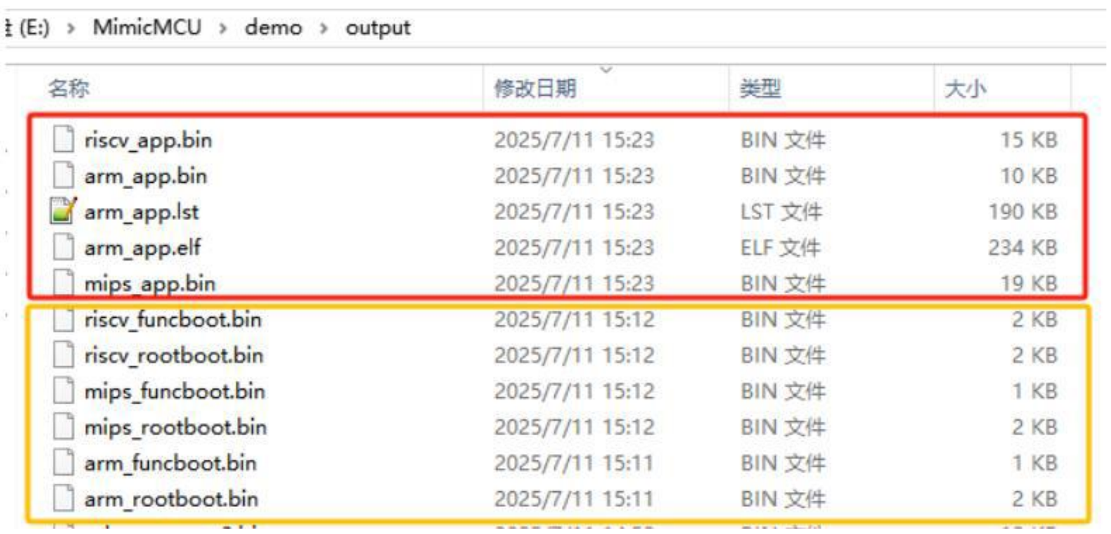


Figure 13 One-click compilation for functional CPUs.

3.2.2. Output Path

The generated files are located in demo\output. Refer to the files within the red box in the image below. The files within the yellow box do not need to be modified and are provided directly.



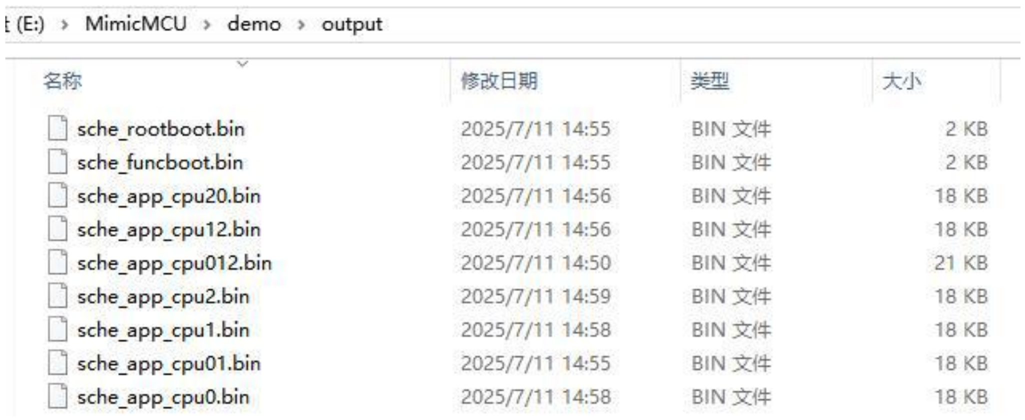
(E:) › MimicMCU › demo › output

名称	修改日期	类型	大小
riscv_app.bin	2025/7/11 15:23	BIN 文件	15 KB
arm_app.bin	2025/7/11 15:23	BIN 文件	10 KB
arm_app.lst	2025/7/11 15:23	LST 文件	190 KB
arm_app.elf	2025/7/11 15:23	ELF 文件	234 KB
mips_app.bin	2025/7/11 15:23	BIN 文件	19 KB
riscv_funcboot.bin	2025/7/11 15:12	BIN 文件	2 KB
riscv_rootboot.bin	2025/7/11 15:12	BIN 文件	2 KB
mips_funcboot.bin	2025/7/11 15:12	BIN 文件	1 KB
mips_rootboot.bin	2025/7/11 15:12	BIN 文件	2 KB
arm_funcboot.bin	2025/7/11 15:11	BIN 文件	1 KB
arm_rootboot.bin	2025/7/11 15:11	BIN 文件	2 KB

Figure 14 Full Compilation Output Path for Functional CPUs

3.3. Familiarize with the Scheduler CPU Program

The scheduler CPU program is fixed (pre-configured). The following files are provided in demo\output for direct user use:



(E:) › MimicMCU › demo › output

名称	修改日期	类型	大小
sche_rootboot.bin	2025/7/11 14:55	BIN 文件	2 KB
sche_funcboot.bin	2025/7/11 14:55	BIN 文件	2 KB
sche_app_cpu20.bin	2025/7/11 14:56	BIN 文件	18 KB
sche_app_cpu12.bin	2025/7/11 14:56	BIN 文件	18 KB
sche_app_cpu012.bin	2025/7/11 14:50	BIN 文件	21 KB
sche_app_cpu2.bin	2025/7/11 14:59	BIN 文件	18 KB
sche_app_cpu1.bin	2025/7/11 14:58	BIN 文件	18 KB
sche_app_cpu01.bin	2025/7/11 14:55	BIN 文件	18 KB
sche_app_cpu0.bin	2025/7/11 14:58	BIN 文件	18 KB

Figure 15 Scheduler CPU Program

Among them, sche_rootboot.bin and sche_funcboot.bin remain unchanged. The sche_app_cpuxxx.bin files are used to control the operating mode of the entire MimicMCU system. Specifically:

- _cpu0.bin controls enabling only the MIPS functional core, i.e., configuring it as MIPS single-mode. Similarly, _cpu1.bin and _cpu2.bin control enabling only the ARM functional core and RISC-V functional core, respectively.
- _cpu01.bin controls enabling only the MIPS and ARM functional cores, i.e., configuring it as MIPS+ARM dual-mode. Similarly, _cpu12.bin configures ARM+RISC-V dual-mode, and _cpu20.bin configures MIPS+RISC-V dual-mode.
- _cpu012.bin controls enabling all three functional cores, i.e., triple-mode.

4.Flashing

4.1. Test Board

This description applies to the ESC0830EVBT0.1 development board. Please refer to the following documents for usage details:

- ESC0830 EVB T0.1 Development Board User Manual.pdf
- ESC0830 EVB T0.1 Development Board Schematic.pdf

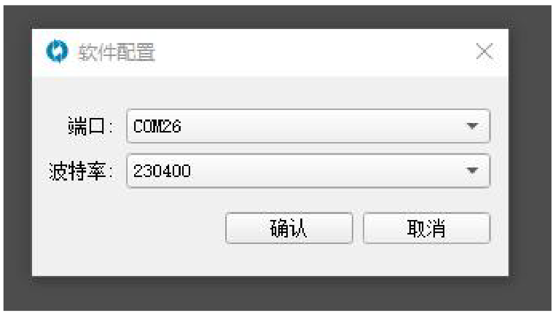
4.2. Flashing and Testing

The previous sections have prepared 12 bin files in the output folder, and the test board has been connected to the host computer. This chapter covers the flashing and testing process.

4.2.1. Flashing

Set the test board to Flash programming mode, turn on the power, click the PML icon in the upper right corner of VSCode, select "mcuBoot," and the flashing tool will pop up.

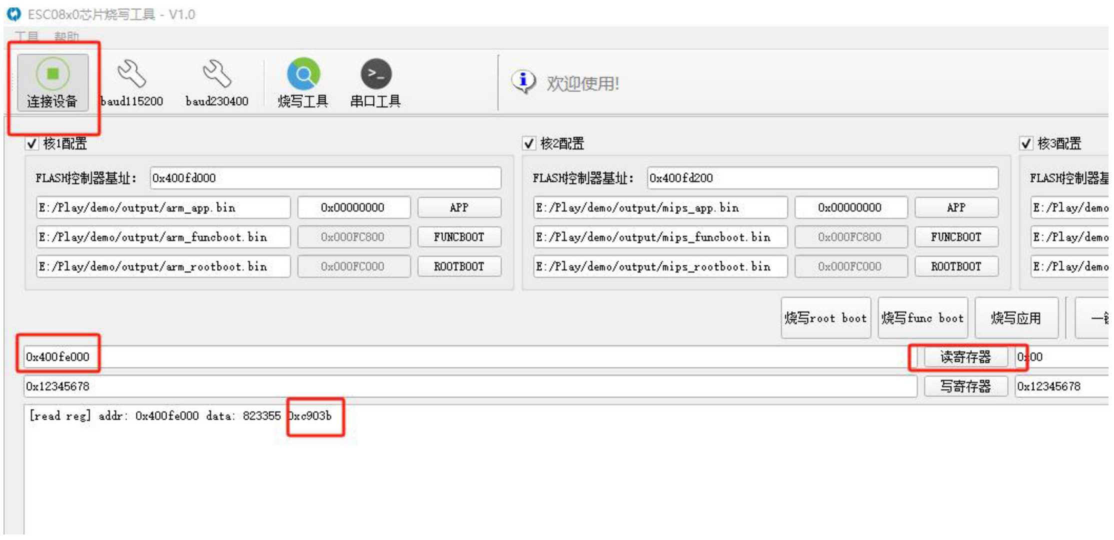
Click "工具"（**Tool** ）→ **"软件配置"（Software Configuration）**, select the **"端口"**（port,） and set the **"波特率"**（baud rate） to **230400**:



端口: COM26
波特率: 230400

Figure 16 Configure the flashing baud rate.

Connect the device（click "连接设备")and read the registers(click "读寄存器"). If c903b is displayed, it indicates that the flashing channel has been successfully established:



核1配置

FLASH控制器基址: 0x400fd000

```
E:/Play/demo/output/arm_app.bin        0x00000000  APP
E:/Play/demo/output/arm_funcboot.bin   0x000FC800  FUNCBOOT
E:/Play/demo/output/arm_rootboot.bin   0x000FC000  ROOTBOOT
```

核2配置

FLASH控制器基址: 0x400fd200

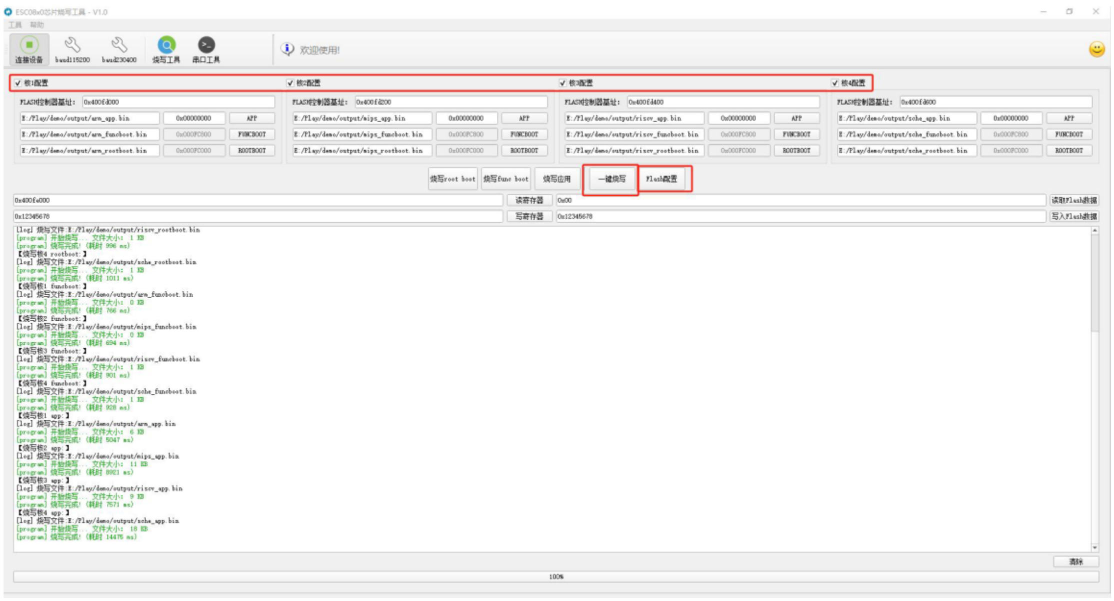
```
E:/Play/demo/output/mips_app.bin        0x00000000  APP
E:/Play/demo/output/mips_funcboot.bin   0x000FC800  FUNCBOOT
E:/Play/demo/output/mips_rootboot.bin   0x000FC000  ROOTBOOT
```

核3配置

```
0x400fe000
0x12345678
[read reg] addr: 0x400fe000 data: 823355 0xc903b
```

Figure 17 Establish Flashing Channel.

As shown in the diagram, configure the base address (a fixed value for the ESC0830 chip as illustrated below) and the bin files, select all cores, and then click ″**Flash 配置**″ (required only once per chip configuration) followed by ″**一键烧写**″:



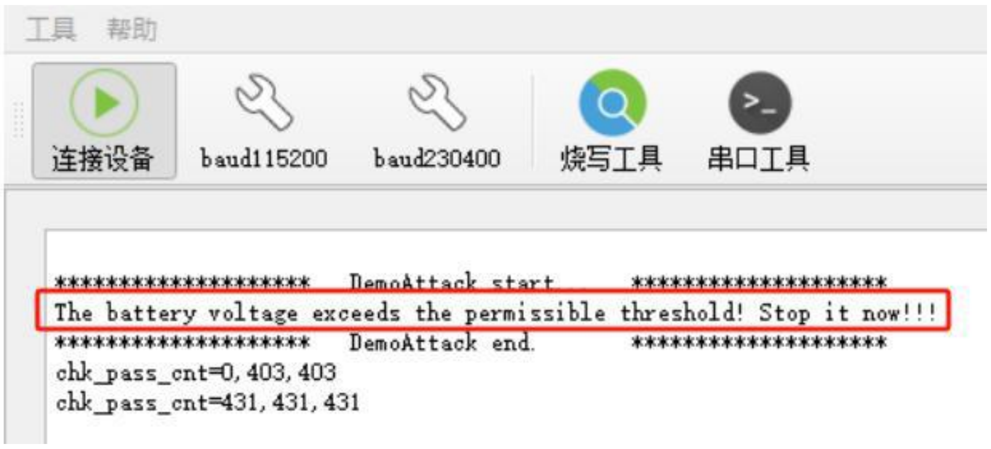
```
[log] 烧写文件:E:/Play/demo/output/riscv_rootboot.bin
[program] 开始烧写... 文件大小: 1 KB
[program] 烧写完成! (耗时 996 ms)
【烧写核4 rootboot:】
[log] 烧写文件:E:/Play/demo/output/sche_rootboot.bin
[program] 开始烧写... 文件大小: 1 KB
[program] 烧写完成! (耗时 1011 ms)
【烧写核1 funcboot:】
[log] 烧写文件:E:/Play/demo/output/arm_funcboot.bin
[program] 开始烧写... 文件大小: 0 KB
[program] 烧写完成! (耗时 766 ms)
【烧写核2 funcboot:】
[log] 烧写文件:E:/Play/demo/output/mips_funcboot.bin
[program] 开始烧写... 文件大小: 0 KB
[program] 烧写完成! (耗时 694 ms)
【烧写核3 funcboot:】
[log] 烧写文件:E:/Play/demo/output/riscv_funcboot.bin
[program] 开始烧写... 文件大小: 1 KB
[program] 烧写完成! (耗时 901 ms)
【烧写核4 funcboot:】
[log] 烧写文件:E:/Play/demo/output/sche_funcboot.bin
[program] 开始烧写... 文件大小: 1 KB
[program] 烧写完成! (耗时 928 ms)
【烧写核1 app:】
[log] 烧写文件:E:/Play/demo/output/arm_app.bin
[program] 开始烧写... 文件大小: 6 KB
[program] 烧写完成! (耗时 5047 ms)
【烧写核2 app:】
[log] 烧写文件:E:/Play/demo/output/mips_app.bin
[program] 开始烧写... 文件大小: 11 KB
[program] 烧写完成! (耗时 8921 ms)
【烧写核3 app:】
[log] 烧写文件:E:/Play/demo/output/riscv_app.bin
[program] 开始烧写... 文件大小: 9 KB
[program] 烧写完成! (耗时 7571 ms)
【烧写核4 app:】
[log] 烧写文件:E:/Play/demo/output/sche_app.bin
[program] 开始烧写... 文件大小: 18 KB
[program] 烧写完成! (耗时 14475 ms)
```

Figure 18 One-click flashing.

4.2.2. Operation

The examples involve printing task information via UART0 and alarm information via UART2, with the baud rate set to 115200 bps for both. Set the test board to normal operation mode, connect to the serial port, reset the test board, and observe the serial output.

Observe UART0: Click the built-in "串口工具"(Serial Port Tool) in the flashing tool.



```
********************    DemoAttack start      ********************
The battery voltage exceeds the permissible threshold! Stop it now!!!
********************    DemoAttack end.       ********************
chk_pass_cnt=0,403,403
chk_pass_cnt=431,431,431
```

Figure 19UART0 Serial Port Results

Observe UART2: Connect an external USB-to-serial adapter. Connect the Tx pin of the adapter to the PD6 pin on the development board, and the Rx pin to the PD7 pin, as shown in Figure 20 below. Use UartAssist V5.0.10.exe to configure and establish the connection.

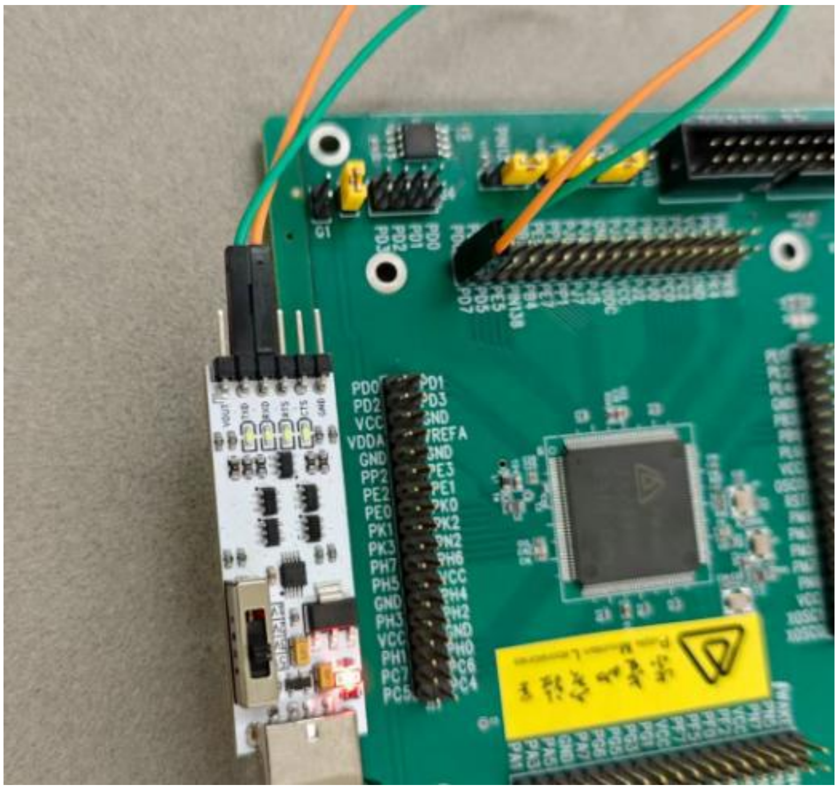
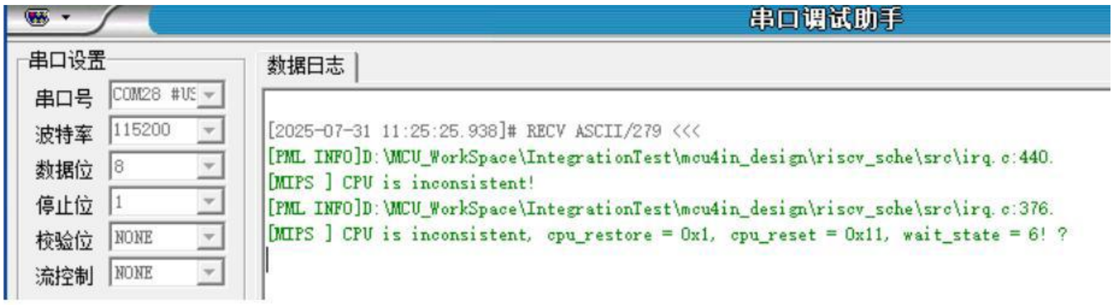


Figure 20 UART2 Serial Port Connection

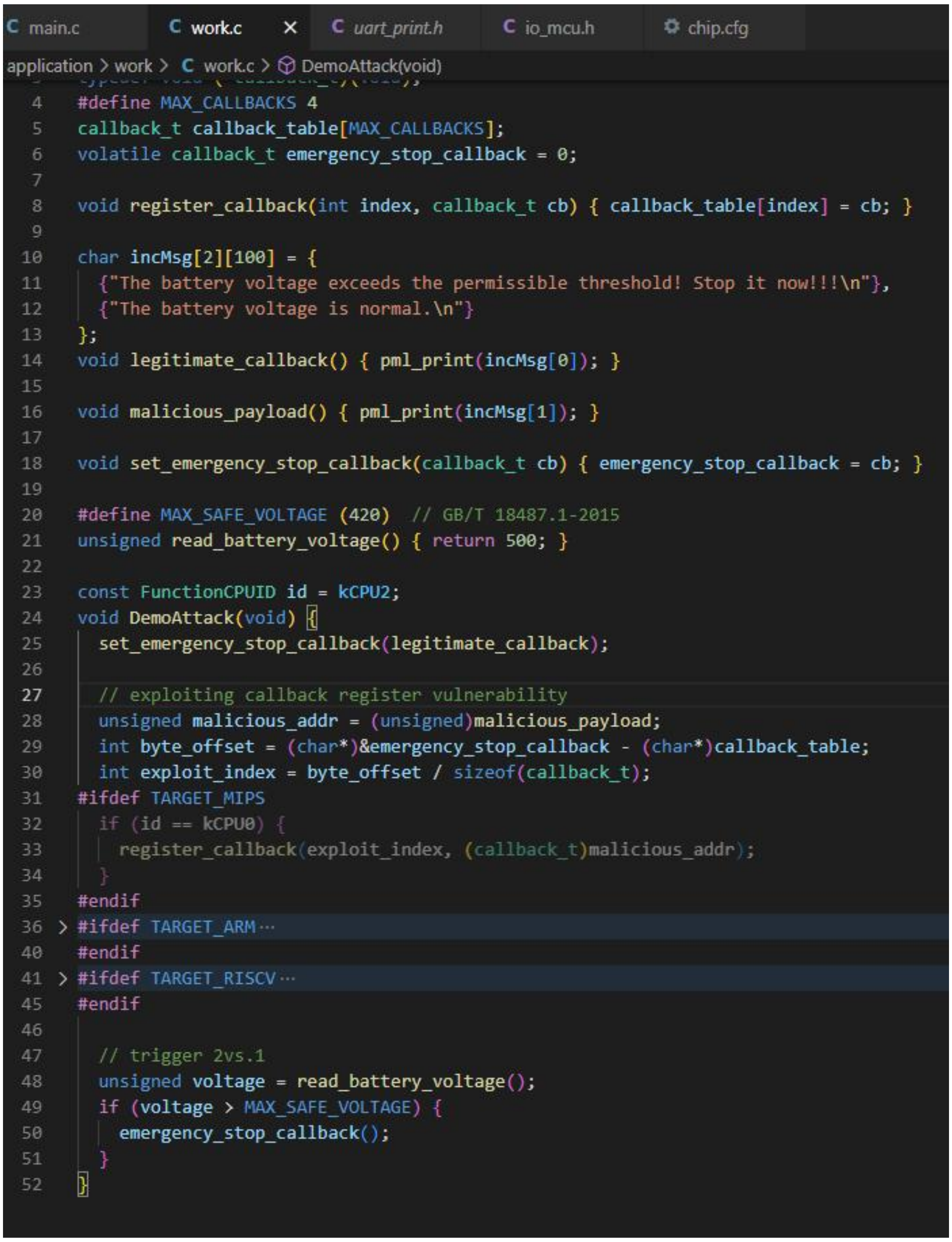


串口号	COM28 #US
波特率	115200
数据位	8
停止位	1
校验位	NONE
流控制	NONE

```
[2025-07-31 11:25:25.938]# RECV ASCII/279 <<<
[PML INFO]D:\MCU_WorkSpace\IntegrationTest\mcu4in_design\riscv_sche\src\irq.c:440.
[MIPS ] CPU is inconsistent!
[PML INFO]D:\MCU_WorkSpace\IntegrationTest\mcu4in_design\riscv_sche\src\irq.c:376.
[MIPS ] CPU is inconsistent, cpu_restore = 0x1, cpu_reset = 0x11, wait_state = 6! ?
```

Figure 21 UART2 Serial Port Result.

4.2.3. Attack Principle



```
#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();
  }
}
```

Figure 22Attack Example Source Code.

As shown in Figure 22 above, all source files in the demo\function_cpus\application\work folder will be compiled, and all header files can be included. Participating teams must implement the void DemoAttack(void); function declared in demo\function_cpus\application\work.h.

In the example, the DemoAttack() function implemented in demo\function_cpus\application\work\work.c exploits an array indexing vulnerability to overwrite a function address near the array. This causes emergency_stop_callback() to execute legitimate_callback() on non-attacked cores, while executing malicious_payload() on the attacked core.

Based on the UART0 results, the overall system continues to execute tasks according to legitimate_callback(). The single-core attack fails to propagate to the external system operating in triple-mode.

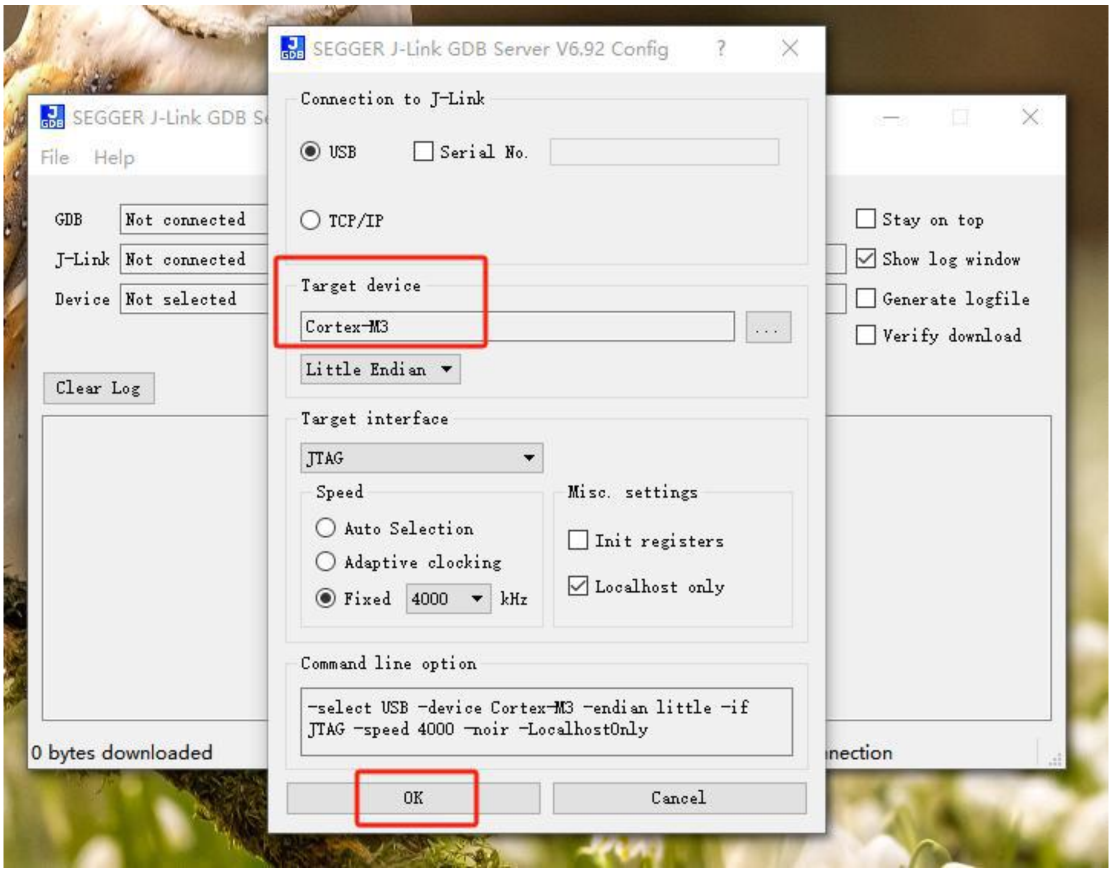
Based on the UART2 results, the system detects and cleans the faulty core, issuing an alarm message.

Note: In Figure 19, only the information inside the red box is controlled by the participating teams. The information outside the red box is automatically generated and used for automatic system state evaluation.

4.3. Single-Step Debugging of ARM Single Core

A dedicated debugging probe, such as J-Link BASE (see www.segger.com), is required.

1. Flash the scheduler core program that enables only the ARM single core. The application layer corresponds to sche_app_cpu1.bin.
2. Flash the ARM functional core program.
3. Set the test board to run mode and open the serial port at a baud rate of 115200.
4. Connect the J-Link BASE debugger to the test board. Open J-Link GDB Server, set the test board to normal operation mode, select "Cortex-M3" as the target device, and connect to the ARM single core:



SEGGER J-Link GDB Server V6.92 Config

Connection to J-Link: [✓ USB] Serial No. TCP/IP

Target device: Cortex-M3

Little Endian

Target interface: JTAG

Speed: Auto Selection, Adaptive clocking, [✓ Fixed] 4000 kHz

Misc. settings: Init registers, [✓ Localhost only]

Command line option:

```
-select USB -device Cortex-M3 -endian little -if
JTAG -speed 4000 -noir -LocalhostOnly
```

Figure 23Connect to the ARM core.

Open the built-in Windows cmd terminal and run the following commands to start single-step debugging:

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

For the debugging process, please refer to the video: ESC0830 单步调试 arm 录屏.wmv