



D12x 快速入门

Version 2.1

修订日期：2025-01-24

版权声明

本文件是匠芯创科技（“ArtInChip”）的原创作品，匠芯创科技拥有该文件的全部版权。全部或部分复制必须获得匠芯创科技的书面批准，并向版权所有人明确确认。凡侵犯本公司版权等知识产权的，本公司将保留依法追究其法律责任的权利。

在法律允许的范围内，在此声明：使用前请仔细阅读合同条款和条件以及相关说明，并严格遵守本文件中的说明。匠芯创科技不对不当行为的后果（包括但不限于电压过高、超频或温度过高）承担任何责任。

匠芯创科技提供的信息仅作为参考或典型应用，本文件中的所有声明、信息和建议不构成任何明示或暗示的担保。匠芯创科技保留随时更改电路设计和/或规格的权利，恕不另行通知。

客户应全权负责获得实施解决方案/产品可能需要的第三方许可，匠芯创科技不承担任何与第三方许可相关的许可费或特许权使用费。对于任何要求的第三方许可证所涵盖的事项，匠芯创科技不承担任何保证、赔偿或其他义务。

凡以任何方式直接或间接使用本文件资料者，视为自愿接受本文件声明的约束。

修订记录

下表记录了 2025-01-09 至今的所有修订记录：

表 0-1 修订记录

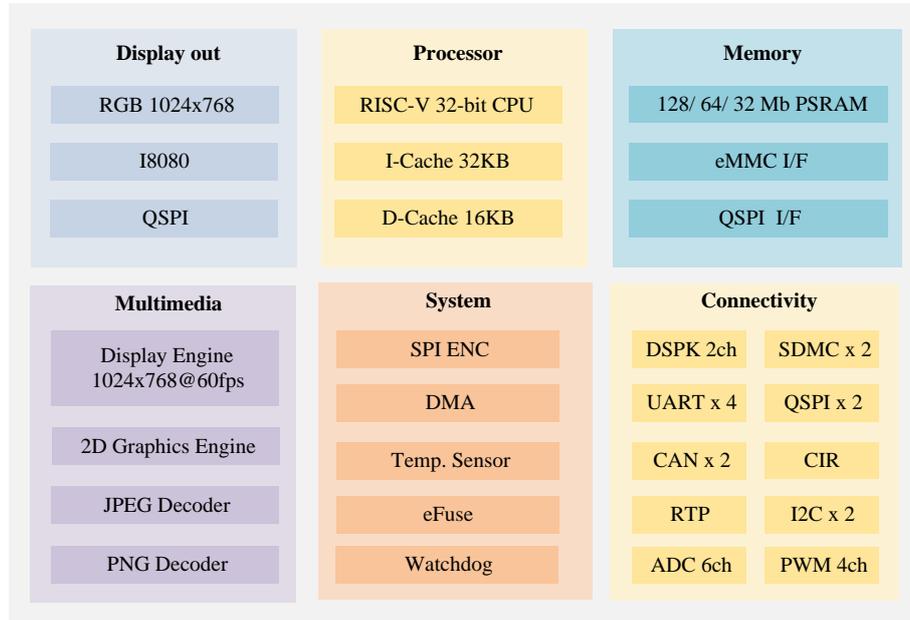
版本	章节	修订说明
V2.1	下载代码仓库	更新了 Gitee 代码库下载链接及相关描述。
	文档资源	
V2.0	-	优化了内容和格式，并调整了章节顺序。

内容

版权声明.....	ii
修订记录.....	iii
1. SoC.....	5
2. 开发板.....	6
2.1. D12x-Demo-V1.0.....	6
2.1.1. 开发版标识.....	6
2.1.2. 规格.....	6
2.1.3. 器件布局.....	6
2.1.4. 实物图.....	7
2.1.5. 方案配置.....	7
2.2. D12x-HMI-V1.1.....	7
2.2.1. 开发版标识.....	7
2.2.2. 规格.....	7
2.2.3. 器件布局.....	8
2.2.4. 实物图.....	8
2.2.5. 方案配置.....	9
3. 下载代码仓库.....	10
4. 编译 SDK.....	11
4.1. RTOS.....	11
4.1.1. VSCode.....	11
4.1.2. Eclipse.....	15
4.1.3. Windows.....	19
4.1.4. Ubuntu.....	21
4.2. Baremetal.....	23
4.2.1. Linux 系统.....	23
4.2.2. Windows 系统.....	23
4.2.3. 编译 Baremetal.....	24
5. 烧写 SDK.....	25
6. 刷机工具.....	26
7. 调试 SDK.....	28
8. 文档资源.....	34
8.1. 文档中心.....	34
8.2. Gitee 下载.....	34
8.3. SDK 内嵌文档.....	34
9. 教学视频.....	35

1. SoC

D12x 是一款基于 RISC-V 的高性能、国产自主、工业级高清显示与智能控制 MCU，配备强大的 2D 图形加速处理器、PNG/JPEG 解码引擎、丰富的接口，支持工业宽温，具有高可靠性、高开放性，可广泛应用于工业自动化控制、串口屏等智慧工业和智慧家居领域。



2. 开发板

D12x 针对不同的封装共开发了下列开发板供用户参考。

开发板	存储	封装	DDR	屏幕	工程目录	其他功能
D12x-Demo-V1.0	SPI NOR	QFN68	8 MB	RGB565	demo68_nor	-
D12x-HMI-V1.1	SPI NOR	QFN68	8 MB	RGB565/ QSPI	hmi_nor	-

2.1. D12x-Demo-V1.0

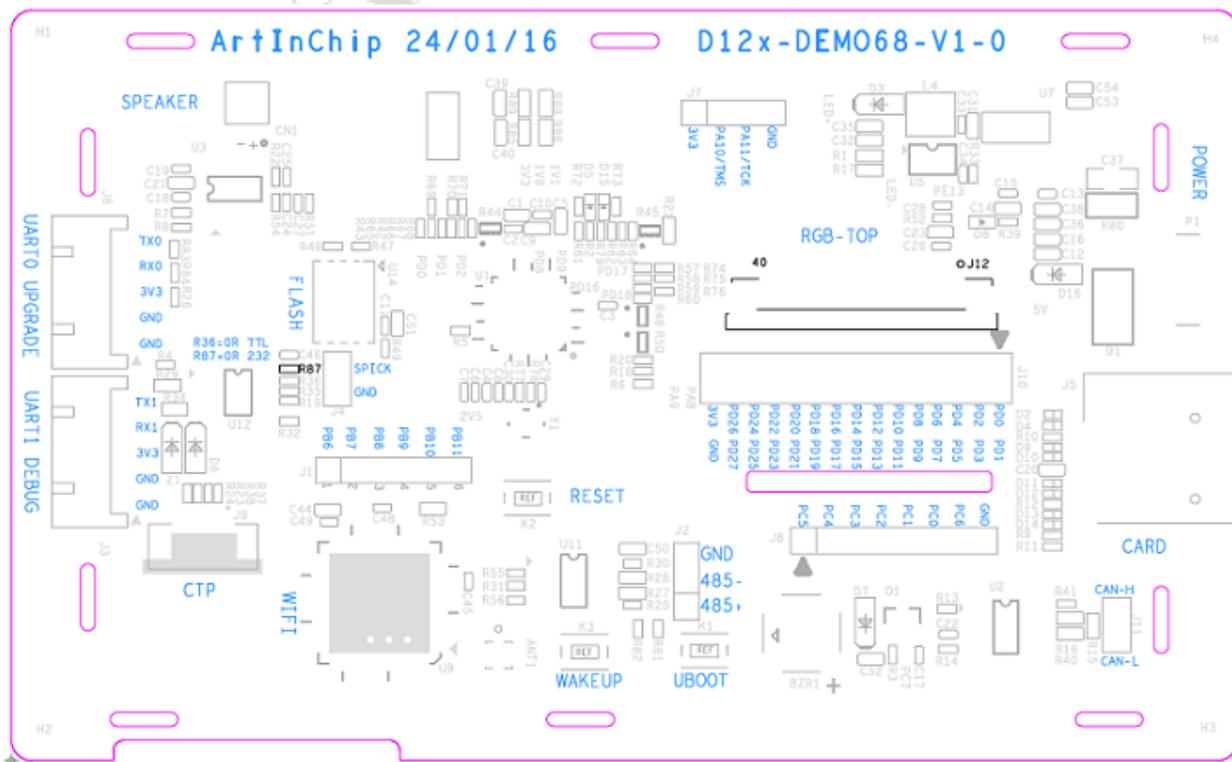
2.1.1. 开发版标识

D12x-Demo68-V1.0

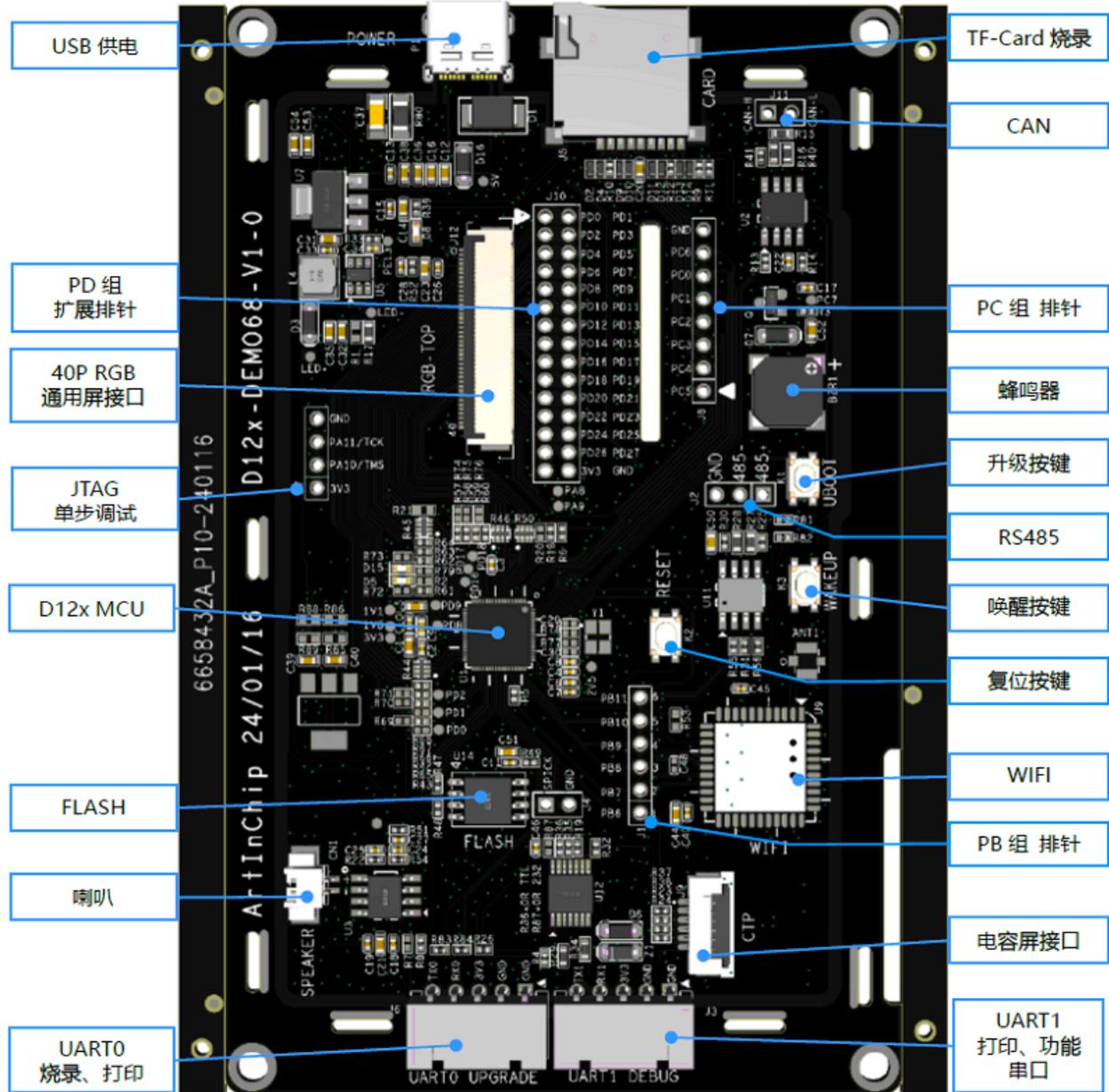
2.1.2. 规格

- RGB888 x 480 x 272
- 8 MB PSRAM + 16 MB NOR
- RS485 + RS232 + UART
- SDIO WiFi + TF-Card
- DSPK + Buzzer

2.1.3. 器件布局



2.1.4. 实物图



2.1.5. 方案配置

方案的配置对应的是 `target/D12x/demo68_nor/` 工程

编译选项: `D12x_demo68-nor_rt-thread_helloworld_defconfig`

固件: `D12x_demo68-nor_v1.0.0.img`

2.2. D12x-HMI-V1.1

2.2.1. 开发版标识

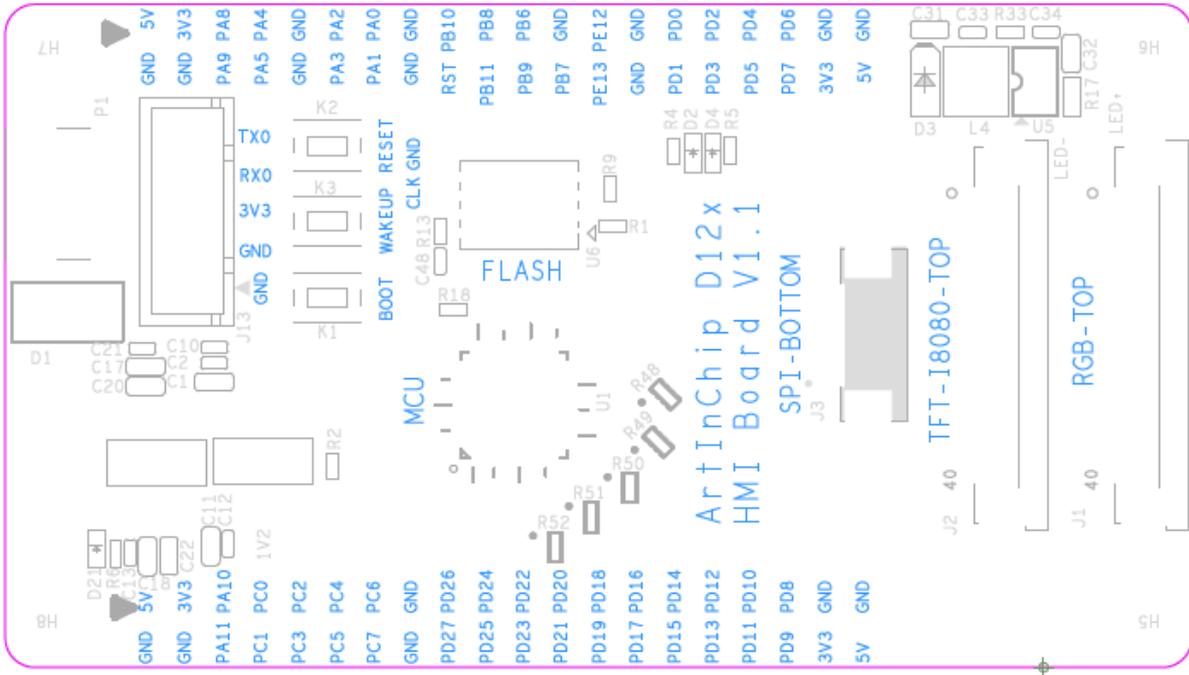
ArtInChip HMI Board V1.1

2.2.2. 规格

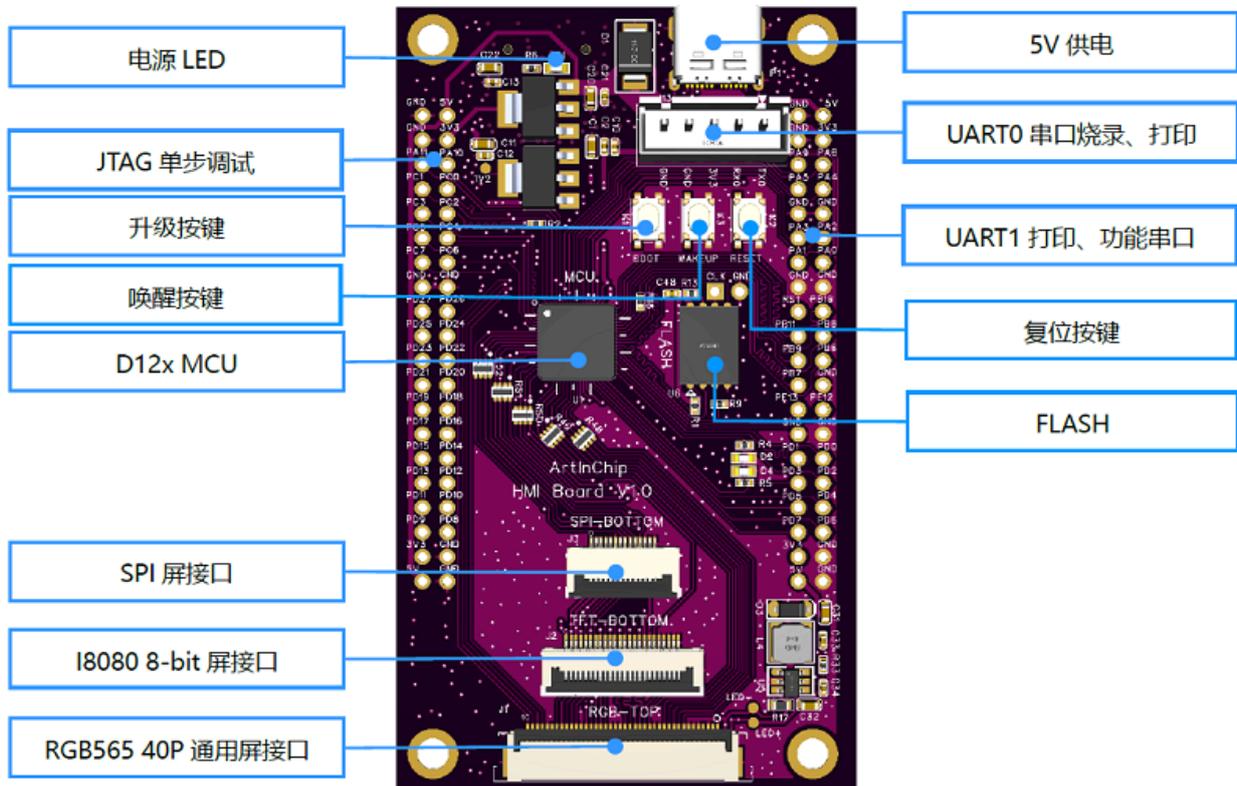
- RGB888 x 480 x 272 + SPI + I8080
- 8 MB PSRAM + 16 MB NOR

- UART x 2
- TF-Card
- 2 x 扩展排针

2.2.3. 器件布局



2.2.4. 实物图



2.2.5. 方案配置

方案的配置对应的是 `target/D12x/hmi_nor/` 工程

编译选项: `D12x_hmi-nor_rt-thread_helloworld_defconfig`

固件: `D12x_hmi-nor_v1.0.0.img`

ArtInChip

3. 下载代码仓库

ArtInChip 通过码云 (gitee) 提供相关仓库的下载资源且全部开源：

- Luban-Lite (RTOS) 代码仓库：

```
git clone https://gitee.com/artinchip/luban-lite.git
```

- Baremetal (裸机) 代码仓库：

```
git clone https://gitee.com/artinchip/baremetal.git
```

- 文档仓库：

```
git clone https://gitee.com/artinchip/docs.git
```

- 工具仓库：git clone https://gitee.com/artinchip/tools.git

广东匠芯创科技有限公司
广东匠芯创科技有限公司以SoC芯片设计、工业人机交互、工业智能算法为核心，致力于成为世界一流的工业应用芯片解决方案供应商。
✉ keliang.liu@artinchip.com 🌐 www.artinchip.com

仓库 (6)

- Docs**
本仓库由广东匠芯创科技有限公司维护，专门用于发布匠芯创的官方文档。这些文档旨在为开源社区提供一个资源，允许任何个人或组织自由地使用、复制、修改...
HTML 1 0 0
- Luban-lite**
Luban-Lite SDK 是由ArtInChip设计的，旨在为系统级芯片(SoC)设计提供一个轻量级、高效且易于使用的软件开发工具包。该SDK的设计规划兼顾了简单性与广泛的套...
34 97 25
- Luban**
本仓库是针对ArtInChip平台优化的编译框架，基于Buildroot的优秀架构并进行了必要的重构和改进。它具有简化的芯片架构和平台、精简的软件包、优化的系...
14 48 29
- Tools**
本仓库是广东匠芯创科技有限公司官方发布的相关工具集。这些工具主要面向系统级芯片 (SoC) 和微控制器单元 (MCU) 的开发与应用，提供了一系列的辅助功...
11 18 0
- Baremetal**
本仓库由广东匠芯创科技有限公司维护，专门用于发布裸机软件开发工具包的官方文档。该SDK作为一个开源项目，允许任何个人或组织自由地使用、复制、修改...
8 11 0
- thirdparty-app**
本仓库用于存储第三方源码 (DL) 库。DL库指的是不由SoC制造商直接开发的、来自外部供应商或开源社区的软件库。
5 1 0

图 3-1 ArtInChip 的仓库示例



注：

ArtInChip 仓库会跟随 ArtInChip 的产品发布而不时更新。

4. 编译 SDK

ArtInChip 提供下列 SDK 供用户选择：

- Luban-Lite 是 ArtInChip 基于 RT-Thread 深度开发的嵌入式实时系统，具有下列特性：
 - 支持 baremetal 构建模式
 - 支持 freerots
 - 支持 rt-thread 核和 rt-thread 生态
- Baremetal 是 ArtInChip 的嵌入式裸机系统。

本章节主要介绍如何使用不同的操作环境快速搭建 SDK 编译环境并编译固件。用户可根据选择的 SDK 和操作环境，执行对应的编译流程。

关于 Eclipse 等 IDE 工具的使用，可参考详细文档。

4.1. RTOS

4.1.1. VSCode

Luban-Lite 支持在 VSCode 环境中完成全流程的开发，包括代码编辑、编译、调试和烧写。VSCode 是一款开源、免费、跨平台的源代码编辑器，由 MicroSoft 开发，特点是轻量级、高性能和可扩展。



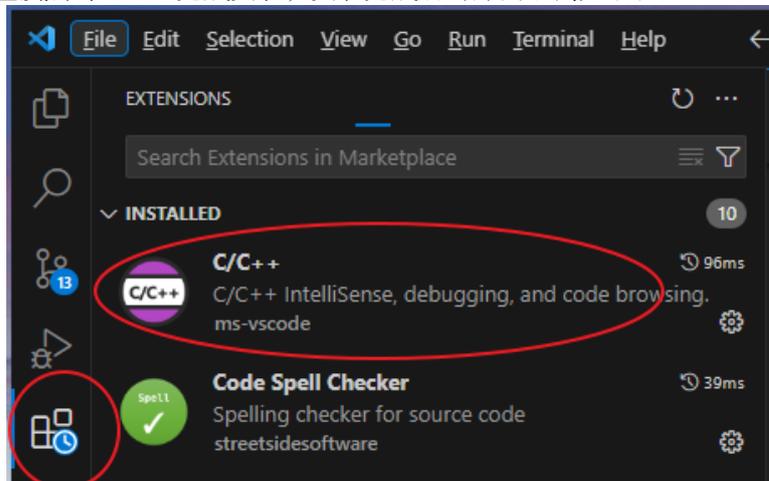
注：

仅在打开 Luban-Lite 根目录的前提下，VSCode 才能完成以下的编译、调试和烧写操作。

4.1.1.1. 插件安装

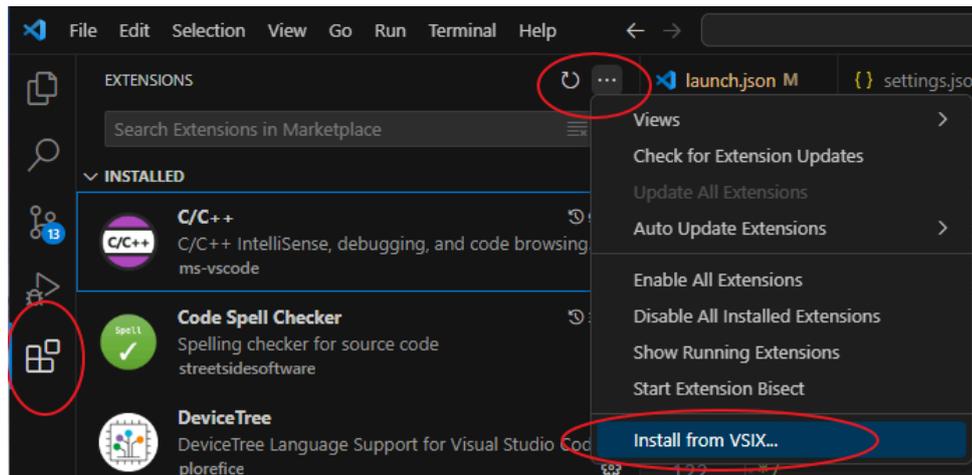
如需在 VSCode 中进行 C/C++ 语言的调试，必须先安装插件 C/C++，如下所示：

- 在 PC 联网的条件下，直接点击 **Install** 完成安装，安装完成后的界面示例如下。



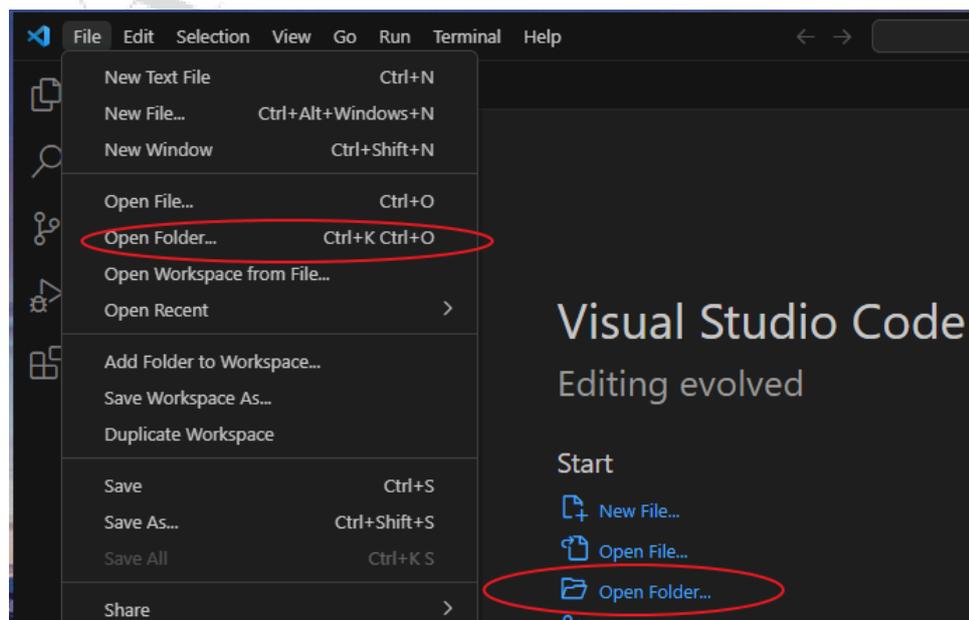
- 在 PC 未联网的条件下，则需要按照以下步骤，手动完成安装：

1. 前往 VSCode 官网下载 C/C++ 插件的安装文件。
插件文件的后缀名为 `.vsix`。
2. 打开 VSCode 终端后，在插件管理界面选择 **Install from VSIX**，如图所示：

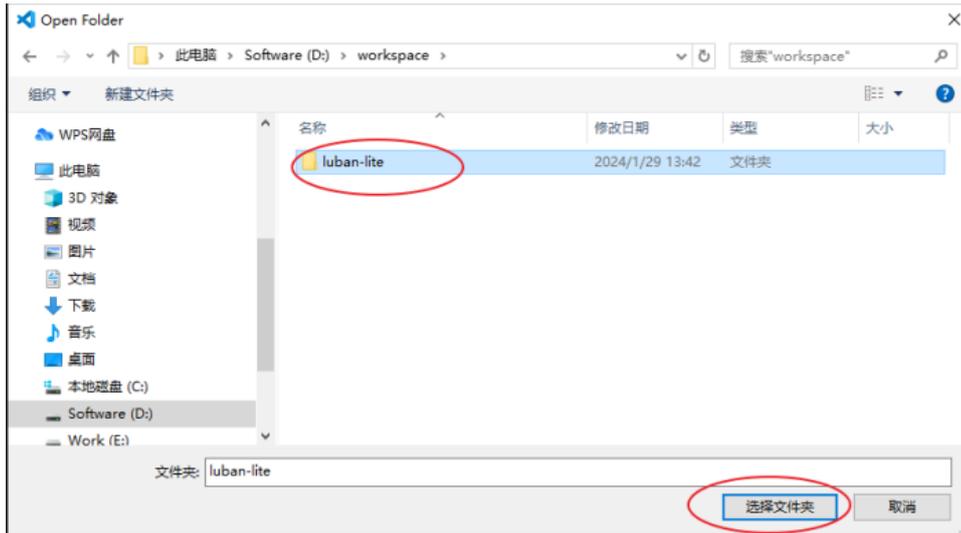


4.1.1.2. 打开工程文件

1. 选择以下任意方式打开一个文件夹目录：
 - 在菜单栏中，选择 **File > Open Folder...**
 - 在 VSCode 的编辑视图中，选择 **Start** 区域的 **Open Folder...**



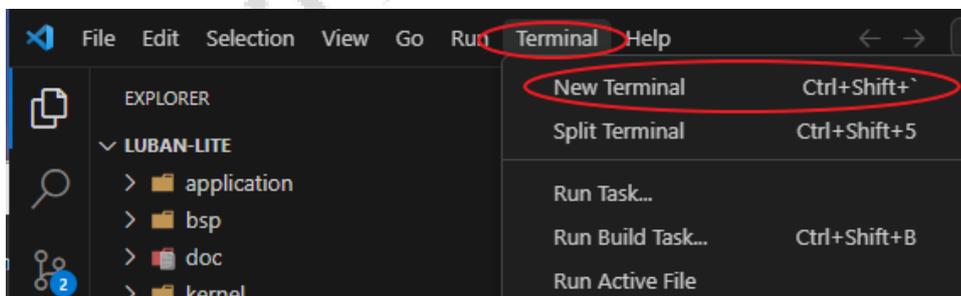
- 在弹出的文件夹浏览窗口中选择 Luban-Lite 根目录，并点击**选择文件夹**：



4.1.1.3. 编译

Luban-Lite 编译前，在 VSCode 的终端中使用 lunch 命令选择一个方案配置。

- 在菜单栏选择 **Terminal > New Terminal** 打开 VSCode 终端窗口。



可使用快捷键 **Ctrl+Shift+`**。

- 在 VSCode 终端窗口中输入并执行 win_cmd.bat。



- 使用 list 命令查询所有工程文件，查询结果如下图所示：

```
E:\AIC\luban-lite\luban-lite>list
scons: Reading SConscript files ...
Built-in configs:
 0. d12x_demo68-nand_baremetal_bootloader
 1. d12x_demo68-nand_rt-thread_helloworld
 2. d12x_demo68-nor_baremetal_bootloader
 3. d12x_demo68-nor_rt-thread_helloworld
 4. d12x_hmi-nor_baremetal_bootloader
 5. d12x_hmi-nor_rt-thread_helloworld
 6. d13x_demo88-nand_baremetal_bootloader
 7. d13x_demo88-nand_rt-thread_helloworld
 8. d13x_demo88-nor_baremetal_bootloader
 9. d13x_demo88-nor_rt-thread_helloworld
10. d13x_kunlunpi88-nor_baremetal_bootloader
11. d13x_kunlunpi88-nor_rt-thread_helloworld
12. d21x_demo128-nand_baremetal_bootloader
13. d21x_demo128-nand_rt-thread_helloworld
14. g73x_demo100-nor_baremetal_bootloader
15. g73x_demo100-nor_rt-thread_helloworld
```

4. 选择开发板或者方案所对应的配置编号，执行命令 `lunch list_no.`

例如，当前环境使用的是 `d12x_demo68-nor_rt-thread_helloworld` 方案，其配置的编号是3，则执行命令 `lunch 3`。

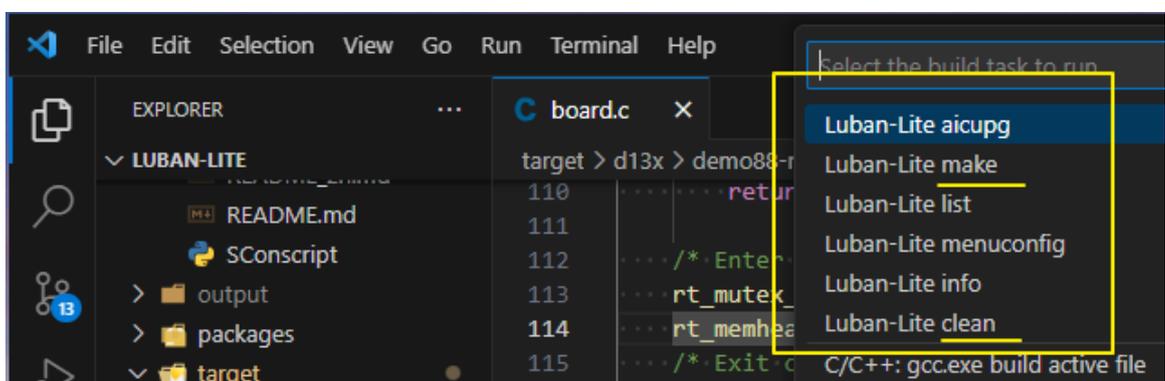
```
E:\git\luban-lite>lunch 3
scons: Reading SConscript files ...
Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig
```

5. 使用以下任意一种方式，可以触发编译：

- 在 VSCode 终端中输入 `m` 命令
- 通过 VSCode 的快捷命令 **Ctrl+Shift+B** 选择 **Luban-Lite make**。

6. 使用以下任意一种方式，可以清除工程：

- 在 VSCode 终端中输入 `c` 命令。
- 通过 VSCode 的快捷命令 **Ctrl+Shift+B**，选择 **Luban-Lite clean**。

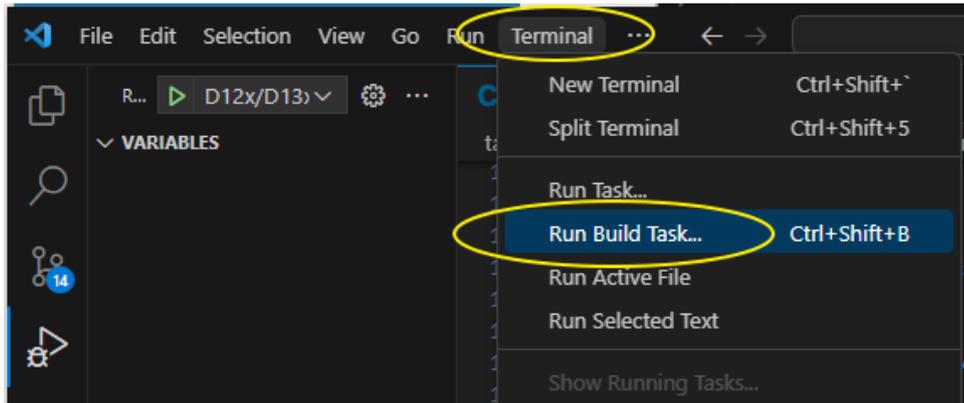


编译成功后的系统输出结果如下所示，表示生成的镜像文件为 `luban-lite\output\D12X_demo88-nor_rt-thread_helloworld\images\D12X_demo88-nor_v1.0.0.img`：

```
Luban-Lite is built successfully
scons: done building targets.
```

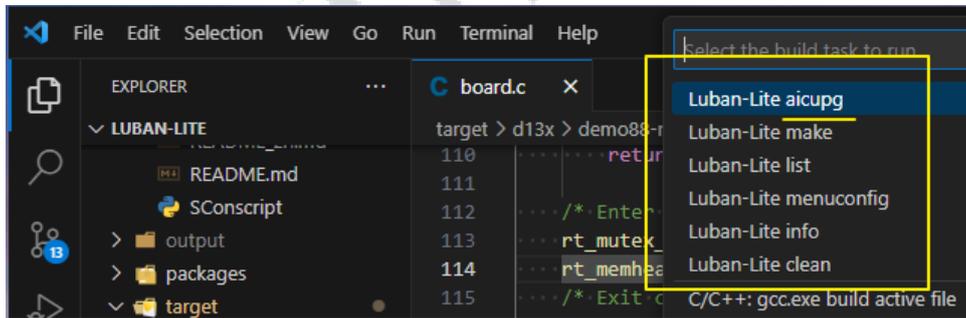
4.1.1.4. 烧写

1. 打开 VSCode 终端窗口 **Terminal**，并选择 **Run Build Task**：



2. 在弹出的命令列表中，选择 **Luban-Liteaicupg** 执行烧写操作。

4.1.1.5. 快捷命令清单



如图所示，Luban-Lite 提供了多个快捷命令，简化用户操作，包括：

- aicupg：执行烧写操作
- make：触发编译
- list：列出当前所有方案配置
- menuconfig：打开 menuconfig 配置界面
- info：查看当前的方案配置
- clean：清理工程

4.1.2. Eclipse

本节介绍了使用 Eclipse IDE 创建、编译和修改 Eclipse 工程的详细流程。

使用 Eclipse IDE 前，确保已经下载和安装 [Eclipse IDE for Embedded C/C++ Developers](#)。

Eclipse 工程编译涉及下列工程文件，两者的区别如下：

- **eclipse** 工程：源码文件和 Luban-Lite SDK 共享，可以使用 menuconfig 配置菜单来更改工程配置。
建议在开发阶段使用 **eclipse** 类型的工程。

- `eclipse_sdk` 工程：把所有的源码文件拷贝一份，脱离了 Luban-Lite SDK 框架，不能使用 `menuconfig` 配置菜单来更改工程配置。

建议在发布阶段使用 `eclipse_sdk` 类型的工程。

4.1.2.1. 生成 Eclipse 工程

一键生成工程文件之前，确保已经在命令行环境下正确配置并且能成功编译工程文件。如果工程配置有更新，则在命令行下编译成功后再次生成 `eclipse` 工程文件。根据实际项目需求，生成当前工程对应的工程文件，具体流程如下所示，下列文件类型二选一：

- 生成 `eclipse` 工程文件：

1. 进入 SDK 根目录：

```
cd luban-lite
```

2. 使用下列命令生成当前工程对应的 Eclipse 工程文件：

```
scons --target=eclipse
```

3. 将生成的 `eclipse` 工程文件存储到 `luban-lite/output/xxxx/project_eclipse` 目录中：

```
ls -a output/d21x_demo100-nand_rt-thread_helloworld/project_eclipse  
./ ../ .cproject .project .settings/
```

- 生成 `eclipse_sdk` 工程文件：

1. 进入 SDK 根目录：

```
cd luban-lite
```

2. 使用下列命令生成当前工程对应的 Eclipse SDK 工程

```
scons --target=eclipse_sdk
```

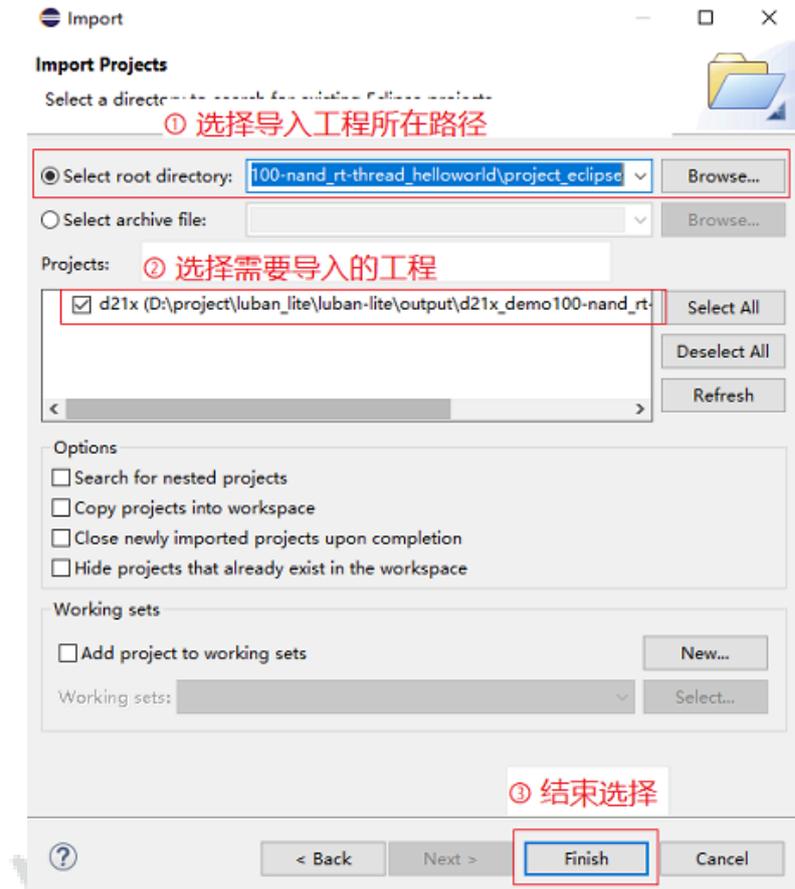
3. 将生成的 `eclipse_sdk` 工程文件存储到 `luban-lite/output/xxxx/project_eclipse_sdk` 目录中。

该目录拷贝了所有必要文件，可以作为一份独立的 SDK 拷贝到任何路径下进行调试。

4.1.2.2. 导入 Eclipse 工程

1. 打开下载的 `Eclipse IDE for Embedded C/C++ Developers` 文件。
2. 在菜单栏中，选择 `File > Import > Existing Projects into Workspace`。
3. 在 `Import Projects` 界面，分别选择导入工程所在路径和需要导入的工程文件。

示例如下：



4. 选择 **Finish** 完成导入。

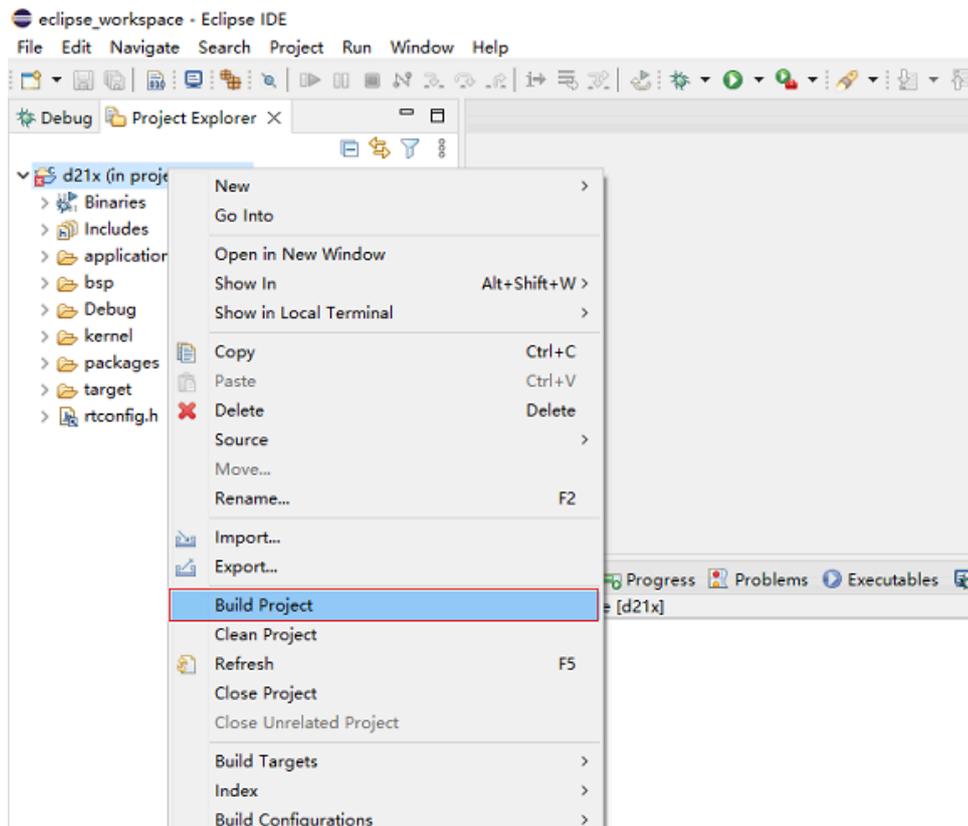
4.1.2.3. 编译

1. 在 **Project Explorer** 中选择和打开成功导入的工程。
2. 选中所需工程，并点击鼠标右键。
3. 在鼠标右键菜单中选择 **Build Project**，对整个工程进行编译。



注：

对于首次编译操作，需要选择 **Clean Project** 清理工程，避免出现环境的兼容性问题，



4. 等待编译完成。

编译生成的文件存放在 `luban-lite/output/xxxx/project_eclipse/Debug` 目录中，示例如下：

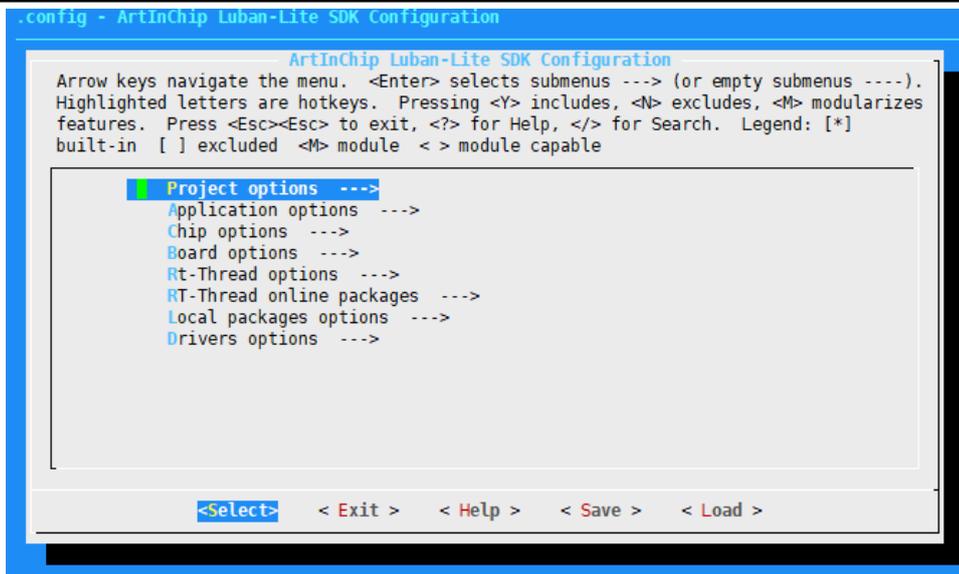
```
ll output/d21x_demo100-nand_rt-thread_helloworld/project_eclipse/Debug/
d21x.bin
d21x.elf // 调试需要的 elf 文件
d21x.map
d21x_demo100_nand_page_2k_block_128k_v1.0.0.img // 烧录需要的 img 文件
```

4.1.2.4. 更改 Eclipse 工程配置

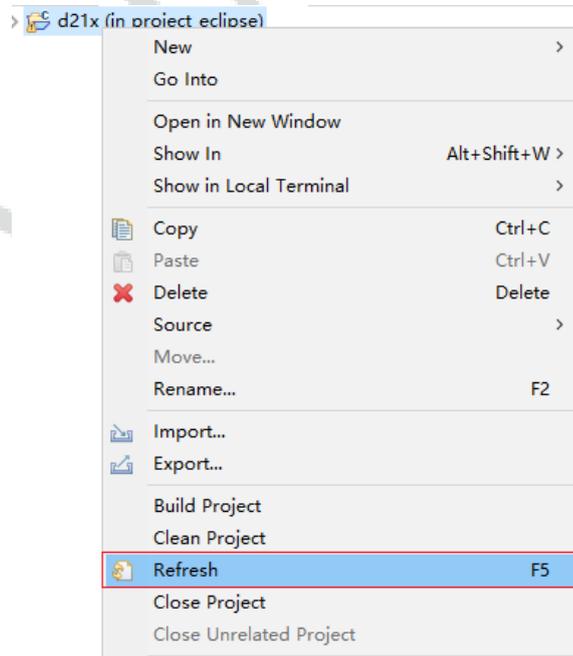
对于使用 `scons --target=eclipse` 命令生成的 Eclipse 工程，可以通过 `menuconfig` 菜单来更新工程的配置。

1. 进入 Luban-Lite 根目录后，执行 `scons --menuconfig`。

```
cd luban-lite
scons --menuconfig // 更改当前工程配置
```



2. 在 menuconfig 菜单退出时，系统会自动更新 `luban-lite/output/xxxx/project_eclipse` 目录下的 Eclipse 配置文件。用户在 Eclipse 工程的右键菜单中选择 **Refresh** 刷新即可同步配置：



4.1.3. Windows

本节介绍了 Windows 环境下可以采用的编译方式，以及两个命令行工具的使用。

Luban-Lite SDK 采用了 Scons 作为编译框架的基础语言，Windows 环境中使用的工具存放在 `luban-lite/tools/env/tools` 目录中，不需要单独安装。

4.1.3.1. 常规编译

Windows 环境下的常规编译流程如下所示：

1. 工程加载

双击 SDK 根目录下的 `win_cmd.bat` 或 `win_env.bat`，加载工程的现有配置：

- `scons --list-def`
- `scons --apply-def=<项目索引或名称>`



注:

`win_cmd.bat` 和 `win_env.bat` 为两种不同的批处理文件, 详情可查看[批处理文件](#)。

2. 配置

在加载完工程配置后, 使用 `scons --menuconfig` 命令来修改当前配置。

3. 编译

使用 `scons` 命令进行编译。

编译成功的结果输出示例如下:

```
Imagefile is generated:
luban-lite/output/d21x_demo100-nand_rt-thread_helloworld/images/d21x_demo100_nand_page_2k_block_128k_v1.0.0.img
```

编译后固件名称为 `d21x_demo100_nand_page_2k_block_128k_v1.0.0.img`

- 使用 `scons --verbose` 命令打印详细的编译信息。
- 使用 `scons --clean` 命令清理当前工程。
- 使用 `ls` 编译生成的目标文件:

```
ls output/$chip_$board_$kernel_$app/images/$soc.elf
```

4.1.3.2. OneStep

OneStep 是 ArtInChip 对 SCons 工具二次封装的总称, 在基础命令上开发了一组更高效和方便的快捷命令, 以实现任意目录、一步即达的目的。在 CMD 或者 ENV 窗口启动后, OneStep 命令已经生效, 可以从任意目录执行命令。关于 OneStep 命令的详细描述, 可查看 OneStep 命令参考指南。

```
E:\workspace\luban-lite>h
Luban-Lite SDK OneStep commands:
h          : Get this help.
lunch [No.] : Start with selected defconfig, .e.g. lunch 3
me        : Config SDK with menuconfig
m        : Build all and generate final image
c        : Clean all
croot/cr  : cd to SDK root directory.
cout/co   : cd to build output directory.
cbuild/cb : cd to build root directory.
ctarget/ct : cd to target board directory.
list      : List all SDK defconfig.
i        : Get current project's information.
buildall  : Build all the *defconfig in target/configs
rebuildall : Clean and build all the *defconfig in target/configs
aicupg    : Burn image file to target board
```

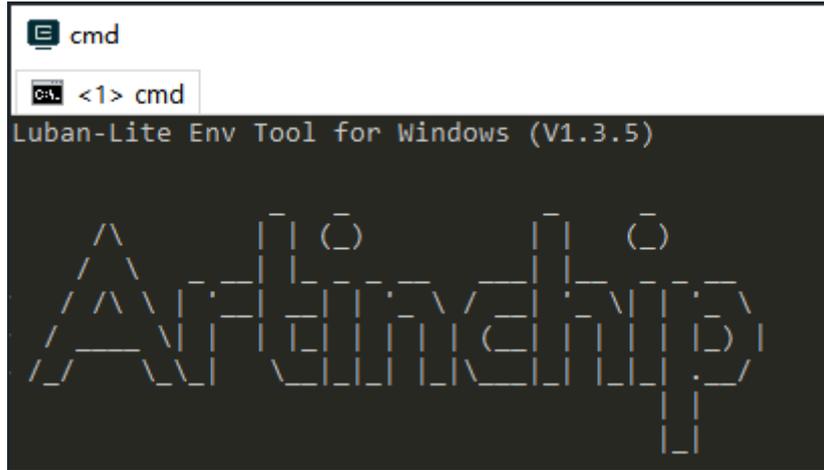
4.1.3.3. 批处理文件

在 SDK 根目录下有两个批处理文件来实现命令行的使用, 如下所示:

• ENV 运行环境

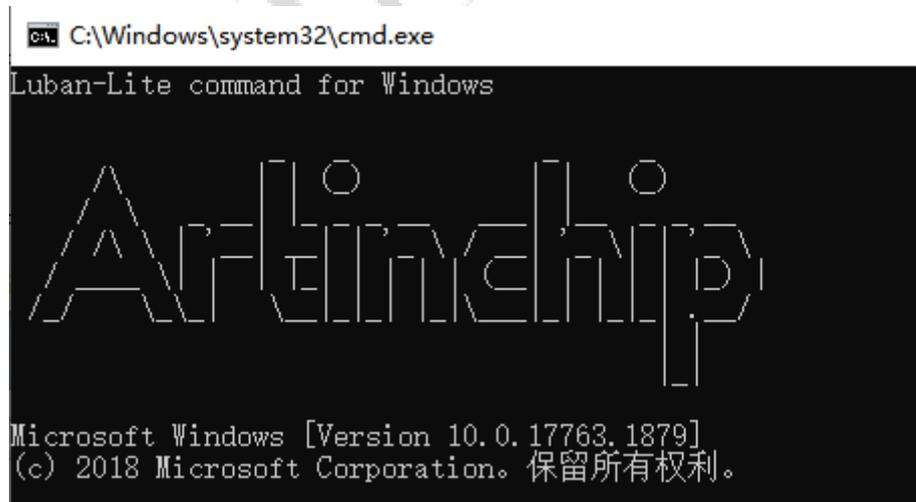
直接双击 `luban-lite/win_env.bat` 打开 Windows 专有的 `env` 命令行工具, 后面所有命令都在该命令行工具中进行操作。

ENV 是 RT-Thread 的原生工具，是 SDK 包中集成了编译所需要使用的所有的工具的一种使用方式



• CMD 运行环境

直接双击 `luban-lite/win_cmd.bat` 打开 Windows 的 CMD 命令行工具，后面所有命令的使用和 ENV 相同。



CMD 是 Windows 的使用环境，除了 SDK 的命令外，还可以使用系统自己安装的工具的命令，因此功能更强大。

4.1.4. Ubuntu

Luban-Lite SDK 的开发可以在 Linux 系统中进行，Luban-Lite SDK 目前自动支持的 Linux 发行版为：

- Ubuntu 18.04、20.04、22.04
- CentOS 7.x、8.x

ArtInChip 推荐的 Linux 发行版为 Ubuntu 20.04 LTS (Long Term Support) 版本，本节以此版本展示 Ubuntu 系统安装注意事项。如使用其他版本，需要安装软件包对应的依赖和版本。

4.1.4.1. 系统安装注意事项

安装 Ubuntu 系统时，需注意以下事项：

- 至少保留 10 GB 磁盘空间，用于保存 SDK 源码。
- 若使用虚拟机，不建议将 SDK 放在虚拟机与实体机的共享目录。

**注:**

本节不涉及 Ubuntu 系统安装的详细步骤。关于详细安装流程，可查看[安装 Ubuntu](#)。

4.1.4.2. 准备编译环境

Luban-Lite SDK 的开发环境中，还需要安装一些依赖包，所需关键工具的版本要求和说明如下所示：

- Python2：用于编译
- SCons：自动化构建工具
- Python3 + pycryptodomex：用于打包和签名

依赖包的安装方法很多，本文以基于 apt 的在线安装方案为例。

1. 资源库更新

使用 apt 安装软件时，可能出现 <http://cn.archive.ubuntu.com/ubuntu> 无法访问或者访问速度较慢的情况，此时可使用境内镜像网站提供的安装包，例如 mirrors.aliyun.com。

```
sudo gedit /etc/apt/sources.list 中 cn.archive.ubuntu.com 全部更换为 mirrors.aliyun.com
sudo apt-get update
```

2. 安装 SCons

Luban-Lite 选择 SCons 作为构建工具，且同时支持对 Makefile 的调用。SCons 是一个以 Python 语言编码的自动化构建工具，是 Make 经过改进后的替代品，可跨平台使用。

```
sudo apt install SCons
```

3. 安装 pycryptodomex

pycryptodomex 是 Python 的一个加密库，可以通过 pip 安装 whl 文件，或通过源码编译安装。SDK `tools/env/local_pkgs/` 中内置了 pycryptodomex 源码。

pycryptodomex 的两种安装方式都需要安装 pip 来提供相应的组件。Ubuntu20.04 默认安装 python3-pip。Pip 安装命令如下：

```
sudo apt install pip
cd tools/env/local_pkgs/
tar xvf pycryptodomex-3.11.0.tar.gz
cd pycryptodomex-3.11.0
sudo python3 setup.py install
```

4.1.4.3. 开始编译

进入 Luban-Lite 根目录，使用 SCons 进行编译，并校验环境是否搭建成功。

**注:**

关于 SCons 命令的详细使用说明，可查看[scons 命令参考指南](#)。

1. 执行下列命令查看所有配置信息：

```
scons --list-def
```

2. 执行下列命令应用具体配置，比如 0 号配置：

```
scons --apply-def=0
```

3. 执行下列命令进行编译:

```
scons
```

OneStep 命令

OneStep 是 ArtInChip 对 SCons 工具二次封装的总称，是在基础命令上开发的一组更高效和方便的快捷命令，以实现任意目录、一步即达的目的。

在 Ubuntu 终端中，进入 SDK 根目录后，使用 `sourcetools/onestep.sh` 命令即可查看所有常见命令。关于 OneStep 详细的使用说明，可查看 OneStep 命令参考指南。

4.2. Baremetal

Baremetal 是 ArtInChip 的嵌入式裸机系统，本章节主要介绍如何在 Linux 和 Windows 上快速搭建环境和编译固件

4.2.1. Linux 系统

在 Linux 系统上搭建 Baremetal 的开发环境需要安装一些依赖包:

1. 进入 SDK 根目录:

```
cd berametal/
```

2. 安装自动化构建工具 scons

```
sudo apt install scons
```

3. 安装 Python2, 用于编译

```
sudo apt install pip
```

4. 安装 Python3 + pycryptodomex, 用于打包和签名

```
cd tools/env/local_pkgs/
```

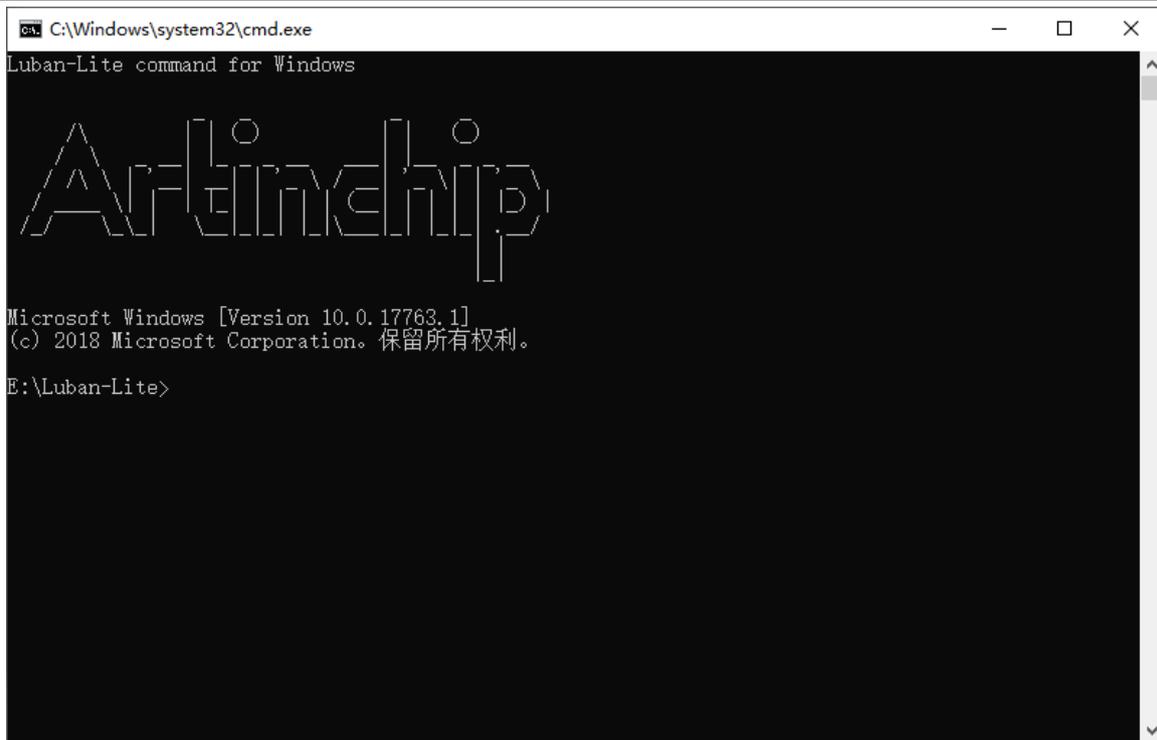
```
tar xvf pycryptodomex-3.11.0.tar.gz
```

```
cd pycryptodomex-3.11.0
```

```
sudo python3 setup.py install
```

4.2.2. Windows 系统

Windows 下对应的各种工具已经存放在 `berametal/tools/env` 目录当中，不需要安装，直接双击 `berametal/win_env.bat` 或者 `berametal/win_cmd.bat` 即可



4.2.3. 编译 Baremetal

```
scons --list-def           //查看有多少配置
scons --apply-def=0       //选择 0 号配置
scons                     //编译
```

Image file is generated:
berametal/output/d21x_demo100-nand_rt-thread_helloworld/images/d21x_demo100_nand_page_2k_block_128k_v1.0.0.img

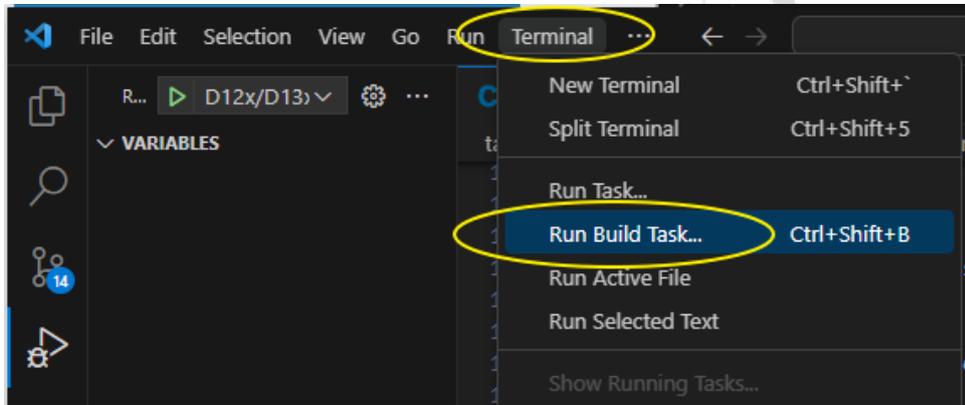
编译后的固件名称为 `d21x_demo100_nand_page_2k_block_128k_v1.0.0.img`。

5. 烧写 SDK

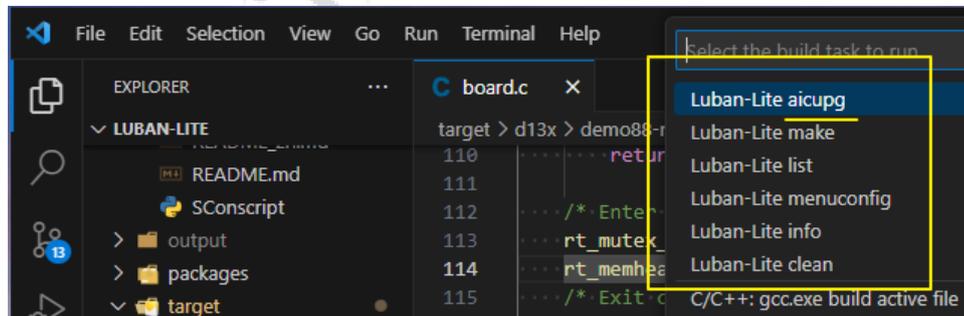
Luban-Lite 的 OneStep 命令 和 VSCode 的快捷命令中都集成了烧写功能。启动方法如下：

1. 选择以下任意方法让板子进入烧写模式。

- OneStep 命令方式：在 VSCode 的终端中执行命令 `aicupg`。
- VSCode 从界面中执行快捷命令的方式，即 **Ctrl + Shift + B**。



2. 在弹出的命令列表中，选择 **Luban-Lite aicupg**：



Luban-Lite 中还提供了其他快捷命令，包括：

- list：列出当前所有方案配置
- menuconfig：打开 menuconfig 配置界面
- info：查看当前的方案配置

6. 刷机工具

ArtInChip 提供两组工具：

- AiBurn：单机调试刷机工具
- AiBurnPro：一拖八量产刷机工具



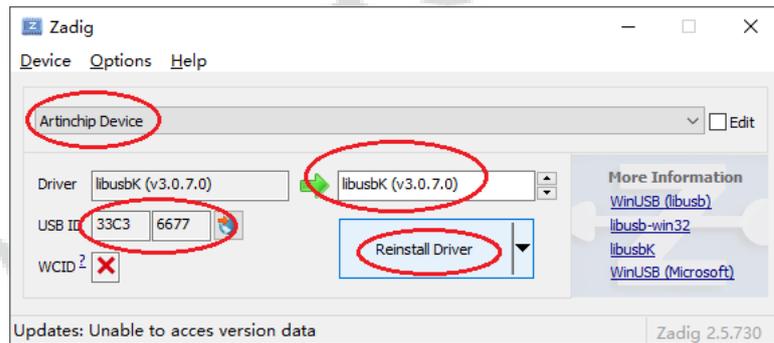
注：

按照[下载代码仓库](#)中的说明，可下载刷机工具。

刷机流程如下所示：

1. 整理驱动。

AiBurn 通过 USB 烧录固件时需要 **libusb** 的支持。如果所用计算的 USB 驱动安装过于复杂而导致 ArtInChip 设备驱动安装异常时，建议使用 **tools** 下的 Zadig 工具整理 USB 驱动，方法如下图：



2. 如果使用 AiBurn，则选择编译好的镜像，在开发板进入烧写模式后点击**开始**按钮即可自动进行烧写。



用户也可以根据实际情况，选择以下任意方式进入烧写模式：

- 终端设备为空片，则上电直接进入 USB 烧写模式。
- 启动时（上电或者按**重启键**），按住**烧录键**直接进入烧录模式。
- 启动时短路，存储介质造成读失败可以进入烧录模式（如果 SPI NAND 的 4 和 5 脚）。
- 终端设备非空片，如果能进入终端，则执行命令 `aicupg`，系统直接重启进入烧写模式。

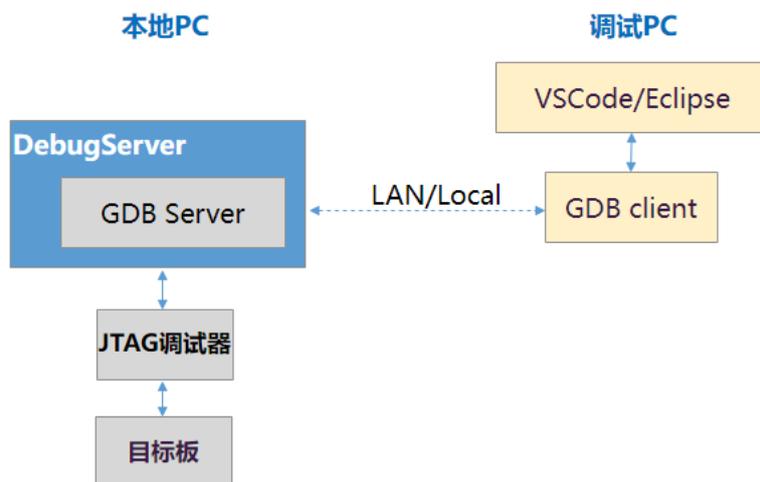
3. 串口调试。烧写镜像完成后可以通过串口进行信息的查看，默认的调试串口配置信息为：

- **BaudRate:** 115200
- **Data bits:** 8
- **Stop bits:** 1
- **Parity:** None
- **Flow control:** None

ArtInChip

7. 调试 SDK

JTAG 调试的整个物理环境示意图如下，本地 PC 和调试 PC 可以是同一台 PC，也可以是局域网内不同的两台 PC：



执行调试流程之前，需要先准备 JTAG 调试的物理环境，包括：

- 硬件：

- 板子上有 JTAG 插座，或者飞线引出了 JTAG 信号线，可以连接到 JTAG 调试器。
- JTAG 调试器：Luban-Lite 支持 CKLink 调试器和 AIC JTAG 两种。



- 保证板子和 JTAG 调试器的信号线正确连接，请参考调试器上的信号标识。

- 软件：

- 安装 T-HeadDebugServer：调试器在 PC 端的代理，提供 GDB Server 调试服务。
- 安装 AiBurn：ArtInChip 烧录软件，需要用到其中的 USB 烧写驱动。

 **注：**

调试前，确保 PC 已安装上述软件。由于安装过程涉及驱动安装，务必开启管理员权限。关于软件的安装包，可以在工具包中找到。所有资源均可在代码仓库中[下载](#)。

- 选择当前需要 JTAG 调试的场景。JTAG 调试包括以下两种场景：
 - 板子刚执行完 PSRAM/ DDR 的初始化，等待 JTAG 连接，Debug 配置选择执行：Dxx load
 - 板子上已经在运行一份镜像，中途用 JTAG 连接，Debug 配置选择执行：Dxx connect only

Luban-Lite 的 VSCode 配置中已经默认提供了四种 JTAG 选择，选择的方法：

SoC 型号	板上无镜像	板上已经在运行镜像
D21x	D21x load	D21x connect only
D12X/D12x	D12X/D12x load	D12X/D12x connect only

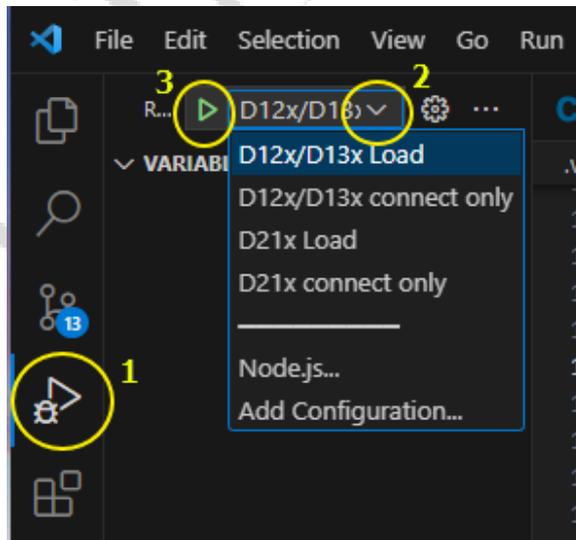
完成上述 JTAG 环境准备后，遵照以下流程调试 SDK：

1. 根据 JTAG 调试的场景，在 VSCode 中选择合适的 Debug 配置方法，点击箭头小图标（快捷键 **F5**），界面操作如下：



注：

Debug 配置只需选择一次，VSCode 会记住上次的配置。



2. 运行 T-HeadDebugServer 并配置下列参数：

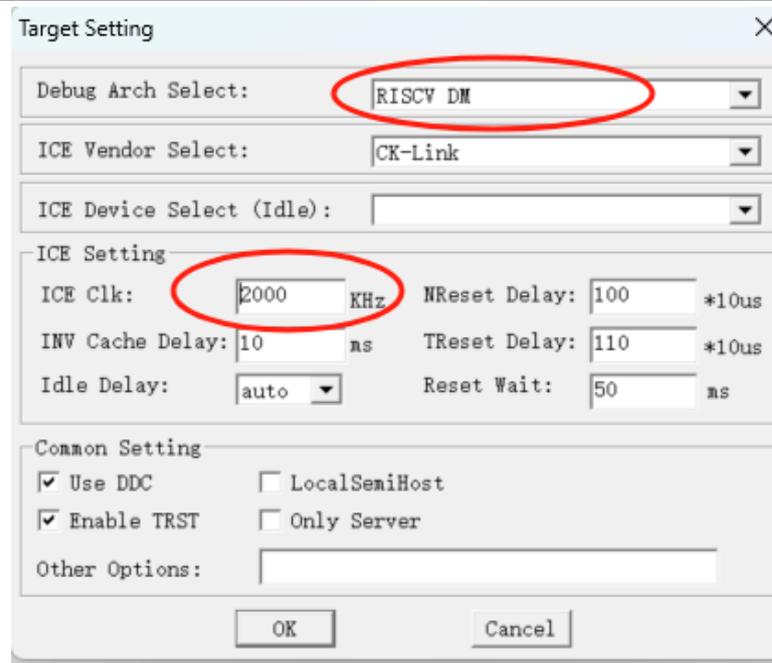


注：

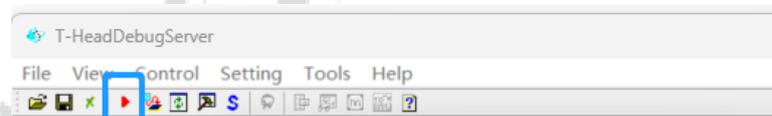
T-HeadDebugServer 完成安装后，桌面会自动创建一个 T-HeadDebugServer 的图标。

在 **Setting** 目录下选择 **Target Setting**

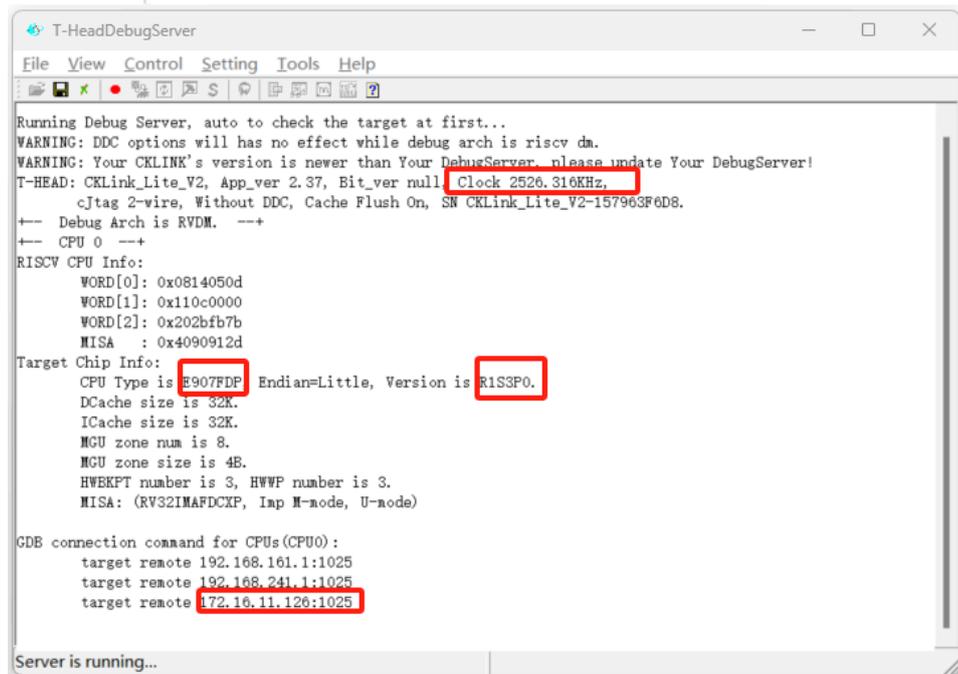
- **Debug Arch Select:** 必须选择 **RISC-V DM**
- **D1:** 需要选择 **CSKY HAD**



3. 等待调试器正常连接且目标板上电后，点击红色小三角按钮，或者点击 **Control > RunDebugServer**，开始连接设备：



正常情况下，执行上述步骤会看到扫描出 CPU 信息如下：



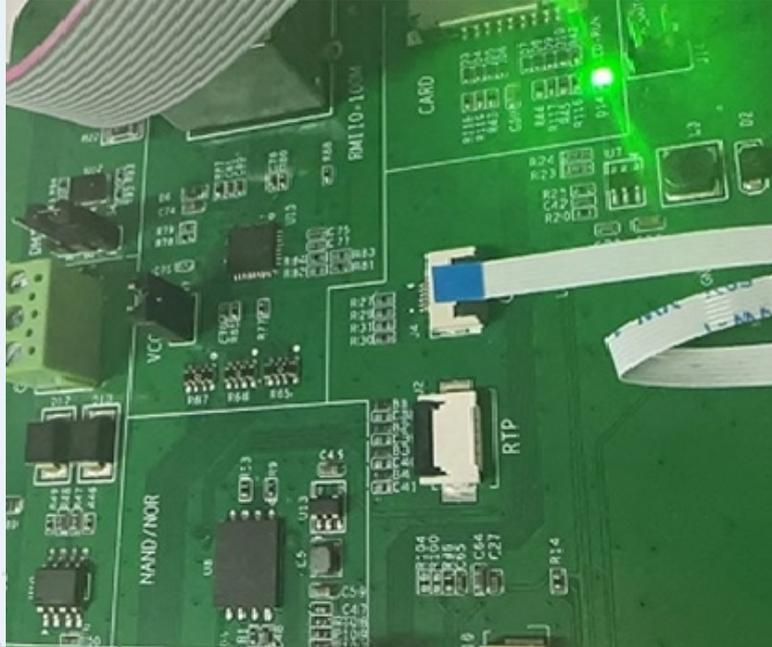
注：

- 通过 JTAG load elf 前，必须要先完成 PSRAM、或者 DDR 初始化，否则 JTAG 在写 PSRAM/ DDR 时会出现异常，使用 JTAG 口需要关闭 IIC、以及 Touch panel，操作命令如下：
 - a. 运行 `scons --menuconfig` 命令。
 - b. 执行下列命令关闭 I2C 和 Touch Panel。



```
Board options --->
  [] Using i2c3
Drivers options --->
  Peripheral --->
    Touch Panel Support --->
      Gt911 touch panel options --->
        [] Using touch panel gt911
```

c. 若使用 JTAG 口，另外需断开 CTP 触屏排线。



4. 在 VSCode 中修改 `Luban-Lite/.vscode/launch.json` 文件的下列参数，与当前方案配置保持一致：

- 路径名、elf 文件名
- DebugServer 的服务 IP 和端口号，选择 **Setting > Socket Setting**
- 断点

如果要添加多个断点，方法如下：

```
53 // FIXME
54 "text": "b.rt_hw_board_init"
55 },
56 {
57   "text": "b.aic_board_pinmux_init"
58 },
59 }
```

以 Dxx load 为例，修改方法如下：

```

8
9   "name": "D12x/U13x Load",
10  "type": "cppdbg",
11  "request": "launch",
12  "cwd": "${workspaceFolder}",
13  "program": "${cwd}/output/d13x_demo88-nor_rt-thread_helloworld/images/d13x.elf", // FIXME
14  "args": [],
15  "stopAtEntry": false,
16  "environment": [],
17  "externalConsole": true,
18  "MIMode": "gdb",
19  "miDebuggerPath": "${cwd}/toolchain/bin/riscv64-unknown-elf-gdb.exe",
20  "setupCommands": [
21    {
22      "description": "Enable pretty-printing for gdb",
23      "text": "-enable-pretty-printing",
24      "ignoreFailures": true
25    },
26    {
27      "text": "set arch riscv:rv32"
28    },
29    {
30      "text": "set height 0"
31    },
32    {
33      "text": "mem 0x30040000 0x3013ffff rw"
34    },
35    {
36      "text": "mem 0x10000000 0x19ffffff rw"
37    },
38    {
39      "text": "mem 0x40000000 0x41ffffff rw"
40    },
41    {
42      "text": "target remote 172.16.11.126:1025" // FIXME
43    },
44    {
45      // MUST use full path
46      "text": "load e:/AIC/luban-lite/luban-lite/output/d13x_demo88-nor_rt-thread_helloworld/images/d13x.elf" // FIXME
47    },
48    {
49      // MUST use full path
50      "text": "file e:/AIC/luban-lite/luban-lite/output/d13x_demo88-nor_rt-thread_helloworld/images/d13x.elf" // FIXME
51    },
52    {
53      // FIXME
54      "text": "b rt_hw_board_init" // 4.修改为自己的断点
55    },
56  ]

```

1.修改为当前方案的配置

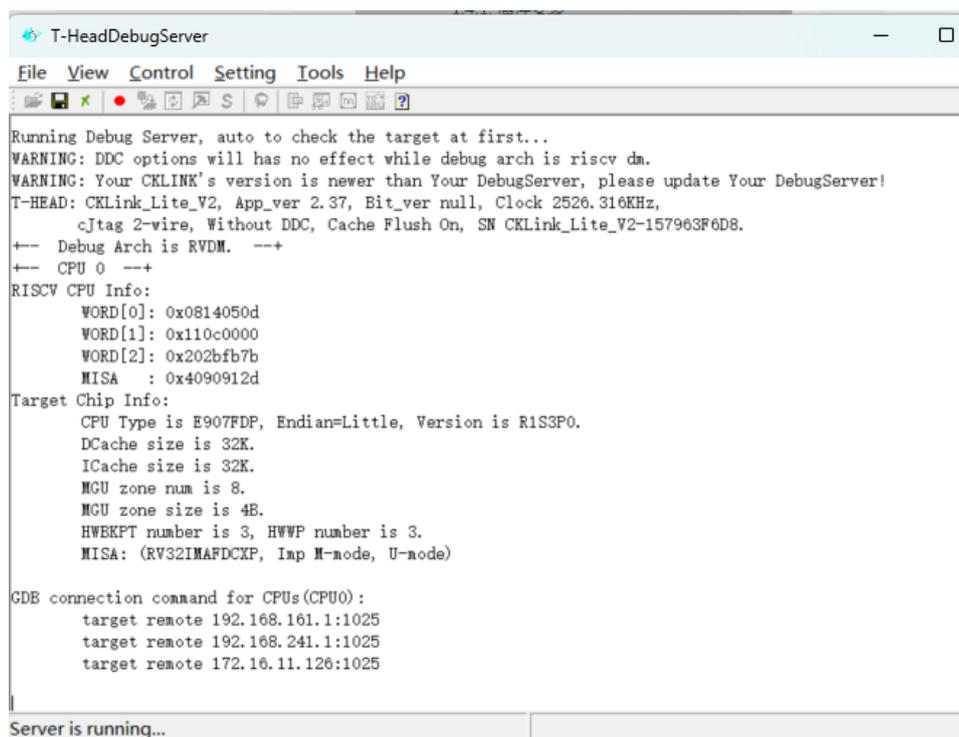
2.T-HeadDebugServer正常连接JTAG调试器后，会打印可连接的IP和端口

3.修改为当前方案的配置
注意：这里必须是全路径

Dxx connect only 需要修改的参数和上面类似，在 `launch.json` 文件中都用关键字 `FIXME` 标注。

5. 打开 DebugServer，确保已经成功连接 JTAG 调试器。

如果成功连接，则界面显示如下：



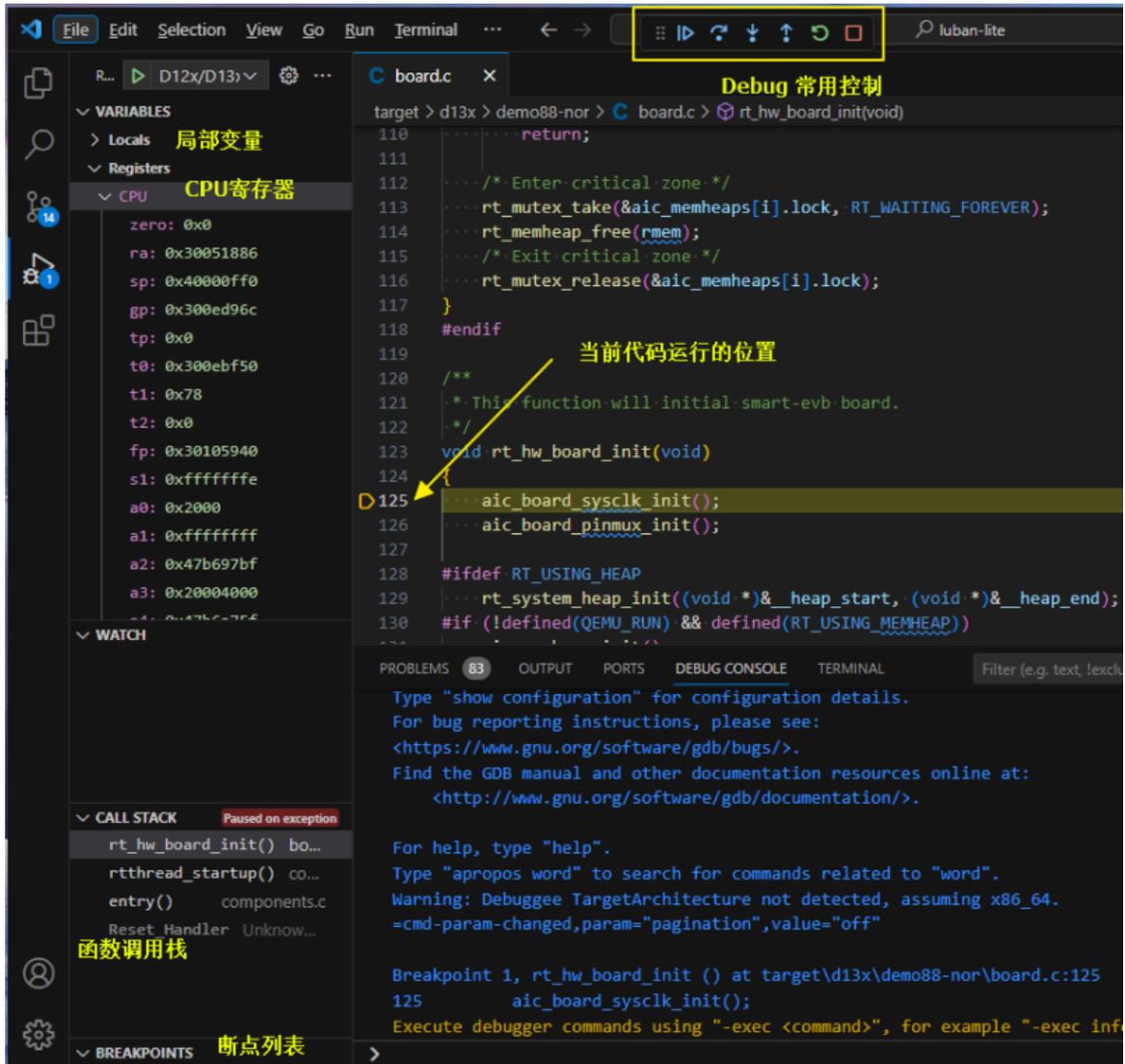
```

T-HeadDebugServer
File View Control Setting Tools Help
Running Debug Server, auto to check the target at first...
WARNING: DDC options will has no effect while debug arch is riscv dm.
WARNING: Your CKLINK's version is newer than Your DebugServer, please update Your DebugServer!
T-HEAD: CKLink_Lite_V2, App_ver 2.37, Bit_ver null, Clock 2526.316KHz,
        cJtag 2-wire, Without DDC, Cache Flush On, SN CKLink_Lite_V2-157963F6D8.
+-+ Debug Arch is RVDM. +-+
+-+ CPU 0 +-+
RISCV CPU Info:
WORD[0]: 0x0814050d
WORD[1]: 0x110c0000
WORD[2]: 0x202bfb7b
MISA : 0x4090912d
Target Chip Info:
CPU Type is E907FDP, Endian=Little, Version is R1S3P0.
DCache size is 32K.
ICache size is 32K.
MGU zone num is 8.
MGU zone size is 4E.
HWEKPT number is 3, HWWP number is 3.
MISA: (RV32IMAFDCXP, Imp M-mode, U-mode)
GDB connection command for CPUs(CPU0):
target remote 192.168.161.1:1025
target remote 192.168.241.1:1025
target remote 172.16.11.126:1025

Server is running...

```

如果连接成功，VSCode 会进入 Debug 界面，如下所示：



6. 按照实际需求，开始调试。

8. 文档资源

8.1. 文档中心

ArtInChip 文档中心可供用户在线查阅所有文档资源，官方网址：<http://aicdoc.artinchip.com>。

8.2. Gitee 下载

产品相关文档使用 Gitee 存储和管理，也是开源仓库，可以通过下面的链接进行下载：

```
git clone https://gitee.com/artinchip/docs.git
```

8.3. SDK 内嵌文档

SDK 仓库中内嵌了相关的使用说明文档，存放在 SDK 根目录中的 `doc` 文件夹中。

9. 教学视频

https://space.bilibili.com/3546578952390720?spm_id_from=333.1007.0.0

ArtInChip