Baremetal SDK 使用指南

SDK 指南文件

Version 2.3

修订日期： 2025-01-23

内容

ArtInChip

ArtInChip

1. SDK 编译

> *介绍不同编译环境下 SDK 的详细编译流程。*

开始编译之前，确保已安装 SConstruct 并准备代码。

1.1. 准备代码

Baremetal SDK 的代码托管于 Gitee 服务器中，为开源代码。本节描述了如何从对应的仓库地址中下载源代码。

1.1.1. Git

推荐通过 Git 用户端下载代码，可以实时关注服务器补丁和版本的发布：

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

1.1.2. HTTP

通过网络下载的方式可直接下载压缩包，详细步骤如下：

- 在 Gitee 上打开 Baremetal 仓库，地址为 。
- 在仓库主界面点击**克隆/下载**按钮。
- 在**克隆/下载**界面选择**下载 ZIP**。

1.1.3. 代码结构

1.1.3.1. 目录总览

代码下载完成后，在代码根目录执行 tree -L 2 命令，可以看到如下所示 SDK 目录结构：

> **注：**
> 以下目录结构仅为参考示例，具体以实际代码库为准。

```
tree -L 2
```

```
├── application      // 存放 APP 入口，helloword 应用的 main 函数
│   ├── baremetal
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
│   ├── freertos
│   ├── Kconfig
│   └── rt-thread
├── bsp //存放 BSP 代码，和 RT-Thread 原生的 bsp 目录功能相同
│   ├── artinchip //ArtInChip SoC 内部的 driver、hal 以及最小系统 sys 代码
│   ├── common     //公共头文件
│   ├── peripheral        //一些外设模块的驱动
│   ├── examples_bare     //各模块在裸机环境的示例代码
│   └── examples          //各模块在 RTOS 环境的示例代码
│       ├── test-alarm
│       ├── test-audio
│       ├── test-can
│       ├── test-dma
│       ├── test-qspi
│       ├── test-touch
│       └── test-uart
├── doc //Baremetal SDK 的介绍文档
│   ├── luban-lite_driver_development_guid.md  //设备驱动开发指南
│   ├── luban-lite_sdk_design.md //SDK 设计说明
│   ├── luban-lite_user_guid_linux.md //Linux 环境的用户使用说明
│   └── luban-lite_user_guid_windows.md //Windows 环境的用户使用说明
├── win_env.bat //启动 RT-Thread 的 env 工具，用于 Windows 环境的开发
├── win_cmd.bat //启动 CMD.exe，用于 Windows 环境的开发
├── kernel //存放各种 RTOS 内核
│   ├── baremetal
│   ├── common
│   ├── freertos
│   └── rt-thread
├── output //SDK 的编译输出
├── packages //组件包
│   ├── artinchip //ArtInChip 开发的组件包
│   └── third-party //第三方的组件包
├── target //方案（板）级的代码和配置
│   ├── configs //整个方案的 SDK 配置文件
│   ├── d12x     //D12x 开发板配置
│   │   ├── common
│   │   ├── demo68-nand
│   │   ├── demo68-nor
│   │   └── hmi-nor
│   ├── d13x     //D13x 开发板配置
│   │   ├── common
│   │   ├── demo88-nand
│   │   ├── demo88-nor
│   │   └── kunlunpi88-nor
│   ├── d21x     //D21x 开发板配置
│   │   ├── common
│   │   └── demo128-nand
│   └── g73x     //G73x 开发板配置
│       ├── common
│       └── demo100-nor
├── toolchain //解压后的工具链存放目录
└── tools //一些工具
    ├── onestep.sh  //ArtInChip 开发的 OneStep 命令行增强工具
    └── toolchain //工具链的压缩包
```

1.1.3.2. application

存放应用和各种示例：

```
├── baremetal           //baremetal 裸机
│   ├── bootloader      //系统的引导加载程序
│   └── helloworld      //系统示例
└── rt-thread
    └── helloworld      //系统示例
```

1.1.3.3. bsp

`bsp` 目录是 SoC 的核心，封装了该 SoC 的所有接口和驱动：

```
├── artinchip
│   ├── drv                 //驱动
│   ├── drv_nare            //裸机驱动
│   ├── hal                 //硬件抽象
│   ├── include
│   ├── SConscript
│   └── sys                 //最小系统
├── common
│   ├── crc32
│   ├── include
│   ├── partition
│   ├── private_param
│   ├── SConscript
│   └── utils
├── Kconfig
├── examples                //模块示例
│   ├── SConscript
│   ├── test-adc
│   ├── test-alarm
│   ├── test-audio
├── peripheral              //三方外设驱动封装
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
│   ├── Kconfig
│   ├── SConscript
│   ├── camera
│   ├── codec
│   ├── touch              //触控
│   └── wireless           //无线
└── SConscript
```

1.1.3.4. doc

遵从 RT-Thread 原始建议使用 Markdown 编写的 SDK 的简单设计和使用文档。

1.1.3.5. kernel

`kernel` 是 RTOS 系统的核心，该目录实现了 Baremetal 支持的三种运行方式的内核以及 AIC 公共接口：

- `baremetal`

  裸机系统的接口封装

- `freertos`

  FreeRTOS 内核的实现

- `rt-thread`

  RT-Thread 内核的实现

- `common`

  AIC 公共接口的封装

1.1.3.6. package

组件包，包括 ArtInChip 开发的组件和 三方组件，支持从 RT-Thread 在线下载：

```
├── artinchip          //ArtInChip 开发
│   ├── Kconfig
│   ├── aic-dm-apps
│   ├── lvgl-ui
│   ├── ota
│   ├── mpp
│   └── SConscript
├── Kconfig
├── SConscript
└── third-party        //三方组件
    ├── bonnie
    ├── cherryusb
    ├── cpu_usage
    ├── Kconfig
    ├── littlefs
    ├── lvgl
    ├── lwip
    ├── mklittlefs
    ├── ramdisk
    ├── SConscript
    └── uffs
```

1.1.3.7. target

存储方案（开发板）的配置信息：

- **`configs`**

  存放某一个方案（开发板）的功能配置信息：

  ```
  d12x_demo128-nand_baremetal_bootloader_defconfig    //bootlaoder 的配置
  d12x_demo128-nand_rt-thread_helloworld_defconfig    //rt-thread 的配置
  ```

  config 文件的内容示例：

  ```
  cat d12x_demo88-nor_rt-thread_helloworld_defconfig
  ```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
CONFIG_SOC_THEAD_SMART=y
# CONFIG_QEMU_RUN is not set
CONFIG_PRJ_CUSTOM_LDS=""
CONFIG_AIC_CHIP_D21X=y
CONFIG_CACHE_LINE_SIZE=64
CONFIG_AIC_CMU_DRV=y
CONFIG_AIC_CMU_DRV_V10=y
CONFIG_AIC_CMU_DRV_VER="10"
```

- **d12x**

  存放某一个方案（开发板）的参数配置信息，以代码的方式存在。

```
├── demo128-nand
│   ├── board.c
│   ├── include
│   ├── Kconfig.board
│   ├── pack
│   ├── pinmux.c
│   ├── SConscript
│   └── sys_clk.c
```

  - `Kconfig.board`：提供方案的外设选择项
  - `pinmux.c`：该方案的 pinmux 配置
  - `SConscript`：该方案目录的编译配置
  - `sys_clk.c`：该方案的 clk 配置
  - `include`：该方案特殊的 include 文件，主要是 `board.c` 的接口
  - `pack`：该方案在打包固件时的配置，如 ddr，分区等配置信息
  - `board.c`：该方案的初始化运行接口

1.1.3.8. toolchain

原始为空目录，用于存放解压后的工具链。

1.1.3.9. tools

该目录用于存放一些常用工具

- `onestep.sh`： Linux 上使用的 OneStep 脚本
- `env`： 环境搭建和运行需要的三方工具
- `script`： 编译和打包固件需要的脚本
- `toolchain`： 工具链包
  - `Xuantie-900-gcc-elf-newlib-x86_64-V2.6.1-20220906.tar.gz`: Linux 工具链
  - `Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-20220906.tar.gz`: Windows 工具链

1.1.3.10. ReleaseNote.md

该文件记录当前 SDK 版本号及主要更新内容，示例如下：

```
# V1.0.5 #
## 新增 ##
- 调屏：支持和 AiPQ V1.1.1 工具配合使用
- 新增 LVGL Demo：
- 支持 VSCode 模拟器的工程导入
- 增加压力测试、独立控件、Gif、slide、lv_ffmpeg、DVP 回显等参考实现
...
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

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 Baremetal 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 Baremetal 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。

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。

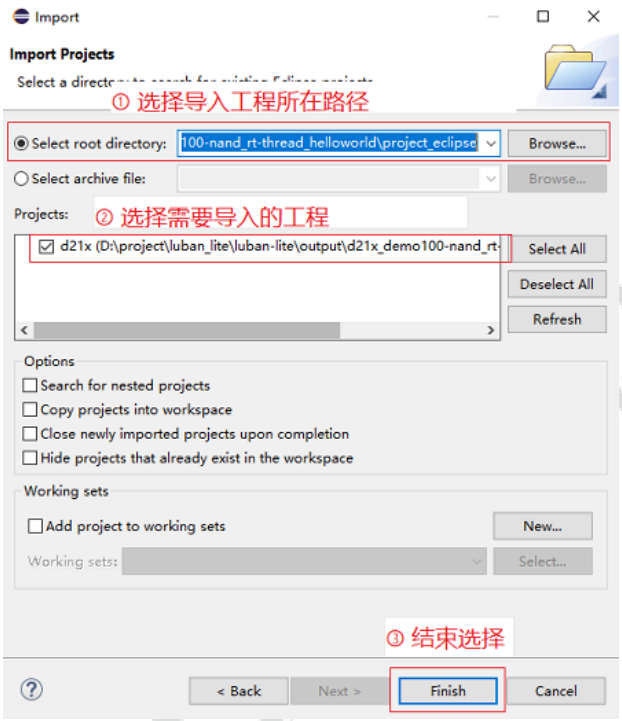
     该目录拷贝了所有必要文件，可以作为一份独立的 SDK 拷贝到任何路径下进行调试。

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：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

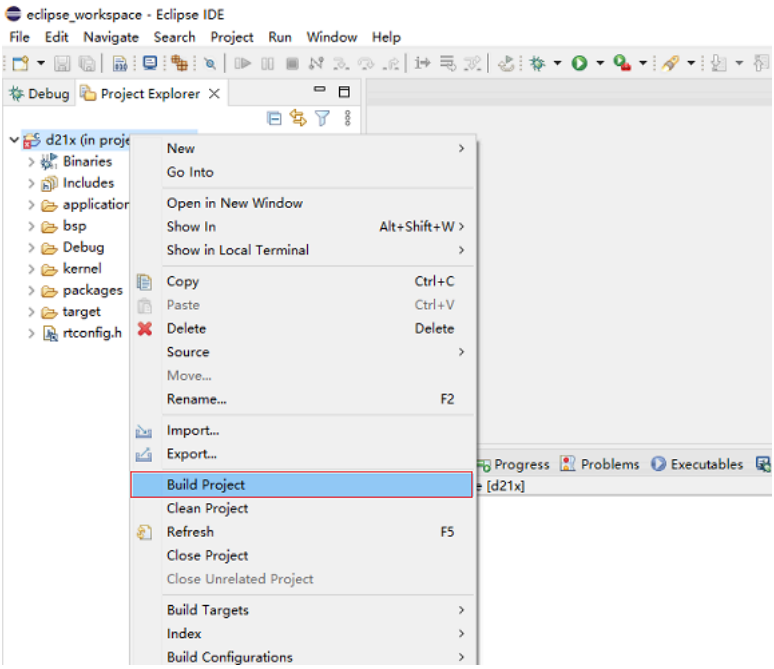


4. 选择 **Finish** 完成导入。

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。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

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

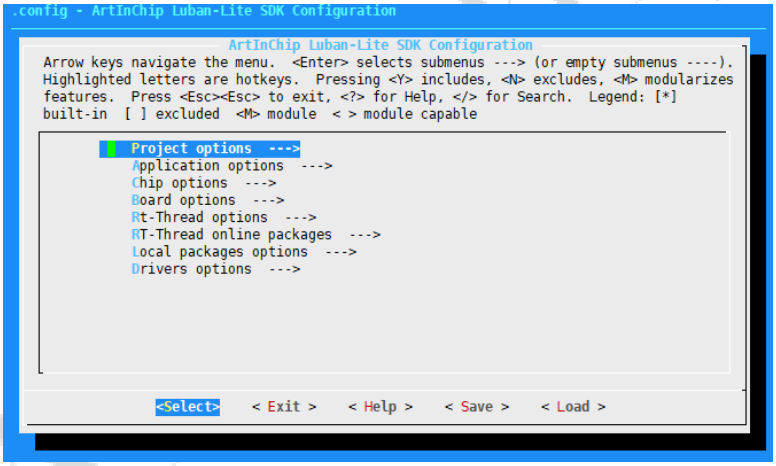
```
ll output/d21x_demo100-nand_rt-thread_helloworld/project_eclipse/Debug/
d21x.bin
d21x.elf                              //调试需要的 elf 文件
d21x.map
d21x_demo100_nand_page_2k_block_128k_v1.0.0.img     //烧录需要的 img 文件
```

1.2.4. 更改 Eclipse 工程配置

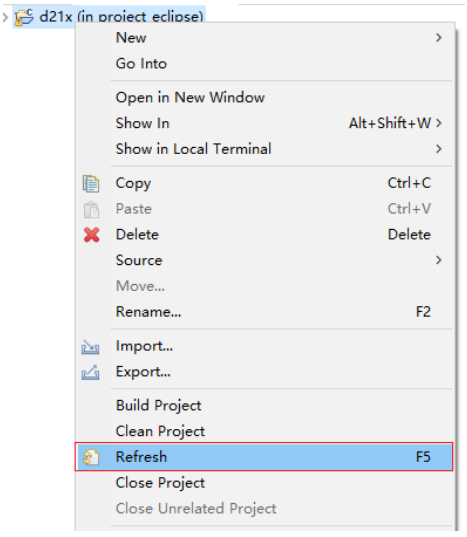
对于使用 scons --target=eclipse 命令生成的 Eclipse 工程，可以通过 menuconfig 菜单来更新工程的配置。

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：



1.3. Ubuntu

Baremetal SDK 的开发可以在 Linux 系统中进行，Baremetal 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。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

1.3.1. 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。

1.4. VSCode

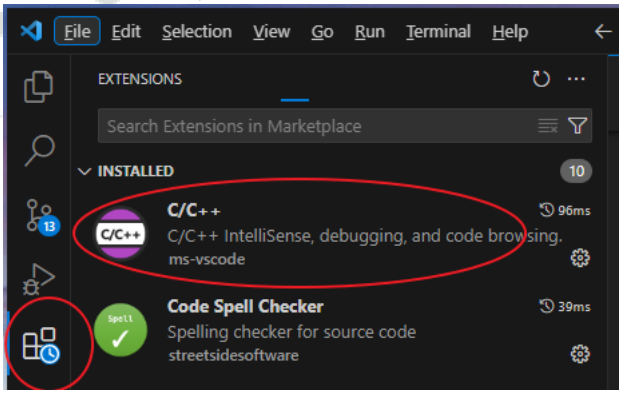
Baremetal 支持在 VSCode 环境中完成全流程的开发，包括代码编辑、编译、调试和烧写。VSCode 是一款开源、免费、跨平台的源代码编辑器，由 MicroSoft 开发，特点是轻量级、高性能和可扩展。

> **注：**
> 仅在打开 Baremetal 根目录的前提下，VSCode 才能完成以下的编译、调试和烧写操作。

1.4.1. 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。

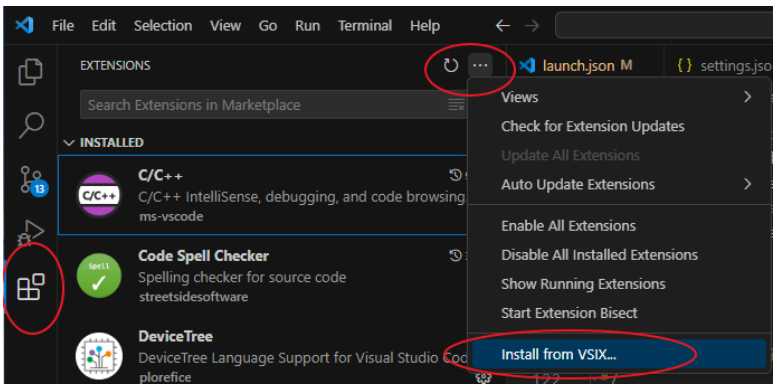


- 在 PC 未联网的条件下，则需要按照以下步骤，手动完成安装：

  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：



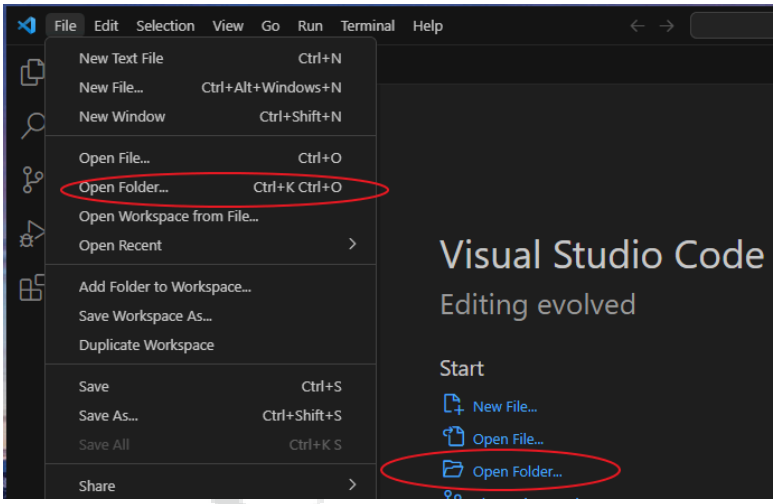
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

1.4.2. 打开工程文件

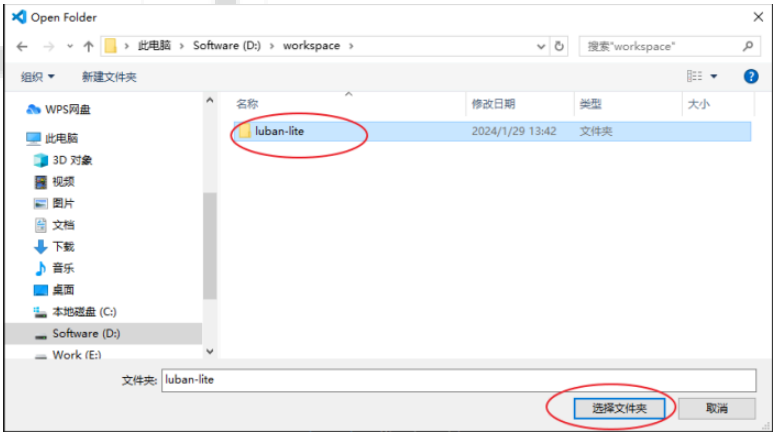
1. 选择以下任意方式打开一个文件夹目录：

   • 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...**



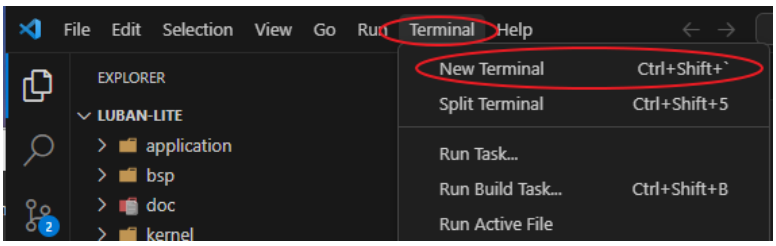
2. 在弹出的文件夹浏览窗口中选择 Baremetal 根目录，并点击**选择文件夹**：



1.4.3. 编译

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

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



   可使用快捷键 **Ctrl+Shift+`**。

2. 在 VSCode 终端窗口中输入并执行 win_cmd.bat。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
E:\AIC\luban-lite\luban-lite>win_cmd.bat
Luban-Lite command for Windows

Microsoft Windows [Version 10.0.22631.3593]
(c) Microsoft Corporation。保留所有权利。

E:\AIC\luban-lite\luban-lite>
```

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

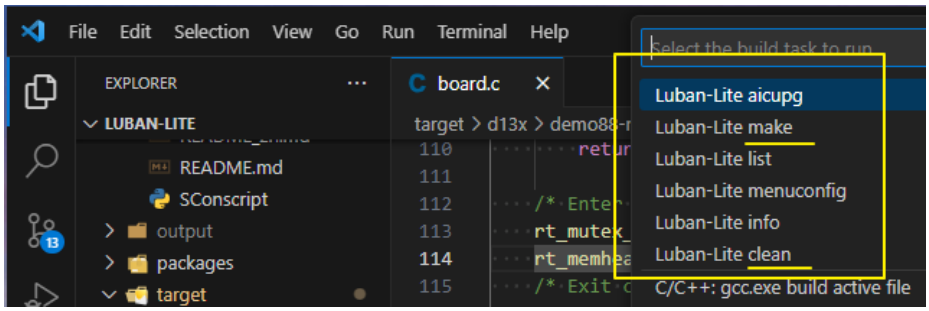
```
E:\git\luban-lite>lunch 3
scons: Reading SConscript files ...
Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig
```

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 **Baremetal make**。

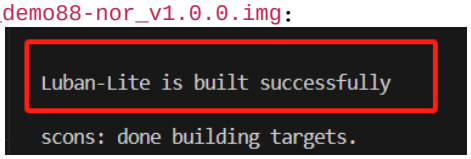
6. 使用以下任意一种方式，可以清除工程：

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



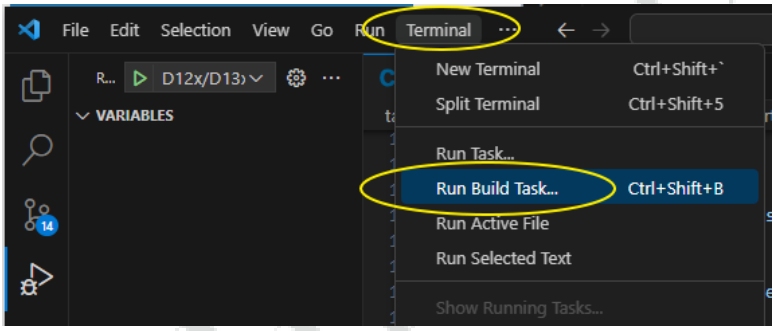
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

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`：



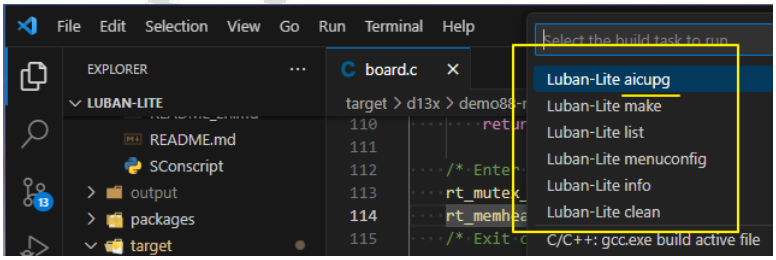
1.4.4. 烧写

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 **Baremetalaicupg** 执行烧写操作。

1.4.5. 快捷命令清单



如图所示，Baremetal 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

1.5. VMWare 安装

> *如需在虚拟机上进行 Baremetal 的开发工作，可以参考本节的安装说明。*

虚拟机（Virtual Machine）指通过软件模拟的、具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统，是 Windows 系统上运行 Linux 软件的一种通用解决方案。

常用的 Windows 上的虚拟机有 VWMWare，VirtualBox 等，本节以 VMWare 为例讲解如何在 Windows 系统上搭建 Baremetal 编译环境。虚拟机需要额外的系统资源资源进行软件模拟工作， 因此在虚拟机上进行 Baremetal 系统的开发和编译工作将需要更久的时间，也会偶尔造成系统使用的卡顿。

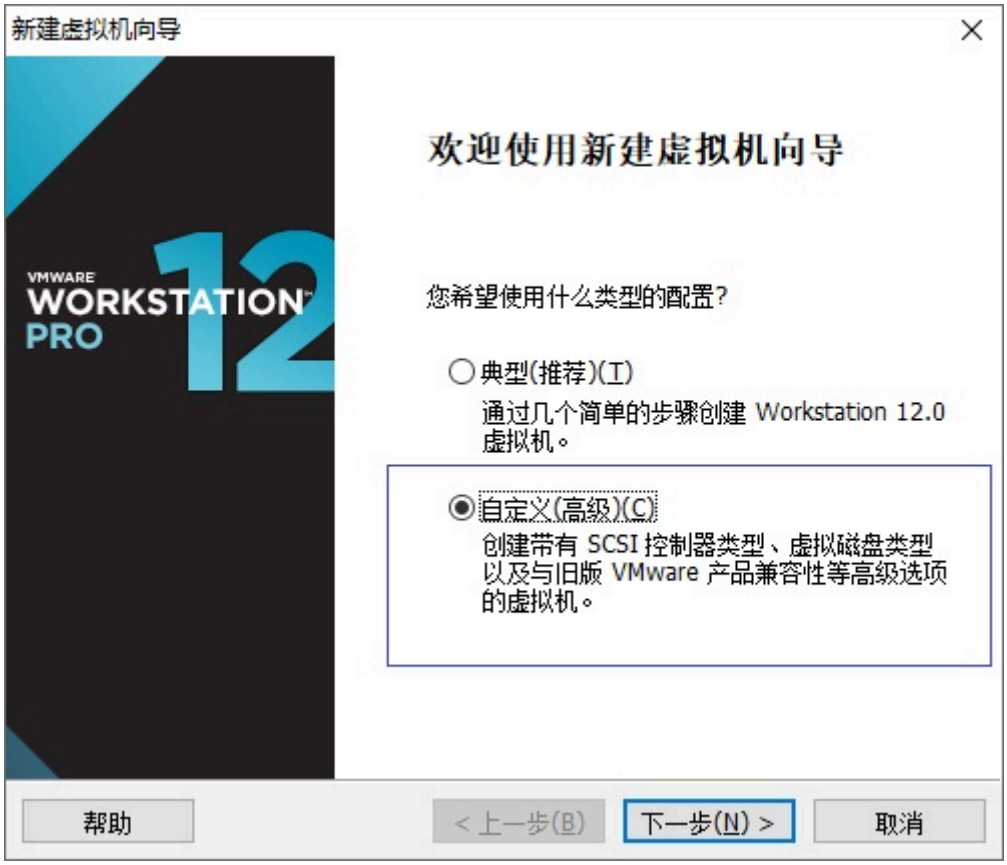
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

VMWare 为收费软件，本文以试用版本（VMware® Workstation 16 Pro）为例讲解。

1.5.1. 创建虚拟机

1.5.1.1. 自定义安装

在新建虚拟机的向导页选择**自定义安装**，并完成 VMWare 和平台关联的定制设置。

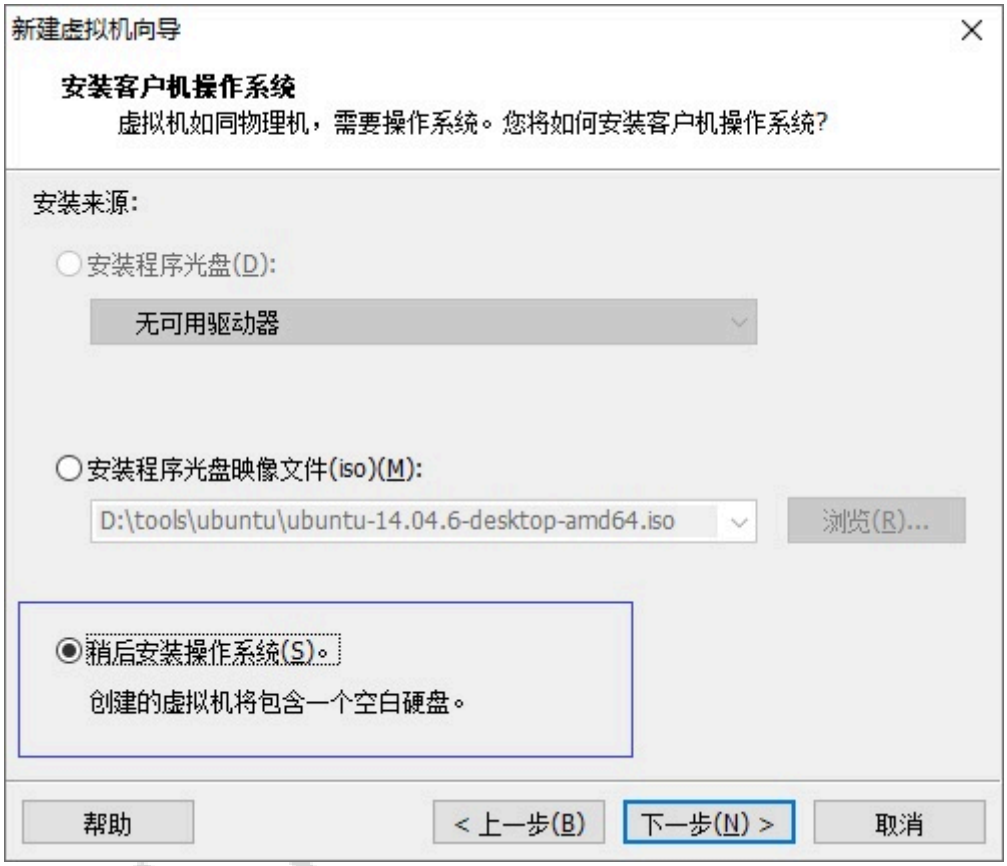


1.5.1.2. 手动安装操作系统

在操作系统安装界面选择**稍后安装操作系统**。

VMWare 在自动安装操作系统时会顺带安装自带的 VMware tools， 但该 tools 以 `iso` 方式存在，需要虚拟光驱方式挂载出来才能安装。该挂载任务只能在系统启动后手动执行，所以自动安装往往会卡在 VMWare tools 的安装处而导致无法安装成功。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

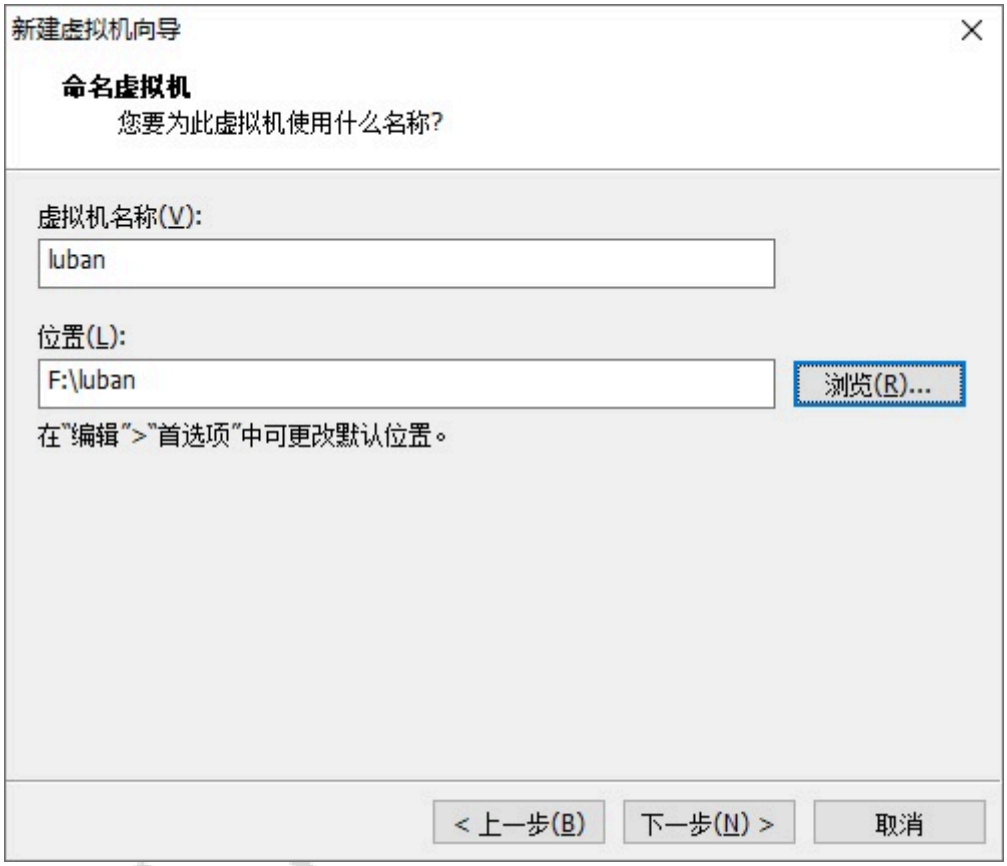


1.5.1.3. 虚拟机文件存储

创建一个新目录存储虚拟机文件。

- 虚拟机文件将被拆分为众多 2G 大小的文件存储，使用单独目录便于管理。
- 虚拟机文件将占用超过 20G 的磁盘空间，且在安装新软件后所占空间也将继续增大，确保预留足够存储空间。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



1.5.1.4. 资源设置

在软件的安装过程种，需要设置可能会影响虚拟机使用体验的参数，建议的参数及其配置如下图所示：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

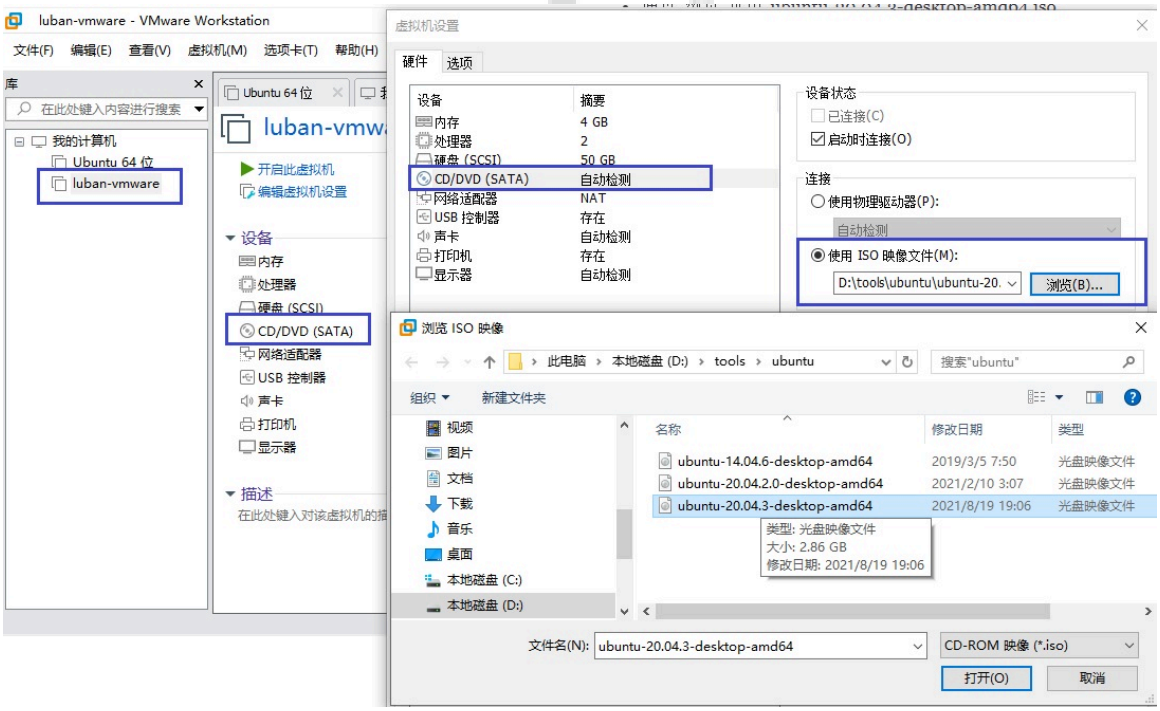
1.5.2. 安装 Ubuntu

本章节介绍在虚拟机上安装 Ubuntu 的方法，以 Ubuntu20.04 版本为例。如使用其他版本，安装方式可能会有差异。

1.5.2.1. 全新安装

1. 设置 Ubuntu 文件路径：

   - 打开的 VMWare
   - 选择**我的计算机**中的 Baremetal 虚拟机
   - 在**设备**中点击 **CD/DVD**。
   - 选中使用 ISO 映像文件。
   - 通过**浏览**选中 `Ubuntu-20.04.3-desktop-amd64.iso`。



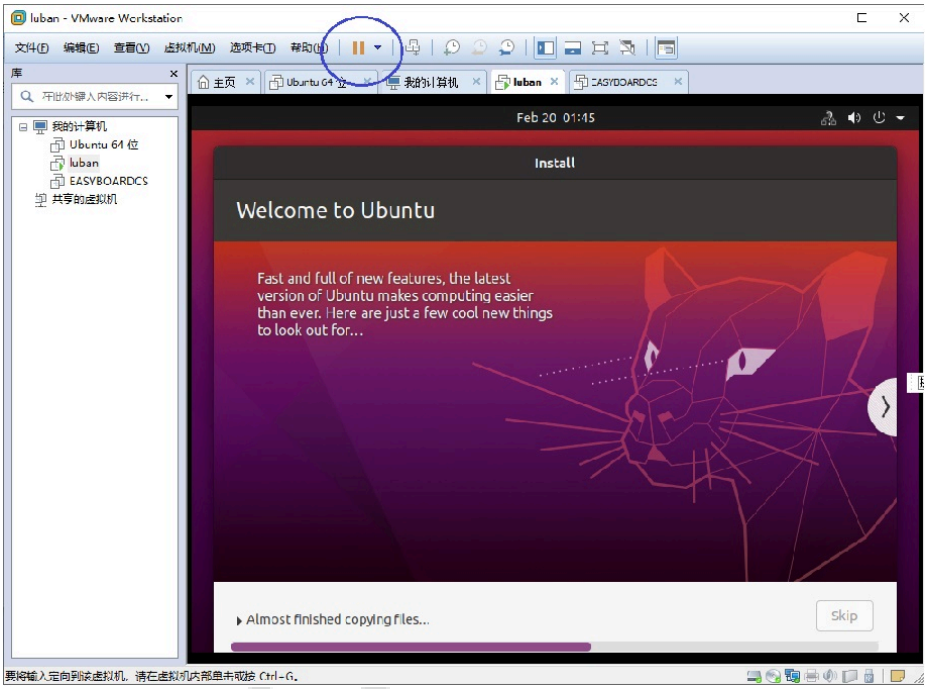
   完成此步骤后，光驱中已经放置了 Ubuntu–20.04 的光盘，并设置了开机光驱引导。

2. 打开 `Ubuntu-20.04.3-desktop-amd64.iso` 虚拟机文件，启动 Ubuntu 安装。

   在 Ubuntu 的安装过程中，如遇到问题，可通过浏览器在互联网上查找相关解决方案。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3. 安装完毕后，取消**使用 ISO 映像文件**，选择**使用物理驱动器**后重新启动虚拟机，可切换到硬盘启动加载已经安装的 Ubuntu 系统。



1.5.2.2. 加载虚拟机

如已获取 ArtInChip 发布的可直接工作的虚拟机镜像，例如`luban-ubuntu 20.04.vmx` 文件，可点击**文件 > 打开**。

通过导入虚拟机镜像的方式加载虚拟机后，需要重新配置使用资源，包括 CPU、Memory 和磁盘。

1.5.3. 配置虚拟机

Ubuntu 系统安装完成后，需要执行本节描述的配置流程。

在完成配置流程后，打开的系统才是拥有完整功能的 Ubuntu20.04 系统。关于如何准备 Baremetal 的编译环境，可参考 Ubuntu。

1.5.3.1. VMWare Tools

VMWare Tools 是 VMWare 提供的一个工具包，必须安装。

除了自带的工具外，VMWare Tools 还可以帮助设置共享和窗口大小，具体安装流程如下所示：

1. 在 VMWare 的**虚拟机**菜单中，点击**安装 vmware tools**。
2. 在 Ubuntu 系统中，打开**文件浏览器**工具，并找到挂载的 `VMWare Tools` 光驱。
3. 打开 `VMWare Tools` 光驱文件目录，并复制 `VMwareTools-<version>.gz` 文件到系统中的任意目录。
4. 解压缩 `VMwareTools-<version>.gz` 后，使用以下命令运行安装脚本：

   ```
   sudo perl vmware-install.pl
   ```

5. 根据提示和安装脚本进行交互，确认相关设置。如果遇到错误，需再次执行一次安装。

   执行安装过程中，会重复出现并不断修复兼容性错误。

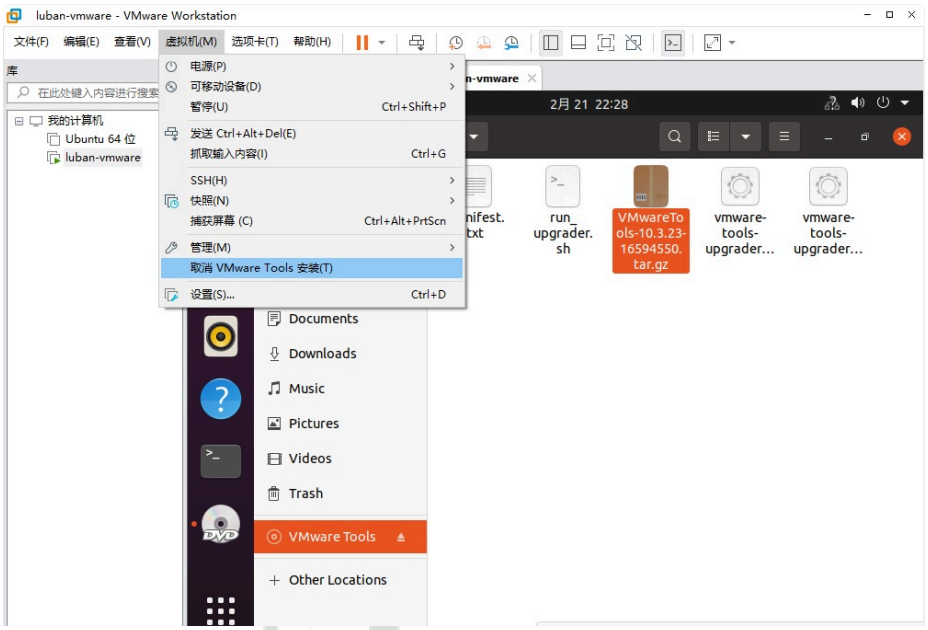
   当 VMWareTools 输出 enjoy the vmware team 提示信息，表示安装成功。

6. 在 VMWare 的**查看**菜单中，选择**自动调整大小 > 自动适应用户机**。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

7. 重启 Ubuntu。

   重启后，屏幕大小可调，表示 VMWareTools 配置成功。



1.5.3.2. 文件共享

虚拟机和 Windows 的文件共享配置是使用虚拟机进行开发的基础。

1. 配置 VMWare 端文件共享：

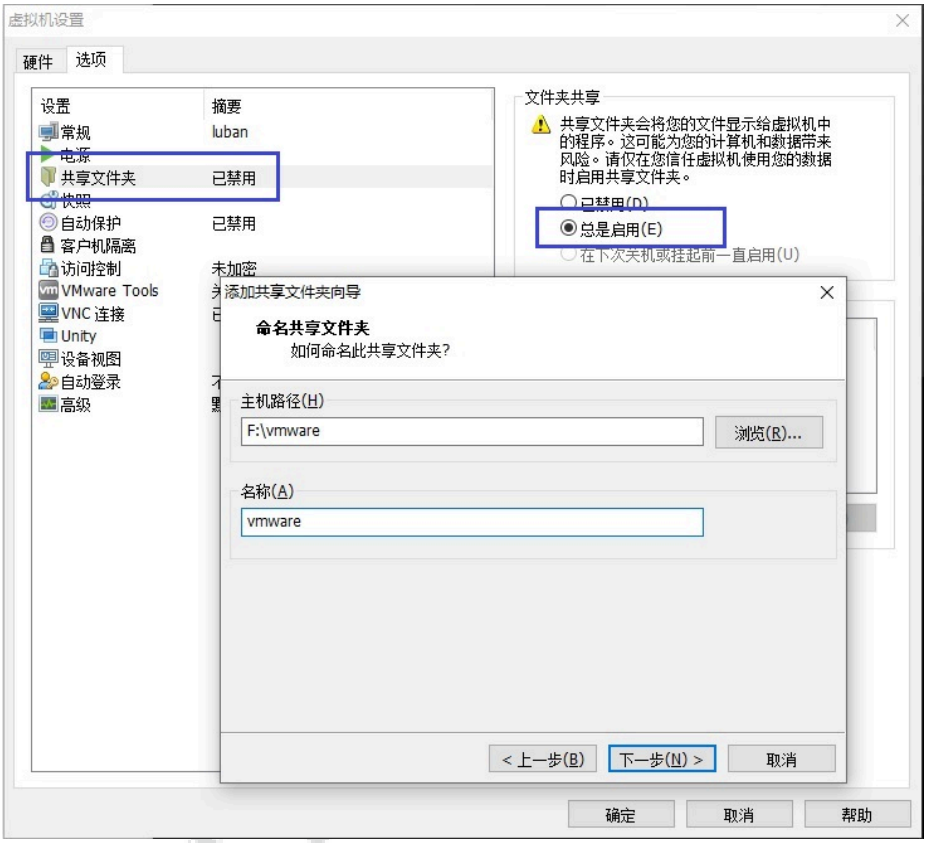
   a. 在 VMWare 的**虚拟机**界面，打开**设置**项。

   b. 在**选项**页，选择**共享文件夹**。

   c. 在右侧配置页面，选择**总是启用**。

   d. 在**主机路径**处，选择 Windows 上用于共享的文件夹，并在**名称**输入框中命名共享文件。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



2. 配置 Ubuntu 端文件共享

   如果在 VMWare Tools 安装过程中，正确打开了文件共享，则 `/mnt/hgfs/` 目录中会挂载出共享的目录。

1.6. 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。

Baremetal 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。

1.6.1. 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**

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

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
```

1.6.2. 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
```

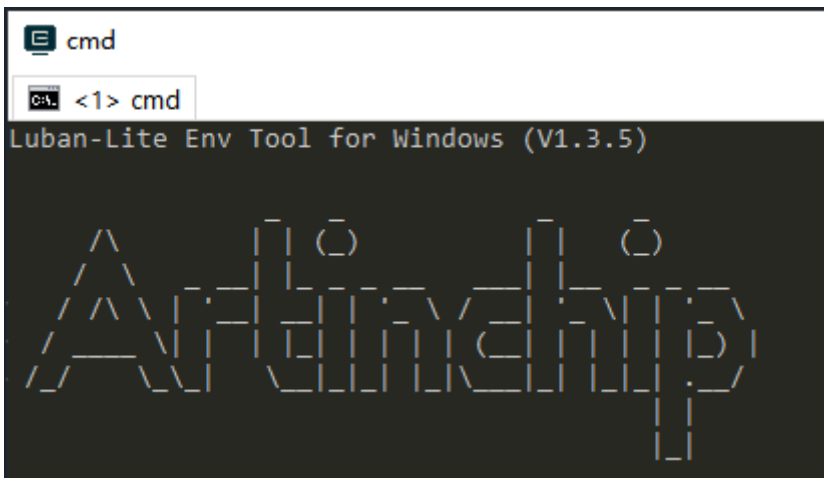
1.6.3. 批处理文件

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



Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- **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。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2. 使用指南

> *系统镜像、编译选项、开发板、应用等相关的详细使用说明。*

2.1. 命令详解

本节介绍 Baremetal SDK 编译过程中涉及的命令类型以及各个命令的详细说明。

2.1.1. scons 命令参考指南

SCons 是一个跨平台的构建系统，被广泛应用于软件开发项目中，支持在 Windows 和 Linux 上使用，且用法相同。本节列示了常用的 SCons 命令和功能。

2.1.1.1. scons --list-def

解压缩 SDK 后，使用 scons --list-def 命令可以罗列 SDK 中发布的所有项目名称，方便用户查看和管理可用的项目资源。示例如下：

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

输出示例结果如下：

```
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
```

2.1.1.2. scons --apply-def=<项目索引或名称>

使用命令 scons --apply-def=[项目索引或名称] 可以选择特定的项目配置进行构建，命令中可以使用项目名称或项目索引完成构建：

- 使用数字索引，例如 --apply-def=3，即应用列表中的第三个配置（从 0 开始计数）。以下示例表示应用名为 `d12x_demo68-nor_rt-thread_helloworld_defconfig` 的配置。

  ```
  scons --apply-def=3
  ```

  ```
  scons: Reading SConscript files ...
  Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig

  scons --apply-def=d12x_demo68-nor_rt-thread_helloworld_defconfig
  scons: Reading SConscript files ...
  Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig
  ```

- 直接使用配置名称，例如 --apply-def=d12x_demo68-nor_rt-thread_helloworld_defconfig。

  ```
  scons --apply-def=d12x_demo68-nor_rt-thread_helloworld_defconfig
  ```

  ```
  scons: Reading SConscript files ...
  Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig
  ```

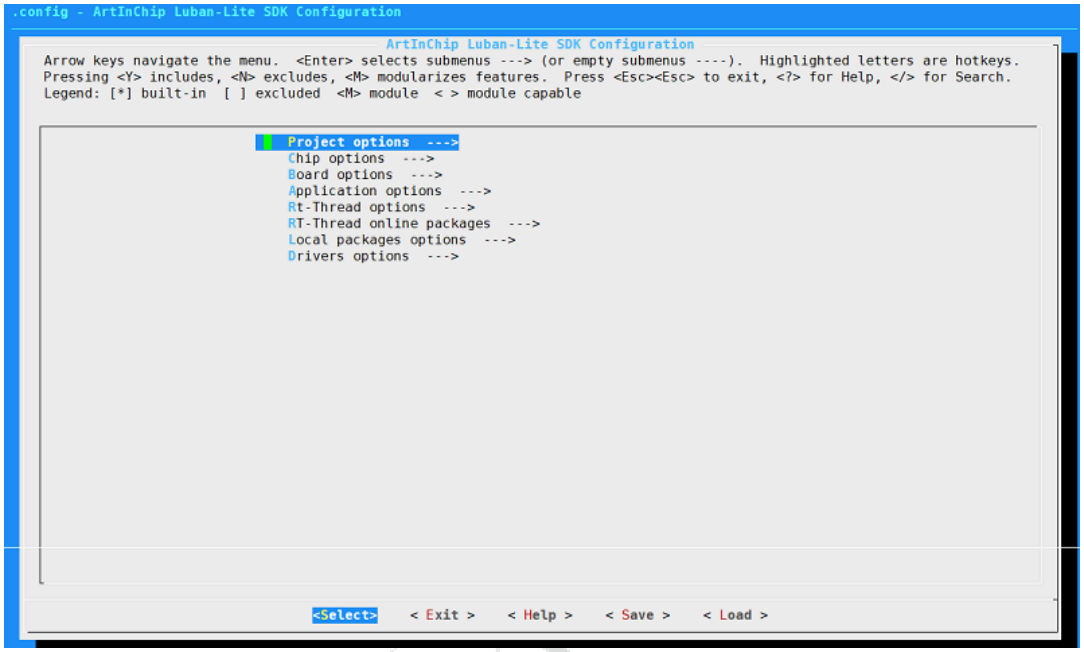
2.1.1.3. scons --menuconfig

使用命令 scons --menuconfig 可以调出 menuconfig 图形界面，以便根据需要调整项目设置。

```
scons --menuconfig
```

menuconfig GUI 界面示例如下：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



完成配置后，保存并退出配置界面，SCons 会根据所做的更改重新编译项目。

> **注：**
> 关于 menuconfig 的更多信息，可查看 menuconfig 命令工具参考指南。

2.1.1.4. scons

使用命令 scons 开始构建过程，编译成功后生成镜像文件。默认在简洁模式下进行构建，不会输出编译选项等信息。

```
scons
```

根据下列打印信息示例，编译成功后生成的文件路径为 `e:\luban-lite\output\d12x_demo68-nor_rt-thread_helloworld\images\d12x_demo68-nor_v1.0.0.img`：

```
Creating e:\luban-lite\output/d12x_demo68-nor_rt-thread_helloworld/images/usbupg-ddr-init.aic ...
Creating e:\luban-lite\output/d12x_demo68-nor_rt-thread_helloworld/images/bootloader.aic ...
Creating e:\luban-lite\output/d12x_demo68-nor_rt-thread_helloworld/images/os.aic ...
Image header is generated.
Meta data for image components:
            Meta for image.updater.psram       offset 0x1600     size 0x6010 (24592)
            Meta for image.updater.spl         offset 0x7e00     size 0x2d310 (185104)
            Meta for image.info                offset 0x0         size 0x800 (2048)
            Meta for image.target.spl          offset 0x35600    size 0x2d310 (185104)
            Meta for image.target.os           offset 0x62e00    size 0xd8800 (886784)
            Meta for image.target.rodata       offset 0x13b600   size 0x2b000 (176128)
            Meta for image.target.data         offset 0x166600   size 0x700000 (7340032)
    Packing file data:
            uartupg-psram-init.aic
            bootloader.aic
            bootloader.aic
            d12x_os.itb
            rodata.fatfs
            data.lfs
    Image file is generated: e:/luban-lite/output/d12x_demo68-nor_rt-thread_helloworld/images/d12x_demo68-nor_v1.0.0.img

Luban-Lite is built successfully

scons: done building targets.
```

2.1.1.5. scons --verbose

使用 scons --verbose 可以显示更详细的编译信息，包括编译器选项等，便于调试。

2.1.1.6. scons --clean

运行命令 scons --clean 清除上一次编译构建过程中生成的所有临时文件，确保在下一次编译构建时，所有源文件都会被重新编译。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
scons --clean
```

或者使用缩写命令：

```
scons -c
```

2.1.1.7. scons --save-def

保存当前板卡默认配置。

2.1.1.8. scons --info

显示当前工程信息，可使用简洁命令 scons i。

2.1.1.9. scons distclean

清除工具链和输出目录。

2.1.1.10. scons target=<*TARGET*>

生成目标工程，例如 `eclipse/eclipse_sdk`，其中 <*TARGET*> 是需要生成的目标工程名称。

2.1.1.11. scons genconfig

通过 `rtconfig.h` 生成 `.config` 文件。

2.1.1.12. scons useconfig=<*USECONFIG*>

通过配置文件生成 `rtconfig.h`，其中 <*USECONFIG*> 是需要使用的配置文件路径。

2.1.1.13. scons --run-qemu

运行当前编译出来的 qemu 目标文件。

2.1.1.14. scons --list-size

size 命令列出所有 `.o` 文件下 `text/data/bss` 中各个 section 的大小。

2.1.1.15. scons --pkgs-update

下载选择的在线 packages。

2.1.2. OneStep 命令参考指南

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。根据操作系统的不同，可能需要不同的设置：

- **Windows**系统设置

  在 Windows 系统中，OneStep 自动集成到 `win_cmd.bat` 和 `win_env.bat` 批处理脚本中。

  在 CMD 或者 ENV 窗口启动后，OneStep 命令已经生效，在其中可以从任意目录执行 OneStep 命令。

- **Linux** 系统设置

  在 Linux 系统中，需要将 OneStep 脚本添加到当前路径中，如执行以下命令：

  ```
  source tools/onestep.sh
  ```

  在 Ubuntu 终端中，进入 SDK 根目录后，使用 source tools/onestep.sh 命令即可查看所有常见命令。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

为了方便使用并加快开发效率，ArtInChip 开发了一系列的 OneStep 子命令，如h|help显示帮助信息以及lunch选择和启动指定的项目配置。

```
h
```

```
Baremetal SDK OneStep commands:
    hmm|h                    : Get this help.
    lunch          [keyword] : Start with selected defconfig.e.g. lunch mmc
    menuconfig|me            : Config application with menuconfig
    bm                       : Config bootloader with menuconfig
    km                       : Config application with menuconfig
    m|mb                     : Build bootloader & application and generate final image
    ma                       : Build application only
    mu|ms                    : Build bootloader only
    c                        : Clean bootloader and application
    mc                       : Clean & Rebuild all and generate final image
    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.
    godir|gd       [keyword] : Go/jump to selected directory.
    list                     : List all SDK defconfig.
    list_module              : List all enabled modules.
    i                        : Get current project's information.
    buildall                 : Build all the *defconfig in target/configs
    rebuildall               : Clean and build all the *defconfig in target/configs
    addboard|ab              : Add new board *defconfig in target/configs
    aicupg                   : Burn image file to target board
```

2.1.2.1. list

查看所有项目文件。

> **注：**
>
> list 命令进行了显示项目精简，隐藏了 `bootloader` 的项目文件。

```
list
```

输出示例如下：

```
scons: Reading SConscript files ...
Built-in configs:
d12x_demo68-nand_rt-thread_helloworld_defconfig
0. d12x_demo68-nor_rt-thread_helloworld_defconfig
1. d12x_hmi-nor_rt-thread_helloworld_defconfig
2. d13x_demo68-nor_rt-thread_helloworld_defconfig
3. d13x_demo88-nand_rt-thread_helloworld_defconfig
4. d13x_demo88-nor_rt-thread_helloworld_defconfig
5. d13x_kunlunpi88-nor_rt-thread_helloworld_defconfig
6. d21x_d215-demo88-nand_rt-thread_helloworld_defconfig
7. d21x_d215-demo88-nor_rt-thread_helloworld_defconfig
8. d21x_demo100-nor_rt-thread_helloworld_defconfig
9. d21x_demo128-nand_rt-thread_helloworld_defconfig
10.g73x_demo100-nor_rt-thread_helloworld_defconfig
11.g73x_demo68-nor_rt-thread_helloworld_defconfig
12.g73x_scan_rt-thread_helloworld_defconfig
```

2.1.2.2. lunch

选择、加载或应用具体配置方案。

- 示例 1：进入 Recovery 系统 SDK 生产环境

  ```
  lunch ota_emmc
  ```

- 示例 2：加载 `d13x_demo88-nor-xip_rt-thread_helloworld_defconfig` 配置文件

  ```
  lunch d13x_demo88-nor-xip_rt-thread_helloworld_defconfig
  ```

2.1.2.3. menuconfig|me

打开 menuconfig 工具界面，修改 RT-Thread 配置

```
menuconfig
```

或

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
me
```

2.1.2.4. bm

打开 menuconfig 工具界面，修改 BootLoader 配置

```
bm
```

2.1.2.5. km

打开 menuconfig 工具界面，修改应用配置

```
bm
```

2.1.2.6. m

编译 SDK 固件，并生成镜像文件。

```
m
```

> **注：**
> 与 mb 命令含义相同。

2.1.2.7. mb

编译 bootloader 和应用，并生成最终镜像文件。

```
mb
```

2.1.2.8. ma

仅编译应用，并生成最终镜像文件。

```
ma
```

2.1.2.9. mulms

仅编译 bootloader，并生成最终镜像文件。

```
mu|ms
```

2.1.2.10. c

清除 bootloader 和应用配置。

```
c
```

2.1.2.11. mc

清除并重新编译，重新生产所有最终镜像文件。

```
mc
```

2.1.2.12. crootlcr

进入 SDK 根目录。

2.1.2.13. coutlco

进入编译输出目录

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.1.2.14. ctarget|ct

回到方案或目标开发板目录。

2.1.2.15. godir|gd

跳转到指定目录。

2.1.2.16. i

查看当前项目的配置信息。

```
i
```

```
scons: Reading SConscript files ...
Target app: application/rt-thread/helloworld
Target chip: d12x
Target arch: riscv32
Target board: target/d12x/demo68-nor
Target kernel: kernel/rt-thread
Defconfig file: target/configs/d12x_demo68-nor_rt-thread_helloworld_defconfig
Root directory: xxxxxxxxxxxxx
Out directory: output/d12x_demo68-nor_rt-thread_helloworld
    Toolchain: toolchain/bin\riscv64-unknown-elf-
```

2.1.2.17. buildall

编译目标配置文件 `target/configs` 中的所有 `*defconfig` 文件。

2.1.2.18. rebuildall

清除前一次配置信息，并重新编译目标配置文件 `target/configs` 中的所有 `*defconfig` 文件。

2.1.2.19. addboard|ab

在 `target/configs` 中添加新开发板的 `*defconfig` 文件。

2.1.2.20. aicupg

将镜像文件烧录到目标开发板中。

2.2. 设计框架

2.2.1. 设计目标

Baremetal 的设计规划旨在提供一个简单易用，且能够满足广泛用户需求的解决方案，同时要确保对主流实时系统的支持。Baremetal 具有以下设计优势和特点：

- 兼容多种市面上最流行的 RTOS 内核，包括 RT-Thread、FreeRTOS 等
- 支持 baremetal 模式
- 提供完整的软件栈生态资源

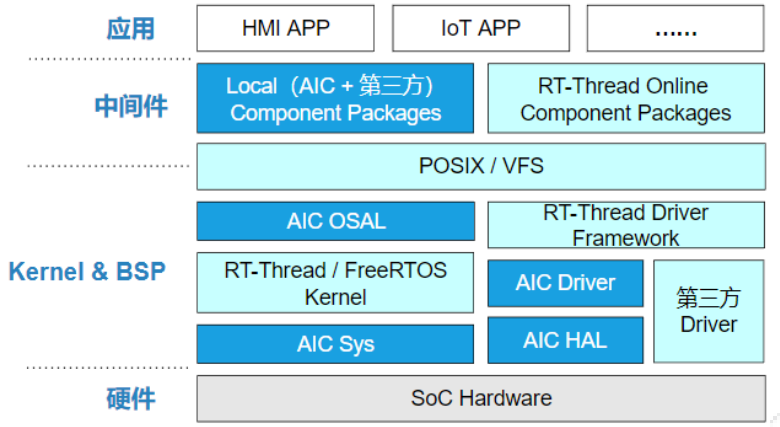
2.2.2. 设计框架

根据是否使用 OS，Baremetal SDK 架构分为两种情况：

RTOS

在使用操作系统的情况下，Baremetal SDK 则能够在 RTOS 环境下运行。通过在 RT-Thread Kernel 的基础上进行封装，Baremetal 能够兼容 RT-Thread 和 Free-RTOS 两种实时操作系统的 API：

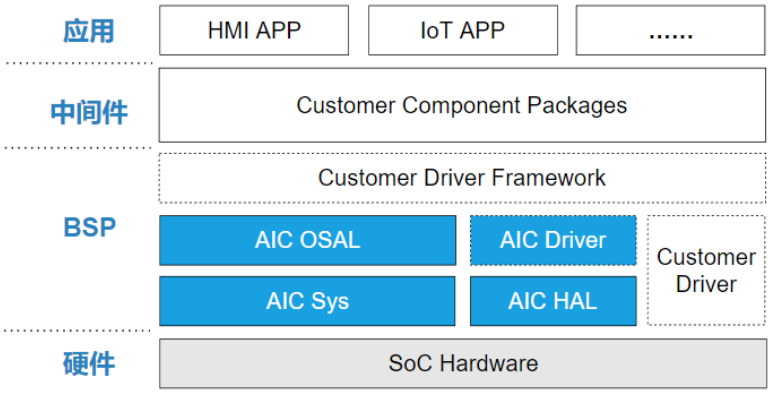
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



基于RTOS的Luban-Lite软件框架

Baremetal

在不使用操作系统的情况下，Baremetal SDK 支持 baremetal 模式，允许开发者在没有操作系统的环境中运行应用程序。



基于baremetal的Luban-Lite软件框架

2.2.3. 四级抽象模型

Baremetal 是一个多维抽象模型的跨平台 SDK，包括软、硬件平台，支持多种应用场景，并能处理以下多种复杂映射关系：

- 多个 SoC 芯片，需要进行驱动和设备的分离、驱动实例化等操作。
- 多块单板，每块板子的外设、IO、性能配置都有所不同。
- 多种应用，一块板子可能支持多个应用运行。
- 若干组件，驱动、组件、应用的对应存在一对多的依赖关系。

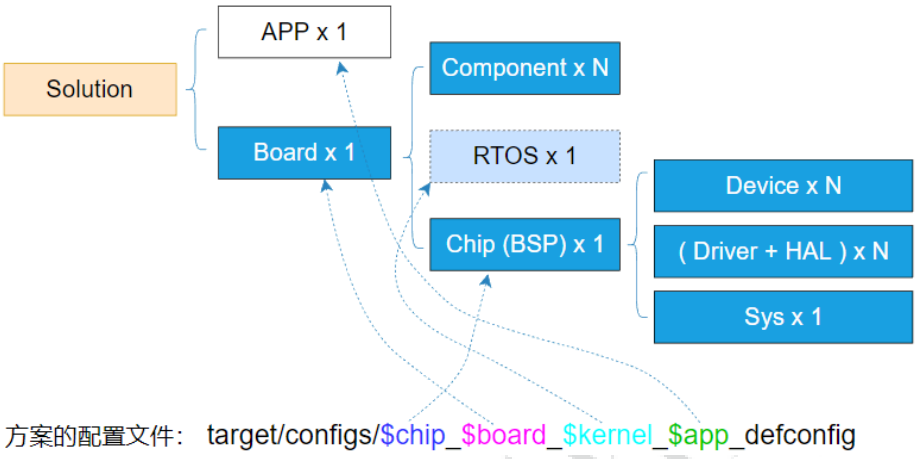
总体上，以上元素交织在一起，形成了复杂的 $N \times N \times N$ 的多对多组合关系。

在满足以上复杂映射关系的基础上，Baremetal SDK 的设计具有以下易用性特点：

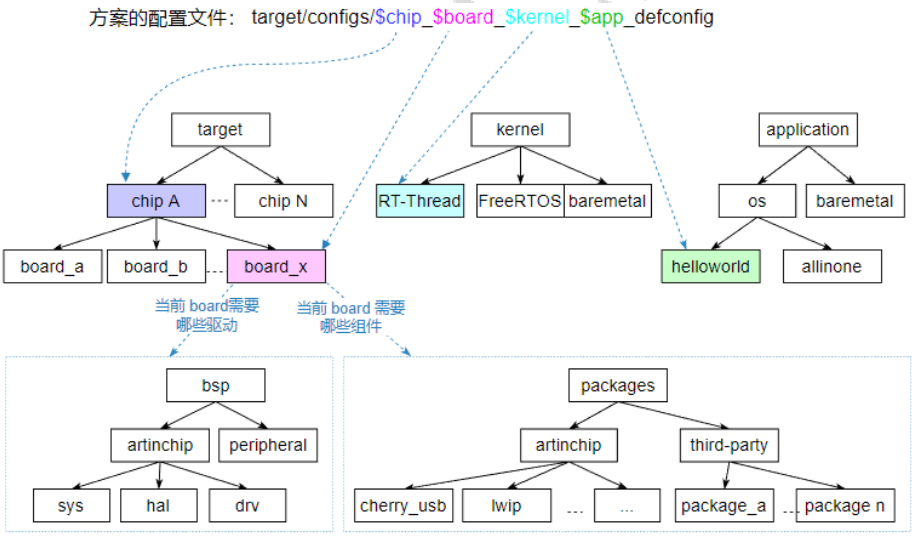
- 用高内聚提供复用：减少代码冗余，减少维护工作量
- 用低耦合应对变化：针对特定方案可灵活配置，满足多元化使用场景

Baremetal SDK 框架可抽象为以下四个层级的元素，各层级与配置文件的对应关系如下图所示：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



在具体的 Baremetal 设计中，从用户角度看，以上四级基本元素和 SDK 目录的对应关系如下图：



2.2.4. 编译框架

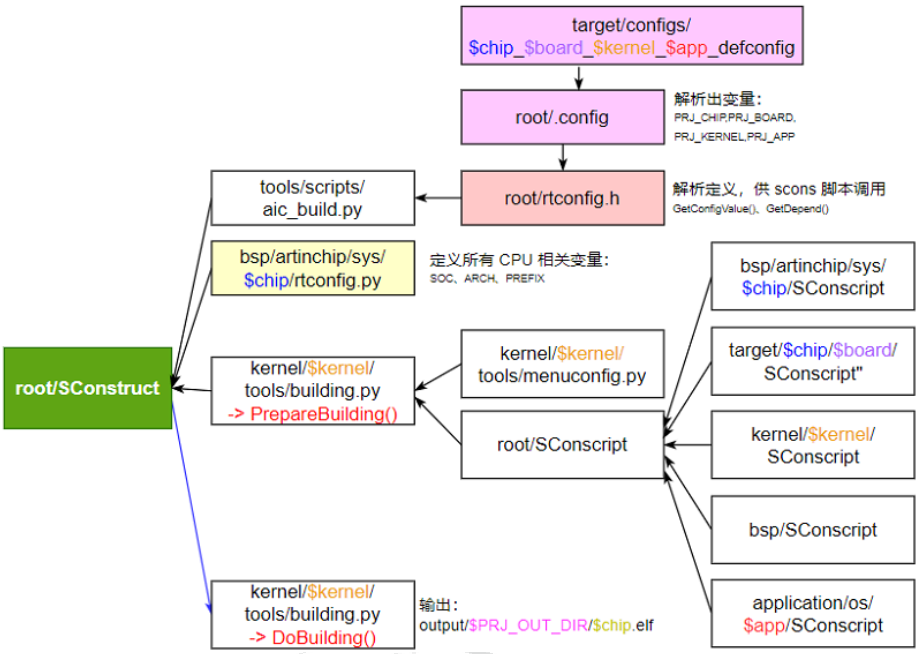
Baremetal SDK 采用了 SCons 作为编译框架的底层语言，提供了灵活的编译支持，覆盖以下三种主要场景：

> **注：**
> 关于 SCons 的详细使用说明，可参考 SConstruct。

- Linux 命令行：开发者可以在 Linux 环境中通过命令行执行 SCons 脚本来配置和编译项目。
- Windows 命令行：在 Windows 操作系统下，通过 CMD 或 Git-bash，以及 RT-Thread env 环境，使用 SCons 命令进行项目的构建和编译。
- Windows IDE：对于习惯在集成开发环境 (IDE) 中工作的开发者，可在 Windows IDE 中集成 SCons，以实现可视化的编译流程。

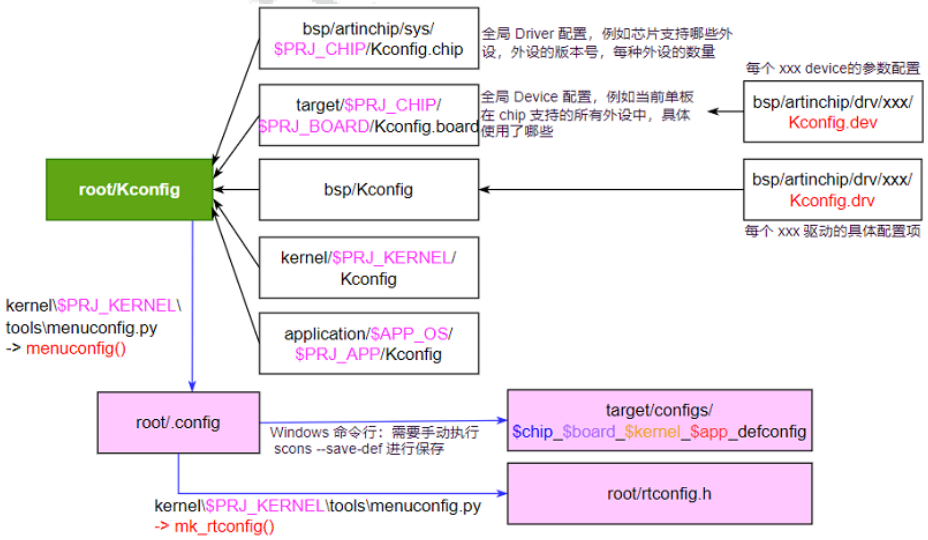
Baremetal 编译框架使用了以下树形结构的层次化引用：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



2.2.5. 配置框架

Baremetal SDK 采用 menuconfig 工具来进行配置，提升用户修改配置的易用性和简洁性。



menuconfig 配置框架使用树形结构进行层次化引用。配置选项清晰、逻辑分明，并且易于用户理解和操作。

1. 在 Baremetal 下，一个 `.config` 文件可同时保存 Driver 和 Device 配置信息。这种涉及简化了配置文件的管理，并使得维护更为集中和方便。
2. 在 Kconfig 的分区设计中，每个模块的 `Kconfig` 被细分成两个部分：
    - `Kconfig.dev`，存放 Device 相关的配置参数，比如 UART 模块的波特率、停止位参数。
    - `Kconfig.drv`，存放 Driver 的通用配置参数，比如 UART 模块的 DMA 开关。

关于 menuconfig 命令工具的详细描述，可查看 menuconfig 命令工具参考指南。

2.2.6. 驱动框架

为了简化开发流程并提高代码复用性，AIC Driver 驱动框架分为以下层次，以适应不同的硬件和应用场景：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- RT-Thread Driver Framework：实现由 RT-Thread 提供的驱动模型的功能。
- AIC Driver Layer： RT-Thread Driver Framework 的具体实现层，负责将 RT-Thread 的标准接口转换为 AIC HAL Layer 可以理解的接口。通过转换，确保了上层应用和底层硬件之间的独立性，使得开发者在不修改硬件相关代码的情况下，可以通过修改这一层的实现来适配不同的硬件。
- AIC HAL Layer：对底层硬件操作的封装，一般是寄存器级别的功能接口，为上层提供了统一的硬件访问接口。除了用于正常的设备驱动之外，这一层还被用于 baremetal 模式的 APP 调用，即在没有操作系统的情况下直接运行应用程序。

驱动复用和移植注意事项

在移植一个新的设备驱动时，开发工作主要涉及 AIC Driver Layer 和 AIC HAL Layer。

为了确保驱动可以在多种形态下复用并最大程度地提高可移植性与维护性，需要遵循以下关键原则和建议：

- 使用 AIC OSAL 接口

  在 AIC Driver Layer 和 AIC HAL Layer 中，应尽可能使用 AIC OSAL 接口，避免直接调用特定的 Kernel 接口。这有助于减少对特定操作系统的依赖，从而提高代码的泛用性和可移植性。

- 避免直接调用 RT-Thread 系统接口

  除了驱动注册不可避免的需要调用 RT-Thread 接口外，AIC Driver Layer 中应避免直接调用 RT-Thread 系统接口和 RT-Thread 的相关类型定义。这种操作可以保持驱动与操作系统之间的独立性，使得驱动更容易在其他操作系统上重用。

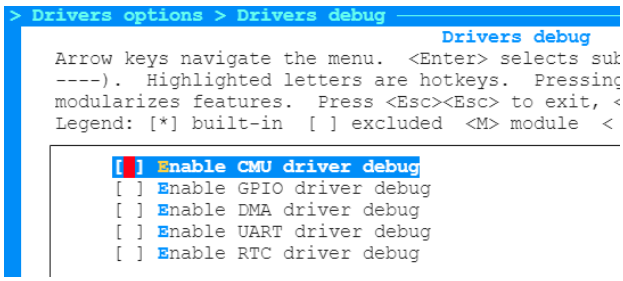
- 集中管理中断和同步机制

  对于中断注册和互斥锁、信号量的操作，应尽可能放在 AIC Driver Layer 中处理，避免放在 AIC HAL Layer 中。这种处理方式有利于上层统一管理资源的同步和并发控制，同时也更加符合分层设计的原则，提高了代码的模块化和可读性。

2.2.7. 驱动调试

在 menuconfig 中，ArtInChip 的每个驱动都设置了 DEBUG 开关，用于打开相应模块的调试信息或者调试命令。

DEBUG 开关统一放置在同一个区域，便于用户查找和使用。如需在 menuconfig 中启用测试代码，可遵循以下配置步骤：

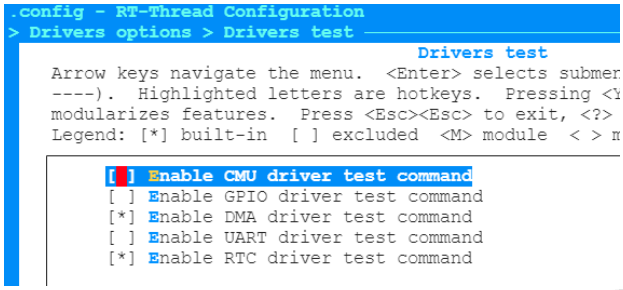


2.2.8. 驱动测试

ArtInChip 在 `bsp/examples/` 目录中实现了部分驱动的示例代码，可以作为模块的测试代码和应用开发设计参考样本。

ArtInChip 预设的示例代码，通常封装为特定的 Shell 命令，可在系统启动后通过输入 Shell 命令的方式来触发预定的代码。如需在 menuconfig 中启用测试代码，可遵循以下配置步骤：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



2.3. SConstruct

SConstruct，一般简称为 SCons，全称为 Software Construction Tool，是一个用 Python 编写的开源构建工具，旨在改进和自动化软件构建过程。SConstruct 提供了一种更简单、更可靠且高效的替代传统构建系统如 GNU Make 的方案。

Baremetal 使用 SConstruct 来实现自动化构建。

2.3.1. 基本用法和特点

SCons 使用 SConstruct 和 SConscript 文件来组织源码结构。

在大型项目中，通常会有一个 SConstruct 和多个 SConscript 文件，分别位于含有源代码的子目录中。原则上每个存放有源代码的子目录下都会放置一个 SConscript，但譬如 BSP 的驱动开发等会集合所有的驱动源码到一个 SConscript 中。

SCons 具有以下特点：

- SCons 使用 Python 脚本做为配置文件，具有强大的表达式力和灵活性，允许执行复杂的逻辑和操作。
- Python 的跨平台特性使得 SCons 自然支持多平台，可以无缝地在 Windows、Linux 及其他操作系统上运行，无需修改构建脚本。
- 自动检测源文件和头文件之间的依赖关系，确保只有在必要时才重新构建项目。
- 支持多种编程语言，包括 C/ C++/ D/ Java/ Fortran/ Yacc/ Lex/ Qt/ SWIG 以及 Tex/Latex，扩展性好，支持用户自扩展编程语言。
- 支持 make -j 风格的并行构建，允许同时运行多个工作，以利用多核处理器的优势，提高构建速度。SCons 管理并行任务时会考虑文件层次结构和依赖性，避免构建错误。
- 使用 Autoconf 风格自动查找系统上的头文件，函数库，函数和类型定义，简化了跨系统构建配置的复杂性。
- 基于 MD5 识别构建文件是否发生变化，比传统的时间戳更精准和安全。

2.3.2. 环境安装

- **Windows**

  Baremetal 具有小体积，良好的兼容性，以及快速的特点，尤其是在命令行的开发和构建方面，推荐在 Windows 系统上进行操作。

  Windows 下的对应工具存放在 `luban-lite/tools/env/tools` 目录中，不需要安装。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

名称	修改日期	类型	大小
bin	2023/6/22 15:17	文件夹	
BuildTools	2023/6/22 15:17	文件夹	
ConEmu	2023/6/22 15:18	文件夹	
fatdisk	2023/6/22 15:18	文件夹	
MinGit-2.25.1-32-bit	2023/6/22 15:18	文件夹	
Python27	2023/6/22 15:17	文件夹	
Python27_32	2023/6/22 15:17	文件夹	
Python39	2023/6/22 15:18	文件夹	
qemu	2023/6/22 15:18	文件夹	
scripts	2023/6/22 15:18	文件夹	
get-pip.py	2023/6/17 9:30	Python 源文件	1,623 KB
维护注意.md	2023/6/17 9:30	Markdown File	1 KB

- **Linux**

  在 Linux 系统上设置 Baremetal 固件构建环境，需要安装关键的依赖包，包括：

  > **注：**
  > 推荐使用 `apt-get` 命令直接安装目标软件和依赖。

  1. python2 + SCons

     安装 SCons

     ```
     sudo apt-get install scons
     ```

  2. python3 + pycryptodomex

     安装 pycryptodomex

     pycryptodomex 是一个 python 编写的加密包，Baremetal 中有源码包可以进行编译安装

     ```
     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
     ```

2.3.3. SConstruct

Baremetal 的 SConstruct 脚本放置在 SDK 根目录下，负责配置编译逻辑、设置全局环境变量，并允许用户添加自定义的私有环境变量 (ENV)，以便实现高度定制化的构建配置：

- 配置编译逻辑：通过定义一系列的规则和依赖关系，SConstruct 脚本可以自动化编译流程。

  相关规则可以指定如何从源代码生成目标文件，包括编译器选择、编译选项及链接过程。用户可以在 SConstruct 中定义不同的编译配置，例如调试版本和发布版本，每种配置可以有不同的编译器选项和优化设置，从而满足不同环境下的构建需求。

- 设置全局环境变量：SConstruct 提供了设置全局环境变量的功能，如编译器路径、包含目录等，确保构建过程能在正确的环境下执行。

  通过调整环境变量，Baremetal 可以适应不同的操作系统和工具链，提高构建系统的灵活性和兼容性。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
import os
import sys

# Luban-Lite root directory
AIC_ROOT = os.path.normpath(os.getcwd())

# luban-lite custom scripts
aic_script_path = os.path.join(AIC_ROOT, 'tools/scripts/')
sys.path.append(aic_script_path)
from aic_build import *
chk_prj_config(AIC_ROOT)
PRJ_CHIP,PRJ_BOARD,PRJ_KERNEL,PRJ_APP,PRJ_DEFCONFIG_NAME,PRJ_CUSTOM_LDS,MKIMAGE_POST_ACTION = get_prj_config(AIC_ROOT)
PRJ_NAME = PRJ_DEFCONFIG_NAME.replace('_defconfig','')
PRJ_OUT_DIR = 'output/' + PRJ_NAME + '/images/'
AIC_SCRIPT_DIR = aic_script_path
AIC_COMMON_DIR = os.path.join(AIC_ROOT, 'bsp/artinchip/sys/' + PRJ_CHIP)
AIC_PACK_DIR = os.path.join(AIC_ROOT, 'target/' + PRJ_CHIP + '/' + PRJ_BOARD + '/pack/')

# Var tranfer to SConscript
Export('AIC_ROOT')
Export('AIC_SCRIPT_DIR')
Export('AIC_COMMON_DIR')
Export('AIC_PACK_DIR')
Export('PRJ_CHIP')
Export('PRJ_BOARD')
Export('PRJ_KERNEL')
Export('PRJ_APP')
Export('PRJ_NAME')
Export('PRJ_DEFCONFIG_NAME')
Export('PRJ_OUT_DIR')
# Var tranfer to Kconfig 'option env=xxx'
os.environ["AIC_ROOT"]            = AIC_ROOT
os.environ["AIC_SCRIPT_DIR"]     = AIC_SCRIPT_DIR
os.environ["AIC_COMMON_DIR"]      = AIC_COMMON_DIR
os.environ["AIC_PACK_DIR"]        = AIC_PACK_DIR
os.environ["PRJ_CHIP"]            = PRJ_CHIP
os.environ["PRJ_BOARD"]           = PRJ_BOARD
os.environ["PRJ_KERNEL"]          = PRJ_KERNEL
os.environ["PRJ_APP"]             = PRJ_APP
os.environ["PRJ_NAME"]            = PRJ_NAME
os.environ["PRJ_DEFCONFIG_NAME"] = PRJ_DEFCONFIG_NAME
os.environ["PRJ_OUT_DIR"]         = PRJ_OUT_DIR

# rtconfig
chip_path = os.path.join(AIC_ROOT, 'bsp/artinchip/sys/' + PRJ_CHIP)
sys.path.append(chip_path)
import rtconfig

# RTT_ROOT
if os.getenv('RTT_ROOT'):
    RTT_ROOT = os.getenv('RTT_ROOT')
else:
    RTT_ROOT = os.path.join(AIC_ROOT, 'kernel/rt-thread/')
os.environ["RTT_ROOT"]            = RTT_ROOT
sys.path.append(os.path.join(RTT_ROOT, 'tools'))
from building import *

# ENV_ROOT
if os.getenv('ENV_ROOT') is None:
    ENV_ROOT = RTT_ROOT + '/../../tools/env'
    os.environ["ENV_ROOT"] =  ENV_ROOT

# TARGET
TARGET = PRJ_OUT_DIR + rtconfig.SOC + '.' + rtconfig.TARGET_EXT

rtconfig.LFLAGS += ' -T ' + ld

# add post action
rtconfig.POST_ACTION += MKIMAGE_POST_ACTION

# create env
env  = Environment(tools = ['mingw'],
AS   = rtconfig.AS,   ASFLAGS   = rtconfig.AFLAGS,
CC   = rtconfig.CC,   CFLAGS    = rtconfig.CFLAGS,
CXX  = rtconfig.CXX,  CXXFLAGS  = rtconfig.CXXFLAGS,
AR   = rtconfig.AR,   ARFLAGS   = '-rc',
LINK = rtconfig.LINK, LINKFLAGS = rtconfig.LFLAGS)
env.PrependENVPath('PATH', rtconfig.EXEC_PATH)

# add --start-group and --end-group for GNU GCC
env['LINKCOM'] = '$LINK -o $TARGET $LINKFLAGS $__RPATH $SOURCES $_LIBDIRFLAGS -Wl,--start-group $_LIBFLAGS -Wl,--end-group'
env['ASCOM'] = env['ASPPCOM']

# signature database
env.SConsignFile(PRJ_OUT_DIR + ".sconsign.dblite")

Export('RTT_ROOT')
Export('rtconfig')

# Var tranfer to building.py
env['AIC_ROOT']                  = AIC_ROOT
env['AIC_SCRIPT_DIR']           = AIC_SCRIPT_DIR
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
env['AIC_COMMON_DIR']            = AIC_COMMON_DIR
env['AIC_PACK_DIR']              = AIC_PACK_DIR
env['PRJ_CHIP']                  = PRJ_CHIP
env['PRJ_BOARD']                 = PRJ_BOARD
env['PRJ_KERNEL']                = PRJ_KERNEL
env['PRJ_NAME']                  = PRJ_NAME
env['PRJ_APP']                   = PRJ_APP
env['PRJ_DEFCONFIG_NAME']        = PRJ_DEFCONFIG_NAME
env['PRJ_OUT_DIR']               = PRJ_OUT_DIR

# prepare building environment
objs = PrepareBuilding(env, RTT_ROOT, has_libcpu=False)

# make a building
DoBuilding(TARGET, objs)
```

2.3.4. SConscript

SConscript 是 SCons 构建系统的配置文件，用于定义构建任务和依赖关系，以支持复杂的构建过程。SConscript 脚本位于源代码树的子目录中，每个 SConscript 文件可以描述该目录下的所有构建任务，从而简化构建流程的管理和维护。

一些常用的 SConscript 方法有：

- 使用 `Program` 生成可执行文件

  `Program` 用于生成可执行文件的示例如下：

```
Program('hello.c')           编译 hello.c 可执行文件，根据系统自动生成(hello.exe on Windows; hello on POSIX)
Program('hello','hello.c')  指定 Output 文件名(hello.exe on Windows; hello on POSIX)
Program(['hello.c', 'file1.c', 'file2.c']) 编译多个文件，Output 文件名以第一个文件命名
Program(source = "hello.c",target = "hello")
Program(target = "hello" , source = "hello.c")
Program('hello', Split('hello.c file1.c file2.c')) 编译多个文件

Program(Glob("*.c"))
src = ["hello.c","foo.c"];
Program(src)
```

- 使用 `Object` 生成目标文件

  `Object` 用于生成目标文件的示例如下：

```
Object('hello.c') 编译 hello.c 目标文件，根据系统自动生成(hello.obj on Windows; hello.o on POSIX)
```

- 使用 `Library` 生成静态/动态库文件

  `Library` 用于生成静态/动态库文件

```
Library('foo', ['f1.c', 'f2.c', 'f3.c']) 编译 library
SharedLibrary('foo', ['f1.c', 'f2.c', 'f3.c']) 编译 shared library
StaticLibrary('bar', ['f4.c', 'f5.c', 'f6.c']) 编译 static library

库的使用：

Program('prog.c', LIBS=['foo', 'bar'], LIBPATH='.') 连接库，不需加后缀或是前缀
```

- 使用 `Depends` 明确依赖关系

  `Depends` 用于明确依赖关系

```
Depends(hello,'other_file')//hello 依赖于 other_file
```

驱动程序的 SConscript 示例

以下是一个典型的驱动程序的 SConscript 示例，用于构建一个 UART 驱动程序。SConscript 文件示例定义了一个名为 `aic_osal` 的构建组，包含了 UART 驱动程序的所有源文件，并根据需要设置了头文件路径和编译选项。详情如下：

```
Import('AIC_ROOT')
Import('PRJ_KERNEL')
from building import *

cwd = GetCurrentDir()
src = Glob('*.c')
CPPPATH = []
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
if GetDepend('DRIVER_DRV_EN'):
    CPPPATH.append(cwd + '/include/drv')
if GetDepend('DRIVER_HAL_EN'):
    CPPPATH.append(cwd + '/include/hal')
    CPPPATH.append(cwd + '/include/uapi')

# UART driver
if GetDepend('AIC_UART_DRV'):
    if GetDepend('DRIVER_DRV_EN'):
        src += Glob('drv/uart/*.c')
    if GetDepend('DRIVER_HAL_EN'):
        src += Glob('hal/uart/*.c')

LOCAL_CCFLAGS += ' -O0'

//DefineGroup(name, src, depend, **parameters)
group = DefineGroup('aic_osal', src, depend=[''], CPPPATH=CPPPATH, LOCAL_CCFLAGS=LOCAL_CCFLAGS)

Return('group')
```

上述 SConscript 示例代码主要完成以下任务：

1. 导入必要的变量和模块。
2. 获取当前目录并设置源文件列表 **src**。
3. 根据条件判断是否包含特定的头文件路径到 **CPPPATH** 中。
4. 如果启用了 UART 驱动程序，根据条件将相应的源文件添加到 **src** 列表中。
5. 添加编译选项 **-O0** 到 **LOCAL_CCFLAGS**。
6. 使用 `DefineGroup` 函数创建一个名为 `aic_osal` 的构建组，其中包含源文件、依赖项、头文件路径和编译选项。
7. 返回创建的构建组。

应用程序的 SConscript 示例

以下是一个典型的应用程序的 SConscript 示例，用于构建一个多媒体播放器：

```
Import('AIC_ROOT')
Import('PRJ_KERNEL')
from building import *

cwd = GetCurrentDir()
path = [cwd + '/include']
path += [cwd + '/base/include']
path += [cwd + '/ge/include']

path += [cwd + '/ve/include']
path += [cwd + '../../../bsp/artinchip/include/uapi']
path += [cwd + '/mpp_test']

if GetDepend(['AIC_MPP_PLAYER_INTERFACE']):
    #audio decoder
    path += [cwd + '/middle_media/audio_decoder/include']
    path += [cwd + '/middle_media/audio_decoder/decoder']

    #base
    path += [cwd + '/middle_media/base/include']
    path += [cwd + '/middle_media/base/parser/mov']
    path += [cwd + '/middle_media/base/parser/rawdata']
    path += [cwd + '/middle_media/base/stream/file']


src = []
CPPDEFINES = []

# mpp
if GetDepend(['LPKG_MPP']):
    src += Glob('./base/memory/*.c')
    src += Glob('./ge/*.c')
    src += Glob('./fb/*.c')
    src += Glob('ve/decoder/*.c')
    src += Glob('ve/common/*.c')
    src += Glob('ve/decoder/jpeg/*.c')
    src += Glob('ve/decoder/png/*.c')
    src += Glob('ve/decoder/h264/*.c')
    src += Glob('./mpp_test/*.c')


if GetDepend(['AIC_MPP_PLAYER_INTERFACE']):
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
    #audio decoder
    src += Glob('middle_media/audio_decoder/decoder/*.c')
    src += Glob('middle_media/audio_decoder/decoder/mp3/mp3_decoder.c')

//DefineGroup(name, src, depend, **parameters)
group = DefineGroup('mpp', src, depend = [''], CPPPATH = path, CPPDEFINES = CPPDEFINES)

Return('group')
```

上述 SConscript 示例代码主要完成以下任务：

1. 导入必要的变量和模块。
2. 获取当前目录并设置头文件路径列表 **path**。
3. 根据条件判断是否包含特定的头文件路径到 **path** 中。
4. 初始化源文件列表 **src** 和编译选项列表 **CPPDEFINES**。
5. 如果启用了多媒体播放器接口，根据条件将相应的源文件添加到 **src** 列表中。
6. 使用 DefineGroup 函数创建一个名为 **mpp** 的构建组，其中包含源文件、依赖项、头文件路径和编译选项。
7. 返回创建的构建组。

2.4. 编译选项介绍

Baremetal SDK 采用 SCons 命令编译：

- 编译操作
  - 查看 SDK 支持板卡：scons --list-def
  - 选择板卡配置：scons --apply-def=项目索引或名称
  - 修改和配置参数：scons --menuconfig
  - 编译：scons
- 查看生成目录：查看编译完成后的目录结构

2.5. 配置分区

在 Baremetal 系统中，分区的设置和管理是系统正常运行和日常维护的重要方面。本章节介绍 Baremetal 系统中分区的特点、规则和分区修改的方法。

按照存储介质的逻辑划分，Baremetal 默认分区包括以下几种：

- 启动分区：存放 BootLoader，用于系统启动。
- 数据分区：用于存储用户数据。
- 系统 A/B 分区：分为 A/B 系统，实现 OTA，即 Flash 上保存前后两个版本的系统程序。

2.5.1. 启动分区

在 Baremetal 系统中，启动分区（"spl" 分区）是存储介质上的第一个分区，用于存放 BootLoader，并且根据不同的存储介质类型和配置，其位置、大小和内容有所不同。

"spl" 分区为默认名称，用户可以根据需要自定义和修改启动分区名称。

Baremetal 系统中的默认分区设置如下：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- SPINOR

  SPINOR Flash 的 0 ~ X 区间被设置为启动分区，主要用于存放 BootLoader，大小设置为 256K。

  尽管 BROM 加载的程序可以大于 256K，但是由于 BROM 运行时的速度较慢，不建议加载过大的程序。

- SPINAND

  SPINAND Flash 的前四个物理块被用作启动块，启动分区内使用四个启动块保存两份启动镜像：

  - 对于块大小为 128K 的 SPI NAND，启动分区为 512K。
  - 对于 块大小为 256K 的 SPI NAND，启动分区为 1M。

- eMMC

  eMMC 的启动分区位于 UDA 区域的 17K ~ X 区间，BROM 从该区域读取程序。

  建议启动镜像小于 512K。

2.5.2. 数据分区

在 Baremetal 系统中，通过对 SPI NOR/ SPI NAND Flash 进行分区可以实现数据的存储和管理，为数据的读写提供了不同的功能和性能特点。

在 Baremetal 系统中，SPI NOR/ SPI NAND Flash 上设置了以下默认分区来保存数据。两个分区在设计和实现上有一些差异，用户需要了解并根据实际情况规划数据存储：

- "rodata"

  目的：保存 APP 应用需要使用的资源文件，这些文件通常是应用的一部分，都是只读的数据，只是为了更好的管理和访问，使用 FAT 文件系统的方式保存。

  应用在访问这些资源文件的时候，通常要求能够快速读取数据，因此 "rodata" 分区在实现的时候，删减了多余的软件层次，通过直接访问 Flash 的方式读取数据，以达到更优的速度。这些优化，也使得 "rodata" 仅支持读取，不支持写入。

  注：

  使用 SPINOR 时，可以通过配置的方式强制使能 "rodata" 可读写，但是需要注意其中的风险：

  1. Flash 写入前需要先按块擦除，但是擦除和重新写入需要的时间比较长
  2. FATFS 没有日志，不支持异地写入，因此需要原地回写，如果擦除和回写时出现掉电，会导致文件系统损坏

- "data"
  - 目的：提供一个可读写的文件系统分区，用于保存设备在使用过程中生成的需要写入 Flash 的数据。
  - 文件系统：
    - 在 SPI NOR Flash 上，使用 LittleFS。
    - 在 SPI NAND Flash 上，使用 FATFS。
    - 在 eMMC 上，每个分区都是可读写的，默认使用 FATFS。
  - 读写支持：可读写，适合存储设备运行过程中产生的临时数据或需要修改的数据。
  - 适用场景：需要频繁读写的场景，如用户设置、日志记录等。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.5.3. 系统 A/B 分区

在 Baremetal 系统中，A/B 系统分区是实现 Over-the-Air (OTA) 软件更新机制的一个重要部分。这种分区方式需要在 Flash 上保存前后两个版本的系统程序，并将每个版本保存在指定的分区中。

A/B 系统的分区命名比较直接，便于轻松识别两套软件版本，详细命名规则如下所示：

- A 系统分区：直接使用其所代表的分区的名称，例如存储只读数据的 "rodata" 分区，则 A 系统分区即命名为 "rodata"。
- B 系统分区：在对应的 A 系统分区名后加上 `_r` 后缀，例如 `rodata_r`。

使用专门的 A/B 分区机制，可以最大化资源使用、实现备份，并且提前配置所需软件更新的存储空间，而不影响系统的运行性能。

2.5.4. 分区与存储设备关系

Baremetal 上使用 RT-Thread 时，每个分区都对应一个或者多个系统设备。不同存储介质的对应关系如下表格所示。系统中出现但此处未列出的其他分区，同样遵循下列规则。

表 2-1 SPI NOR 上的分区、设备和文件系统

分区名	MTD 设备名	块设备名	文件系统	备注
"spl"	"spl"	–	–	–
"os"	"os"	–	–	–
"rodata"	"rodata"	"blk_rodata"	FATFS	FATFS 通过 "blk_rodata" 设备访问 Flash
"data"	"data"	–	LittleFS	LittleFS 直接通过 MTD 设备访问 Flash

- 分区名：
  - "spl"：分区及其设备名，"spl" 分区主要用于存储二次引导加载程序，是系统启动的第一个阶段所加载的程序。
  - "os"："os" 分区，存放操作系统的主要部分，是系统运行的核心区域。
  - "rodata"："rodata" 分区，主要用于存储只读数据，包括系统在运行时需要读取但不修改的配置信息等。
  - "data"："data" 分区，用来存储可变数据，如用户生成的数据或系统运行时产生的临时数据。
- MTD 设备名：对 Flash 的底层直接访问，便于进行读写硬件等操作。
- 块设备名：通过将物理存储抽象为块设备，简化了对文件系统的操作。仅 "rodata" 分区关联了一个块设备名 "blk_rodata"。
- 文件系统："rodata" 分区使用 FATFS 文件系统，使得数据以文件的形式进行存储和管理。"data" 分区使用 LittleFS 文件系统。
- 访问规则：
  - 直接访问 Flash，使用 MTD 设备
  - FATFS 访问 Flash，需要使用块设备
  - LittleFS 访问 Flash，使用 MTD 设备

表 2-2 SPI NAND 上的分区、设备和文件系统

分区名字	MTD 设备名	块设备名	文件系统	备注
"spl"	"spl"	–	–	–
"os"	"os"	–	–	–

表 2-2 SPI NAND 上的分区、设备和文件系统 (续)

分区名字	MTD 设备名	块设备名	文件系统	备注
"rodata"	"rodata"	"blk_rodata"	FATFS	通过 "blk_rodata" 设备访问 Flash，FATFS 只读

Copyright © 2023-2025 ArtInChip Technology Co., Ltd.
All rights reserved

表 2-2 SPI NAND 上的分区、设备和文件系统 (续)

分区名字	MTD 设备名	块设备名	文件系统	备注
"rodata"	"rodata"	"blk_rodata"	FATFS	通过 "blk_rodata" 设备访问 Flash，FATFS 只读
"data"	"data"	"blk_data"	FATFS	"blk_data"设备是通过 NFTL 层创建， FATFS 访问 "blk_data" 时，通过 NFTL 访问 Flash。

- 分区名：
  - "spl"：分区及其设备名，"spl" 分区主要用于存储 二次引导加载程序，是系统启动的第一个阶段所加载的程序。
  - "os"："os" 分区，存放操作系统的主要部分，是系统运行的核心区域。
  - "rodata"："rodata" 分区，主要用于存储只读数据，包括系统在运行时需要读取但不修改的配置信息等。
  - "data"："data" 分区，用来存储可变数据，如用户生成的数据或系统运行时产生的临时数据。
- MTD 设备名：对 Flash 的底层直接访问，便于进行读写硬件等操作。
- 块设备名：通过将物理存储抽象为块设备，简化了对文件系统的操作。"blk_rodata" 和 "blk_data" 分别支持对 "rodata" 和 "data" 分区的文件系统访问。
- 文件系统："rodata" 分区使用 FATFS 文件系统，使得数据以文件的形式进行存储和管理。"data" 分区使用 FATFS 文件系统。
- NFTL 层：网络闪存转换层，主要用来将 Flash 的物理特性映射为一般块设备，便于 FATFS 等文件系统无需关心底层 NAND Flash 的具体特性。

> **注：**
>
> 建议只设置一个使用 NFTL 的分区，并且分区不要太小。因为 NFTL 需要预留至少 6 MB 空间做坏块管理。

- 访问规则：
  - 直接访问 Flash，使用 MTD 设备
  - FATFS 访问 Flash，需要使用块设备，通过 NFTL 层访问 Flash

表 2-3 eMMC 上的分区、设备和文件系统

分区名字	MTD 设备名	块设备名	文件系统	备注
"spl"	–	`mmc0p0`	–	–
"os"	–	`mmc0p1`	–	–
"rodata"	–	`mmc0p2`	FATFS	此处的分区排序仅为示例
"data"	–	`mmc0p3`	FATFS	此处的分区排序仅为示例

- **分区名**：
  - "spl"：分区及其设备名，"spl" 分区主要用于存储 二次引导加载程序，是系统启动的第一个阶段所加载的程序。
  - "os"："os" 分区，存放操作系统的主要部分，是系统运行的核心区域。
  - "rodata" ："rodata" 分区，主要用于存储只读数据，包括系统在运行时需要读取但不修改的配置信息等。
  - "data"："data" 分区，用来存储可变数据，如用户生成的数据或系统运行时产生的临时数据。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- MTD 设备名：对 Flash 的底层直接访问，便于进行读写硬件等操作。
- 块设备名：块设备按照分区顺序从 0 开始编号，例如从 mmc0p0 到 mmc0p3。每个块设备代表 eMMC 中的一个分区，简化了物理存储到逻辑存储的映射，便于后续的文件系统管理。
- 文件系统："rodata" 和 "data" 分区均使用 FATFS 文件系统。
- 访问规则：块设备的命名，按照分区的排序从 0 开始编号

2.5.5. 修改分区

`image_cfg.json` 文件是设置分区信息以及如何生成烧录镜像文件的配置文件，由于分区和烧录信息是相互关联的，因此在一个文件中进行配置。 `image_cfg.json` 文件的保存路径为 `target/<chip>/<board>/pack/image_cfg.json`。

分区数值的书写规则如下：

- 不带单位，表示字节
- 可以使用单位 k 或者 K
- 可以使用单位 m 或者 M
- 可以使用单位 g 或者 G

输入分区 "size" 值时，需要注意存储介质的对齐要求：

- SPINOR: 与最小擦除块大小对齐，一般为 4K 或者 64K，具体请参考 SPINOR 的数据手册。
- SPI NAND: 与最小擦除块大小对齐，一般为 128K 或者 256K，具体请参考 SPINAND 的数据手册。
- eMMC: 需要 512 字节对齐。

分区设置步骤和流程如下所示：

1. 在 `image_cfg.json` 文件中查找对应媒介的分区配置信息。

   `image_cfg.json` 文件是设置分区信息以及配置烧录镜像文件生成方式的配置文件，保存路径为 `target/<chip>/<board>/pack/image_cfg.json`。

   - *<chip>* 为对应芯片型号。
   - *<board>* 为芯片开发板型号。

   以下以 SPI NAND 的分区配置为例：

   > **注：**
   > 对于 SPI NAND Flash 的分区配置，需要注意 NFTL 分区的配置方法。

```
"spi-nand": {
    "size": "128m", // Size of SPI NAND
    "partitions": {
        "spl":      { "size": "1m" },
        "env":      { "size": "256k" },
        "env_r":    { "size": "256k" },
        "os":       { "size": "4m" },
        "os_r":     { "size": "4m" },
        "rodata":   { "size": "12m" },
        "rodata_r": { "size": "12m" },
        "data": {
            "size": "40m",
            "nftl": { // Volume in NFTL device
                "datavol": { "size": "-" },
            },
        },
    },
},
```

2. 根据所需分区尺寸，修改对应的分区 "size" 值，否则可跳过此步。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

例如，将 "data" 分区中的 "size" 从 40M 修改为 50M 后，示例如下：

```
"spi-nand": {
    "size": "128m", // Size of SPI NAND
    "partitions": {
        "spl":      { "size": "1m" },
        "env":      { "size": "256k" },
        "env_r":    { "size": "256k" },
        "os":       { "size": "4m" },
        "os_r":     { "size": "4m" },
        "rodata":   { "size": "12m" },
        "rodata_r": { "size": "12m" },
        "data": {
            "size": "50m",
            "nftl": { // Volume in NFTL device
                "datavol": { "size": "-" },
            },
        },
    },
},
```

3. 如需新增一个分区，在所需位置添加一行配置即可，否则可跳过此步。

   例如，新增一个 "user" 分区，示例如下：

```
"spi-nand": {
    "size": "128m", // Size of SPI NAND
    "partitions": {
        "spl":      { "size": "1m" },
        "env":      { "size": "256k" },
        "env_r":    { "size": "256k" },
        "os":       { "size": "4m" },
        "os_r":     { "size": "4m" },
        "rodata":   { "size": "12m" },
        "rodata_r": { "size": "12m" },
        "user":     { "size": "20m" },      //新添加的分区
        "data": {
            "size": "50m",
            "nftl": { // Volume in NFTL device
                "datavol": { "size": "20m" },
            },
        },
    },
},
```

4. 如果新增的分区需要烧录预制内容，在 "target" 段中新增配置。

   注：

   对于分区表中的分区，如果没有在 "target" 段中设置需要烧录的数据，则在烧录过程中，该分区不会被擦除和修改。

```
"target": { // Image components which will be burn to device's partitions
    "spl": {
        "file": "bootloader.aic",
        "attr": ["required"],
        "part": ["spl"]
    },

    ...

    "mydata": {                     //此处组件名字可自定义
        "file": "userdata.bin", //此处为要烧录到分区的数据
        "attr": ["mtd", "optional"],
        "part": ["user"]            //此处设置要烧录的分区名字
    },
},
```

5. 如需删除一个分区，删除对应的分区配置行即可。

   例如，删除新增的 "user" 分区，示例如下：

```
"spi-nand": {
    "size": "128m", // Size of SPI NAND
    "partitions": {
        "spl":      { "size": "1m" },
        "env":      { "size": "256k" },
        "env_r":    { "size": "256k" },
        "os":       { "size": "4m" },
        "os_r":     { "size": "4m" },
        "rodata":   { "size": "12m" },
        "rodata_r": { "size": "12m" },
        "data": {
            "size": "50m",
            "nftl": { // Volume in NFTL device
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
                "datavol": { "size": "20m" },
            },
        },
    },
},
```

注：

检查 `target` 段中是否有使用该分区，如果有，则删除相关的配置。

2.5.6. 分区配置示例

`image_cfg.json` 文件是设置分区信息以及配置烧录镜像文件生成方式的配置文件。由于分区和烧录信息是相互关联的，因此在一个文件中进行配置。 `image_cfg.json` 文件的保存路径为 `target/<chip>/<board>/pack/image_cfg.json`。

- *<chip>* 为对应芯片型号。
- *<board>* 为芯片开发板型号。

SPINOR 的分区配置示例：

```
"spi-nor": {
    "size": "16m", // Size of SPI NOR
    "partitions": {
        "spl":          { "size": "256k" },
        "env":          { "size": "128k" },
        "env_r":        { "size": "128k" },
        "os":           { "size": "1m" },
        "os_r":         { "size": "1m" },
        "rodata":       { "size": "3m" },
        "rodata_r":     { "size": "3m" },
        "data":         { "size": "7m" }
    },
},
```

SPI NAND Flash 的分区配置示例：

注：

对于 SPI NAND Flash 的分区配置，需要注意 NFTL 分区的配置方法。

```
"spi-nand": {
    "size": "128m", // Size of SPI NAND
    "partitions": {
        "spl":          { "size": "1m" },
        "env":          { "size": "256k" },
        "env_r":        { "size": "256k" },
        "os":           { "size": "4m" },
        "os_r":         { "size": "4m" },
        "rodata":       { "size": "12m" },
        "rodata_r":     { "size": "12m" },
        "data": {
            "size": "40m",
            "nftl": { // Volume in NFTL device
                "datavol": { "size": "-" },
            },
        },
    },
},
```

上述示例中， "data" 分区的配置了如下内容：

- "size"：40 MB
- "nftl" 设备：
  - "datavol"： 表示 NFTL 设备的一个卷。
  - "size"： “-” 表示卷的大小是 NFTL 设备的所有剩余存储空间。

    NFTL 设备需要使用大约 6 MB 的空间做坏块管理，因此 “datavol” 的可用大小大约为 34 MB。

eMMC 的分区配置示例：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

注：

第一个分区的开始位置。eMMC 使用 GPT 分区表格式，需要在 eMMC 的开始和末尾都预留 17K 的空间存放 GPT 信息，因此 “spl” 的开始位置为 “0x4400”。

```
"mmc": {
    "size": "8G", // Size of SD/eMMC
    "partitions": {
        "spl":          { "offset": "0x4400", "size": "256k" },
        "os":           { "size": "1m" },
        "rodata":       { "size": "3m" },
        "data":         { "size": "5m" }
    },
},
```

每一个分区的配置项中，都有 "offset" 和 "size" 两个配置信息，其中 "offset" 用于配置该分区的开始位置。 如未设置 "offset"，表示该分区紧跟上一个分区之后。

2.6. 配置烧录镜像

Baremetal SDK 编译的最终输出结果是一个用于烧录到目标平台的镜像文件。

表 2–4 烧录工具

工具	说明
AiBurn	用于通过 PC 烧录镜像。
串口调试工具	用于通过命令行进入烧录模式、查看烧录、启动状态。

执行下列流程，烧录镜像文件：

1. 打开 AiBurn，进入**烧写镜像**页面：

2. 选择以下任意方式，使板卡进入烧录模式：

   - 板卡上电前，按住开发板上的 UBOOT 键不放，再上电。
   - 板卡上电前，短接 SPI Flash 的 4、5 引脚，再上电。
   - 板卡上电前，拉低烧录引脚（SDK 中，默认为 PA0），再上电。
   - 板卡上电后，按住开发板上的 UBOOT 键不放，再按 RESET 键。
   - 在 RT-Thread 启动之后，在串口命令行输入 aicupg。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3. 进入烧录模式后， AiBurn 会识别板卡，当右下角出现 ArtInChip 设备已连接，表示成功进入烧录模式，如图所示。

4. 选择编译好的镜像文件。加载镜像时， AiBurn 会识别出镜像的基本信息。

5. 点击开始，进行烧录。烧录时，AiBurn 会显示烧录的进度、速率、用时、结果。

2.6.1. 烧录镜像格式和工具

2.6.1.1. 镜像格式

ArtInChip 的烧录镜像文件由组件 (FirmWare Component, FWC) 以及对应的组件元信息组成。 数据分布如下图所示：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

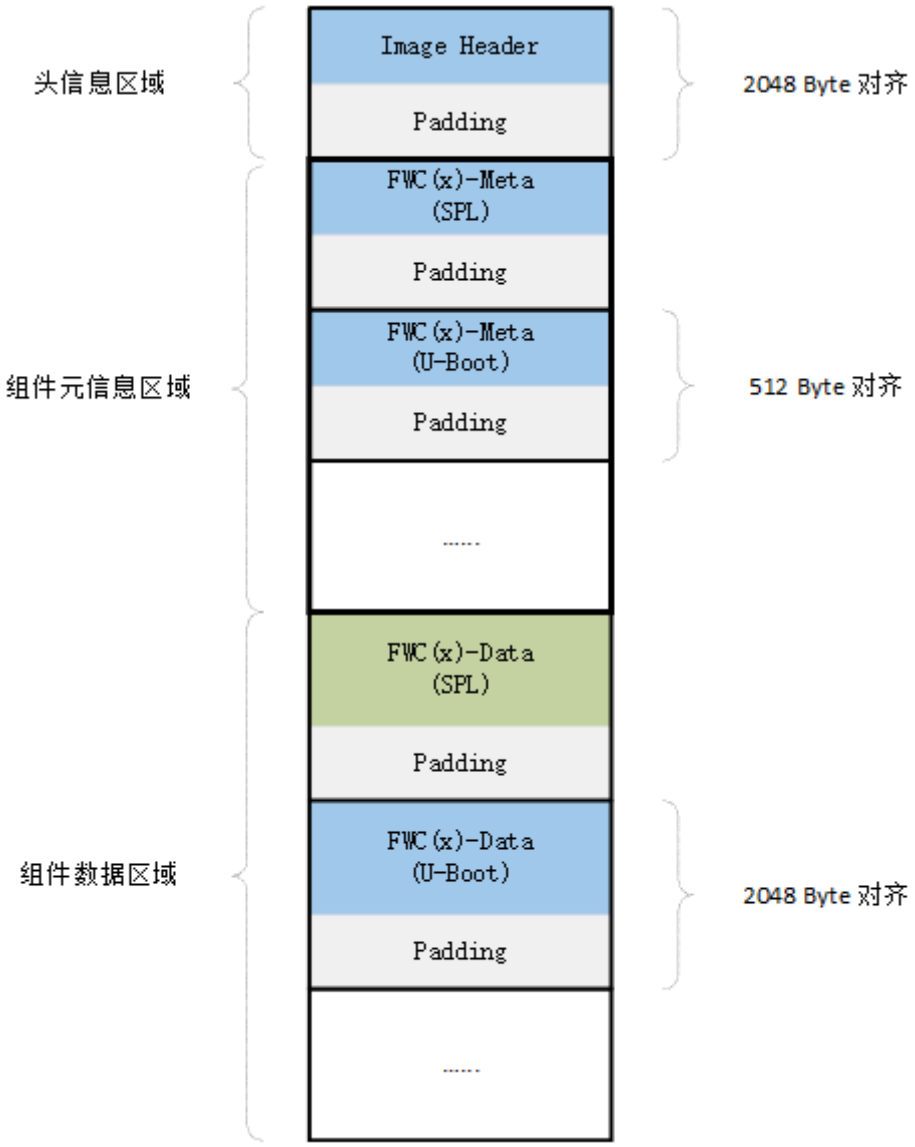


图 2-1 烧录镜像格式

其中一些需要打包的数据文件，都被当做组件 (FWC) 进行处理，包括 **SPL** ， **os** ， **rodata** ， **data** 等数据。

- Image Header 的具体格式如下：

```
struct artinchip_fw_hdr{
    char magic[8];              //固定为 "AIC.FW"
    char platform[64];          //该镜像文件适用的芯片平台
    char product[64];           //该镜像文件适用的产品型号
    char version[64];           //该镜像的版本
    char media_type[64];        //该镜像文件可烧录的存储介质
    u32  media_dev_id;           //该镜像文件可烧录的存储介质 ID
    u8   nand_array_org[64];     /* NAND Array Organization */
    u32  meta_offset;            /* Meta Area start offset */
    u32  meta_size;              /* Meta Area size */
    u32  file_offset;            /* File data Area start offset */
    u32  file_size;              /* File data Area size */
};
```

- FWC Meta 的格式如下：

```
struct artinchip_fwc_meta {
    char magic[8];      //固定为 "META"
    char name[64];      //对应组件的名字
    char partition[64]; //该组件要烧录的分区名字
    u32  offset;         //该组件数据在镜像文件中的偏移
    u32  size;           //该组件数据的大小
    u32  crc32;          //该组件数据的 CRC32 校验值
    u32  ram;            //当组件要下载到平台 RAM 时，要下载的地址
    char attr[64]       //该组件的属性，字符串表示
};
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.6.1.2. 制作工具

2.6.2. 烧录镜像配置文件

使用 `mk_image.py` 制作烧录镜像时，需要提供 `image_cfg.json` 镜像配置文件。

通过嵌套对象的方式，`image_cfg.json` 描述了一个待生成的镜像文件所包含的信息和数据，如下列代码所示，该描述文件分为以下部分：

- 镜像烧录的目标设备描述，详情可查看分区表描述。
- 最终Image 文件描述，包括信息和内容排布，由 `info` 数据，`updater` 数据和 `target` 数据组成。
- 中间文件描述，制作 `image` 的过程中需要生成和使用的临时文件。

```
{
    "spi-nand": {  // Device, The name should be the same with string in image:info:media:type
        "size": "128m",  // Size of SPI NAND
        "partitions": {
            "spl": { "size": "1m" },
            "os":  { "size": "2m" },
            "rodata":  { "size": "4m" },
            "data":  { "size": "28m" },
        },
    },
    "image": {
        "info": {  // Header information about image
            "platform": "d21x",
            "product": "demo128_nand",
            "version": "1.0.0",
            "media": {
                "type": "spi-nand",
                "device_id": 0,
                "array_organization": [
                    { "page": "2k", "block": "128k", "oob": "64" },
        //          { "page": "4k", "block": "256k", "oob": "128" },
                ],
            }
        },
        "updater": {  // Image writer which is downloaded to RAM by USB
            "ddr": {
                "file": "usbupg-ddr-init.aic",
                "attr": ["required", "run"],
                "ram": "0x00103000"
            },
            "bootloader": {
                "file": "bootloader.aic",
                "attr": ["required", "run"],
                "ram": "0x41000000"
            },
        },
        "target": {  // Image components which will be burn to device's partitions
            "bootloader": {
                "file": "bootloader.aic",
                "attr": ["mtd", "required"],
                "part": ["spl"]
            },
            "os": {
                "file": "os.aic",
                "attr": ["mtd", "required"],
                "part": ["os"]
            },
            "res": {
                "file": "app.fatfs",
                "attr": ["mtd", "optional"],
                "part": ["rodata"]
            },
            "app": {
                "file": "page_2k_block_128k_oob_64_data.uffs",
                "attr": ["uffs", "optional"],
                "part": ["data"]
            },
        },
    },
    "temporary": {  // Pre-proccess to generate image components from raw data
        "aicboot": {
            "usbupg-ddr-init.aic": {  // No loader, only PreBootProgram to initialize DDR
                "head_ver": "0x00010001",
                "resource": {
                    "private": "ddr_init.bin",
                    "pbp": "d21x.pbp",
                },
            },
            "bootloader.aic": {
                "head_ver": "0x00010001",
                "loader": {
                    "file": "bootloader.bin",
                    "load address": "0x42000000",
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
                    "entry point":   "0x42000100",
                },
                "resource": {
                    "private": "ddr_init.bin",
                    "pbp": "d21x.pbp",
                },
            },
            "os.aic": {
                "head_ver": "0x00010001",
                "loader": {
                    "file": "d21x.bin",
                    "load address": "0x40000000",
                    "entry point":   "0x40001000",
                    "run in dram": "false",
                }
            },
        },
    },
}
```

2.6.2.1. 分区表描述

`image_cfg.json` 文件的开头描述的是当前要烧录的目标存储设备，以及在设备上的分区配置，以 **spi-nand** 为例：

```
"spi-nand": {  // Device, The name should be the same with string in image:info:media:type
    "size": "128m",  // Size of SPI NAND
    "partitions": {
        "spl": { "size": "1m" },
        "os":   { "size": "2m" },
        "rodata":   { "size": "4m" },
        "data":   { "size": "28m" },
    },
},
```

该示例中各字段具体描述如下表所示：

表 2–5 分区表描述

字段	值类型	描述
<media type>	String	存储设备类型 该字段名字仅可使用此列表所指定的名字。 • **mmc**：eMMC 和 SD Card 设备 • **spi-nand**：SPI NAND 存储设备 • **spi-nor**：SPI NOR 存储设备
size	String	**存储设备**的存储大小 (byte)，可设置。设备的存储大小 (Byte)，单位可为 K、M 或 G，例如，8G。
partitions	Object	**分区表**对象。包含该存储设备的详细分区列表，每一个子对象为一个分区。
offset	String	16 进制字符串。表示该分区 的开始位置离存储设备 的开始位置的偏移（字节）。如果此字段未出现，表示当前分区紧接上一个分区。
size	String	**size**：设备的存储大小 (Byte)，单位可为 K、M 或 G，例如，2m。 最后一个分区可以使用 “-” 表示使用剩下所有的空间。
ubi	Object	**UBI Volume** 对象。当存储设备为 `spi-nand` 时出现，表示当前 MTD 分区是一个 UBI 设备。该对象描述 UBI 设备中的 Volume 表。每一个子对象为一个 **UBI Volume** 。 **UBI Volume**对象： • **offset**：值为 16 进制字符串。表示该 **Volume** 的开始位置离 **MTD** 分区的开始位置的偏移（字节）。如果 **offset**不出现，表示当前 Volume 紧接上一个 Volume。 • **size**：设备的存储大小 (byte)，单位为 K、M 或 G，例如，2m。 最后一个分区可以使用 “-” 表示使用剩下所有的空间。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.6.2.2. Image 文件描述

“image” 对象描述要生成的镜像文件的基本信息，以及要打包的数据。包含几个部分：

- “info”
- “updater”
- “target”

```
"image": {
    "info": {
        ...
    }
    "updater": {
        ...
    }
    "target": {
        ...
    }
}
```

2.6.2.2.1. Info 数据描述

“info” 对象用于描述该烧录镜像的基本信息，这些信息用于生成 Image Header。以 demo128_nand 为例：

```
"info": { // Header information about image
    "platform": "d211",
    "product": "demo128_nand",
    "version": "1.0.0",
    "media": {
        "type": "spi-nand",
        "device_id": 0,
        "array_organization": [
        //    { "page": "4k", "block": "256k" },
                { "page": "2k", "block": "128k" },
        ],
    }
},
```

info 属性	说明
“platform”	字符串，当前项目所使用的 SoC 的名字
“product”	字符串，产品名字、产品型号
“version”	字符串，按照 “x.y.z” 格式提供的版本号，其中 x,y,z 都是数字
“media”	对象，描述存储设备

media 属性	说明
“type”	字符串，取值可参考分区表描述
“device_id”	整数，要烧录的存储设备在 U-Boot 中的索引。
“array_organization”	对象，当存储设备为 “spi-nand” 时使用，描述存储单元的排列结构

array_organization 属性	说明
“page”	当前 SPI NAND 的 Page 大小，取值 “2K”，“4K”，
“block”	当前 SPI NAND 的 Block 大小, 取值 “128K”，“256K”

2.6.2.2.2. Updater 数据描述

Updater 是指进行 USB 刷机或者进行 SD 卡刷机时需要运行的刷机程序，该程序通常由 SPL/U-Boot 实现， 可能与正常启动时所运行的 SPL/U-Boot 相同，也可能不同，因此需要单独列出。

“updater” 对象描述在刷机过程中需要使用到的组件数据，其包含多个子对象，每个子对象即为一个组件 。 其中下列的组件是已知且必要的。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

组件名称	说明
“spl”	第一级引导程序
“”	第二级引导程序，同时也是刷机程序

上述的组件名字并非固定，可根据项目的需要修改、增加或者删除。

Updater 中的组件对象都有以下的配置字段：

Updater 组件属性	说明
“file”	指定该组件的数据来源文件
“ram”	USB 刷机时，指定该文件下载的内存地址
“attr”	该数据对象的属性，可选的内容有: • – “required” : 该数据是必需的，如果指定文件不存在，则生成镜像文件出错。 • – “run” : 该数据是可执行文件，USB 升级时，该数据下载完成之后会被执行。

重要：

“updater” 中组件对象的顺序很重要。

在 USB 升级的过程中，组件数据传输和执行的顺序即为 “updater” 中组件数据出现的顺序， 因此如果组件数据之间有顺序依赖关系，需要按照正确的顺序排布。

2.6.2.2.3. Target 数据描述

“target” 描述要烧录到设备存储介质上的组件 。与 “updater” 中的组件一样，” target” 中出现的组件根据实际需要进行添加，组件的名字也可自行定义。

“target” 中的组件，都有下面的配置字段：

表 2–6 Target 组件属性

属性名	说明
“file”	指定该组件的数据来源文件
“part”	指定该组件被烧录的分区，分区名字通过字符串数组的形式提供。
“attr”	该数据对象的属性，可选的内容有: • “required” : 该组件数据是必需的，如果指定文件不存在，则生成镜像文件出错。 • “optional” : 该组件数据不是必需的，如果指定文件不存在，则在生成镜像文件时忽略该数据对象。 • “burn” : 该组件数据是需要烧录到指定分区当中。 • “mtd” : 表示该组件要烧录的设备是 MTD 设备。 • “ubi” : 表示该组件要烧录的设备是 UBI 设备。 • “block” : 表示该组件要烧录的设备是块设备。

重要：

“target” 中组件对象的顺序

在 USB 升级的过程中，组件数据传输和烧录的顺序即为 “target” 中组件数据出现的顺序。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.6.2.3. 中间文件描述

“temporary” 描述的是镜像文件生成过程中需要生成的中间文件。通过描述数据对象的方式， 描述不同类型的中间文件的生成过程，可用于对组件的签名、加密、再次打包等处理。

当前支持下列两种不同的数据处理：

- “aicboot” ：描述 AIC 启动镜像的生成

AIC 启动镜像

AIC 启动镜像是 BROM 解析和执行的启动程序文件。 当需要在打包过程中生成一个中间的 AIC 启动镜像文件时，需要在 “aicboot” 对象中添加一个子对象， 其对象名字即为生成的文件名字，可配置的内容如下面的示例所示。所列的属性中，只有 “loader” 是必需的， 其他的可根据项目需要进行删减。

```
"aicboot": {
    "usbupg-ddr-init.aic": { // No loader, only PreBootProgram to initialize DDR
        "head_ver": "0x00010001",
        "resource": {
            "private": "ddr_init.bin",
            "pbp": "d21x.pbp",
        },
    },
    "bootloader.aic": {
        "head_ver": "0x00010001",
        "loader": {
            "file": "bootloader.bin",
            "load address": "0x42000000",
                "entry point":  "0x42000100",
        },
        "resource": {
            "private": "ddr_init.bin",
            "pbp": "d21x.pbp",
        },
    },
    "os.aic": {
        "head_ver": "0x00010001",
        "loader": {
            "file": "d21x.bin",
            "load address": "0x40000000",
            "entry point":  "0x40000100",
            "run in dram": "false",
        }
    },
}
```

2.6.3. 镜像烧录媒介及流程

2.6.3.1. SD 卡烧录

芯片支持从 SD 卡的 FAT32 文件系统启动。

要求与步骤

对芯片与板子的要求：

1. 板子 SD 卡接口，并且使用 SDMC1
2. 芯片没有烧录 跳过 SD 卡的 eFuse

对 SD 卡的要求：

1. SD 卡要求只有一个分区
2. SD 卡格式化为 FAT32 文件系统，注意不是 exFAT、或者 FAT16
3. SD 卡最好为专用卡，里面不要放置太多其他文件

执行步骤：

1. 拷贝在编译输出目录 (`images`) 下的两个文件到 SD 卡 FAT32 文件系统的 **根目录**。
2. 确保 `bootcfg` 文件的名字为 `bootcfg.txt`。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3. 将 SD 卡插入板子，重新上电，即可从 SD 卡启动到 Tiny_SPL，并执行烧录。

4. 烧录完成时，需要拔出 SD 卡，然后重新上电启动。

> **注：**
>
> 烧录完成平台并不会主动重启，以防重复进入 SD 卡烧录模式。在 BootLoader 的配置中，勿开启 **Enable the interrup to fSDMC**，否则可能导致烧录阶段无法识别到 SD 卡。

编译配置

SDK 提供的配置，默认关闭该功能。这里进行配置使能以及注意项的说明。

SPI NAND/NOR 方案

使能 SDFAT32 烧录功能，只需要在 BootLoader 的 menuconfig 中勾选配置:

```
Bootloader options
    Upgrading
        [*] Upgrading by SD Card
            (1)   SDMC controller id for SD Card        # 对应板卡的 SDMC 控制器
```

开启 SD 卡烧录功能后，编译完成生成的 `bootloader.aic` 文件大小不能超过 126K，若超过则可以关闭以下配置来减小 CODE SIZE。

1. AIC_BOOTLOADER_CMD_SPI_NAND
2. AIC_BOOTLOADER_CMD_MTD
3. AIC_BOOTLOADER_CMD_MEM

2.6.3.2. U 盘升级

芯片支持通过 U 盘的 FAT32 文件系统进行升级。

要求与步骤

对芯片与板子的要求：

1. 板子使用 USB 接口。
2. 芯片烧录过镜像，且镜像支持 U 盘升级

对 U 盘的要求：

1. U 盘只有一个分区。
2. U 盘格式化为 FAT32 文件系统，注意不得使用 exFAT 或者 FAT16 系统。
3. 最好为专用 U 盘，且未放置其他文件。

执行步骤：

1. 在编译输出目录 (`images`) 中，将两个文件拷贝到 U 盘 FAT32 文件系统的根目录。
2. 确保 bootcfg 文件的名字为 `bootcfg.txt`。
3. 将 U 盘插入板子，重新上电，启动过程中则会自动检测 U 盘是否插入，检测到后进入烧录模式进行升级。
4. 烧录完成时，需要拔出 U 盘，然后重新上电启动。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

注：

烧录完成平台并不会主动重启，以防重复进入 U 盘烧录模式。

编译配置

SDK 提供的配置，默认关闭该功能。这里进行配置使能以及注意项的说明。

SPI NAND/NOR 方案

使能 SDFAT32 烧录功能，只需要在 menuconfig 中勾选配置项 **AICUPG_UDISK_ENABLE** 即可:

```
Bootloader options  --->
    aicupg setting  --->
        [*] aicupg udisk upgrade on
```

同时设置 **AICUPG_USB_CONTROLLER_MAX_NUM=1**。可用的最大 USB HOST 控制器数量，根据实际芯片平台进行设置。

开启 U 盘烧录功能后，编译完成生成的 `bootloader.aic` 文件大小不能超过 126K，若超过则可以关闭以下配置来减小 CODE SIZE。

1. AIC_BOOTLOADER_CMD_SPI_NAND
2. AIC_BOOTLOADER_CMD_MTD
3. AIC_BOOTLOADER_CMD_MEM

注：

在打包镜像前，需要确保 `target/<IC>/<Board>/pack` 目录下的 `image_cfg.json` 文件中 **device_id** 与存储介质所使用的控制器 id 所对应。

例如 eMMC 存储介质使用的控制器为 SDMC0，则应该将 **device_id** 的值设为 0， 如果 eMMC 存储介质所使用的控制器为 SDMC1，则应该将 **device_id** 设为 1。

2.7. GDB 调试

GDB 调试工具及其功能描述如下：

表 2–7 工具介绍

工具	说明
T–HeadDebugServer	运行 GDB Server
AiBurn	镜像烧写工具
AIC–JTAG	AIC 调试器
JTAG – SDMC 转接线	仅用于调试器连接 SDMC 接口使用
Eclipse	适配 Baremetal SDK 的 IDE
riscv64–unknown–elf–gdb.exe	调试工具，位于 `$(SDK)tool/riscv64-gdb/bin/`
d21x.elf	Eclipse 工程目录下生成的符号链接表
Tabby	用于控制调试串口、发送指令

2.7.1. 准备工作

执行 GDB 调试前，需要按照以下步骤准备相关软件环境：

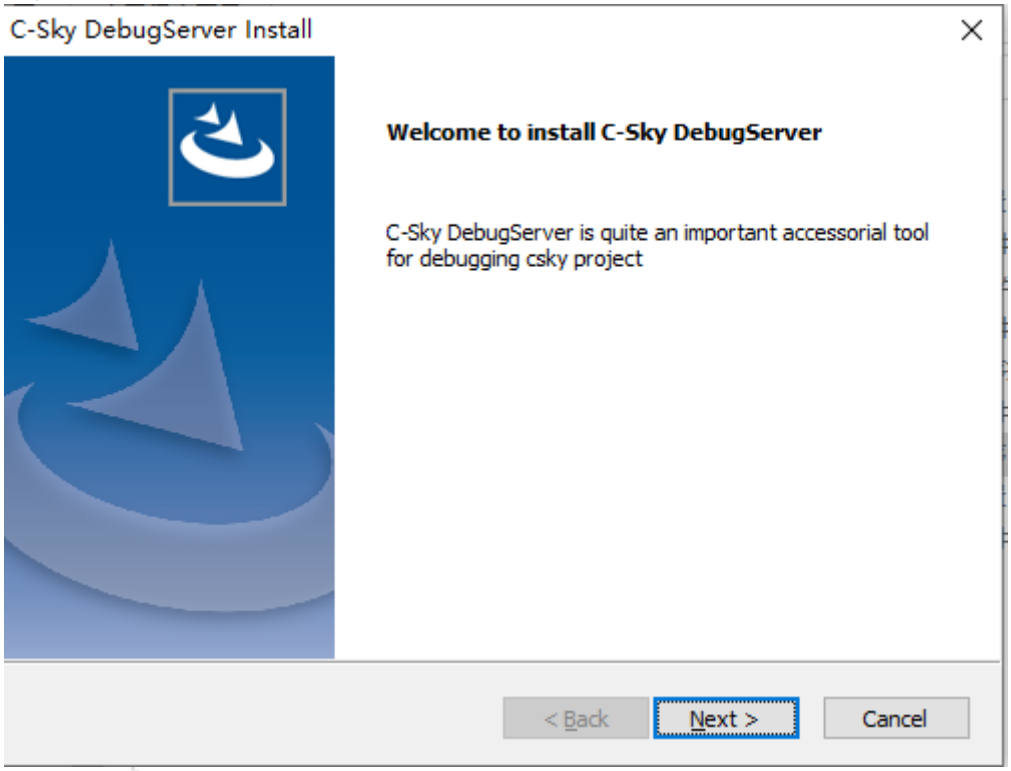
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

1. 安装 T-HeadDebugServer（以下简称 DebugServer）

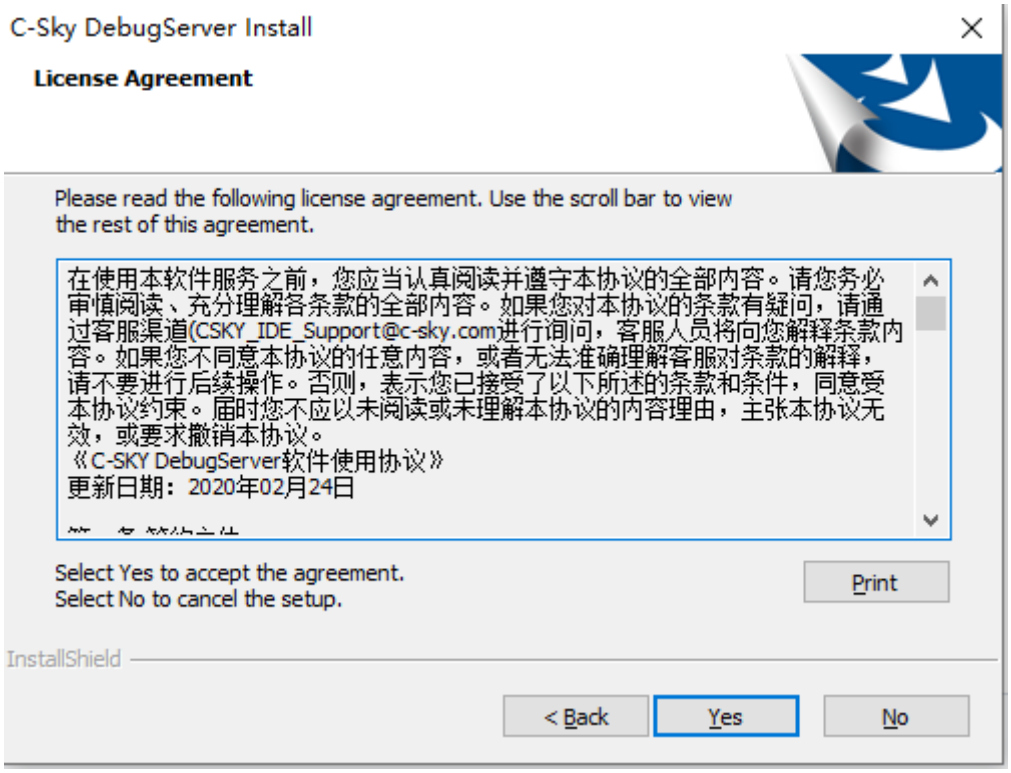
   安装包自带 AIC-JTAG 驱动。

   a. 解压，运行 `setup.exe`。

   b. 在弹出页面中，点击 **Next**：

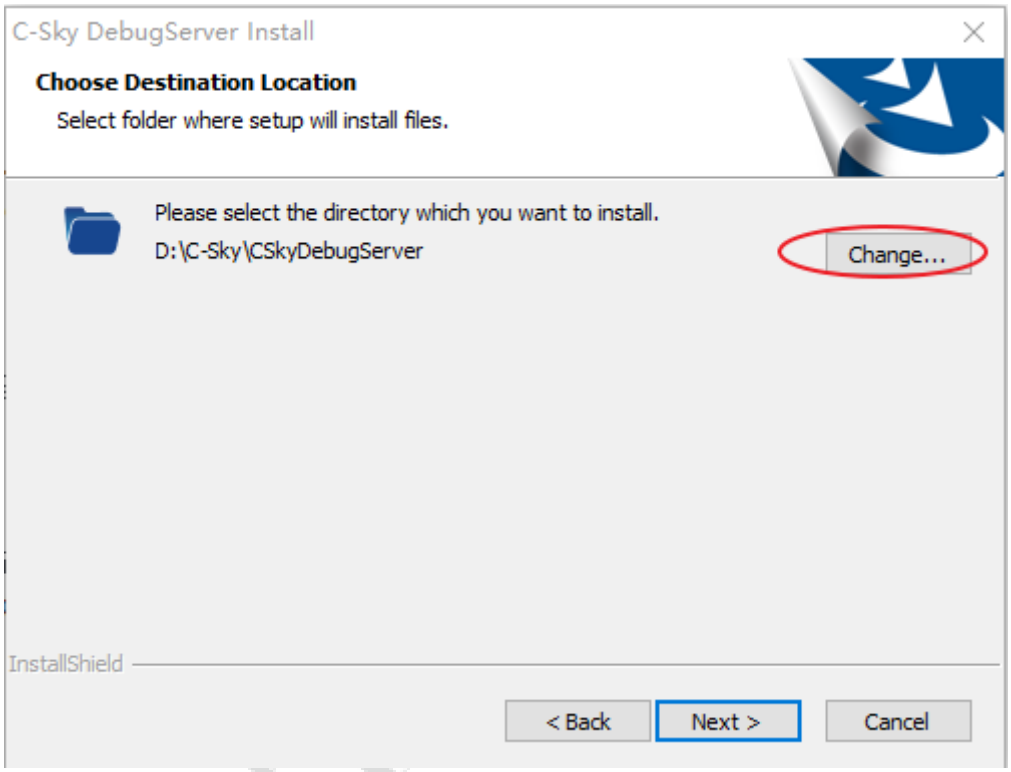


   c. 在用户协议界面中点击 **Yes**：

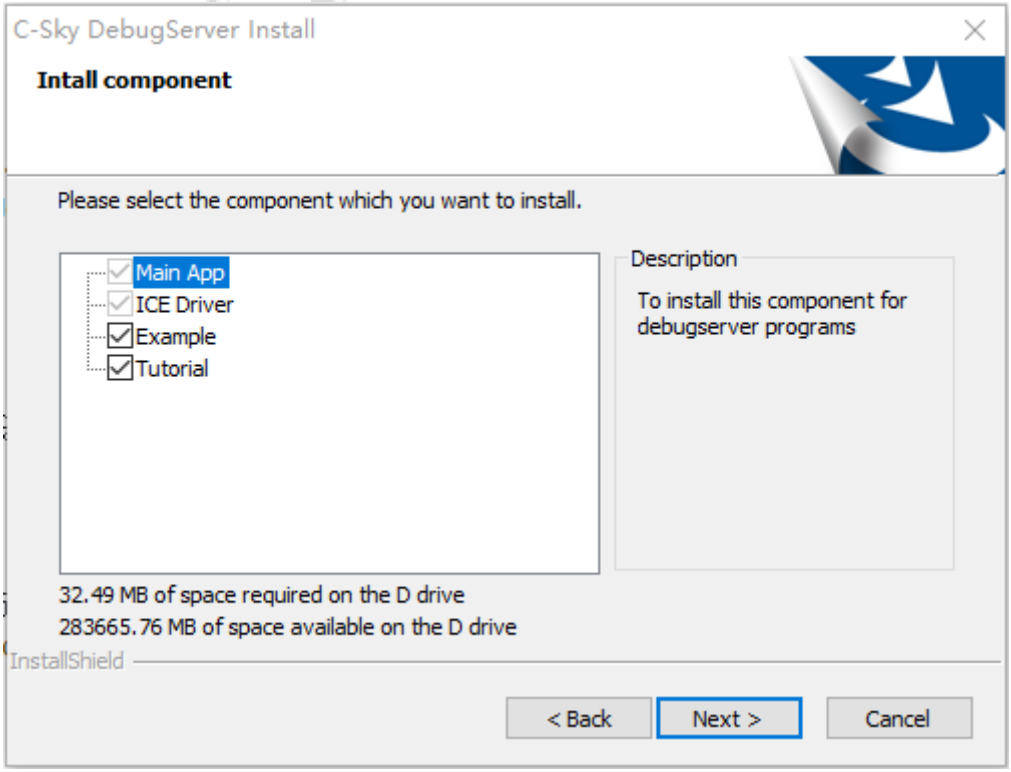


   d. 在安装路径页面，按照实际需求，修改路径：

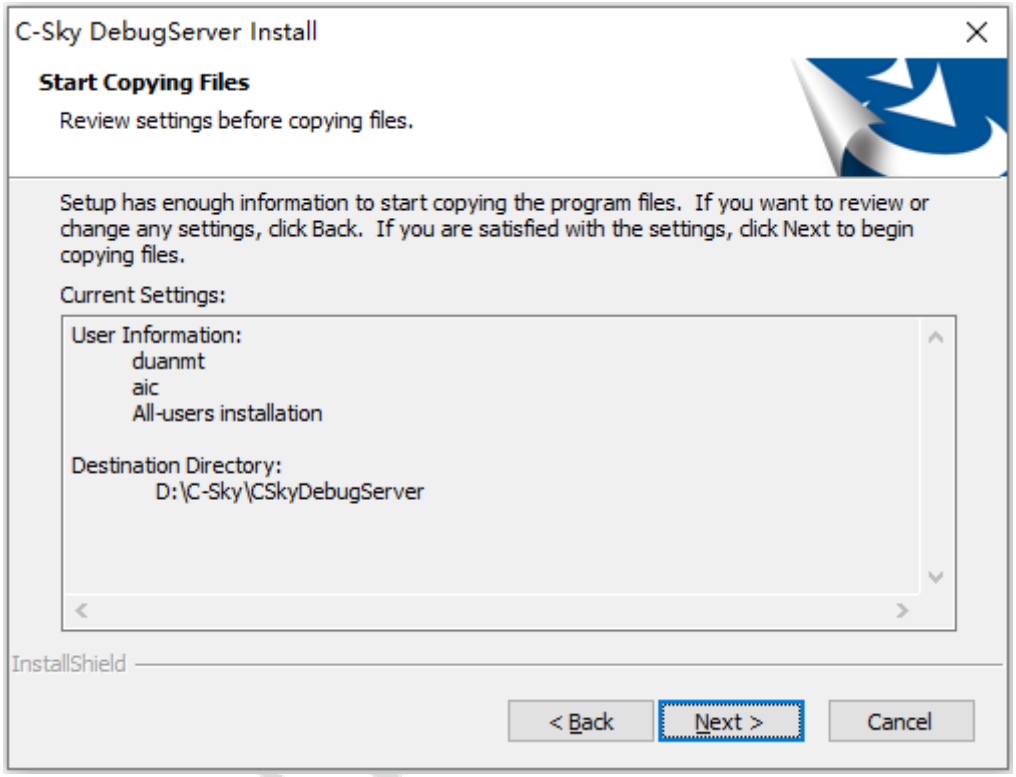
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



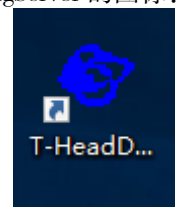
e. 在安装内容页面，建议全选，继续点 **Next**：



Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

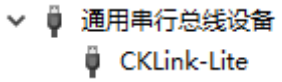


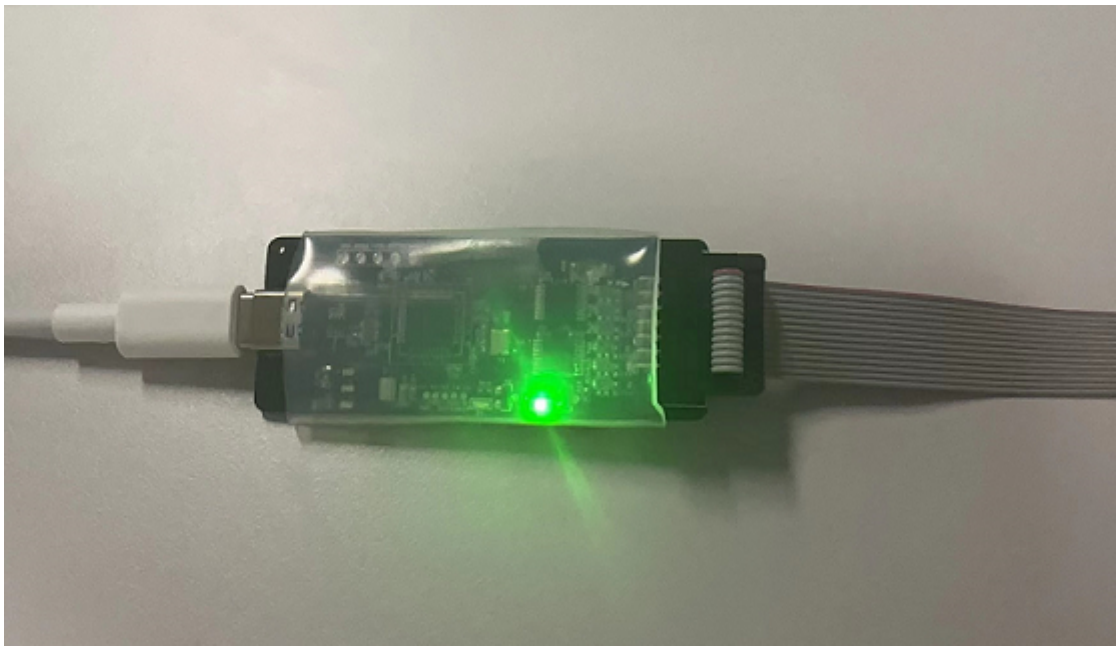
等待完成安装后，桌面会自动创建了一个 DebugServer 的图标：



2. 将 AIC 调试器通过 USB-TypeC 连接 PC。

   成功识别后，AIC 调试器绿灯会亮起。





Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.7.2. SDK 配置

Baremetal SDK 支持两种硬件接口进行调试，分别是 JTAG 和 SDMC 口。 为避免 GPIO 冲突，需要配置 `ddr_init.json` 和 `xxx_defconfig`。

> **注：**
>
> 配置前，确保已经加载目标工程。

根据接口的类型，选择相应的连接和配置操作流程：

- **使用 JTAG 口**，配置 JTAG 接口的流程如下：

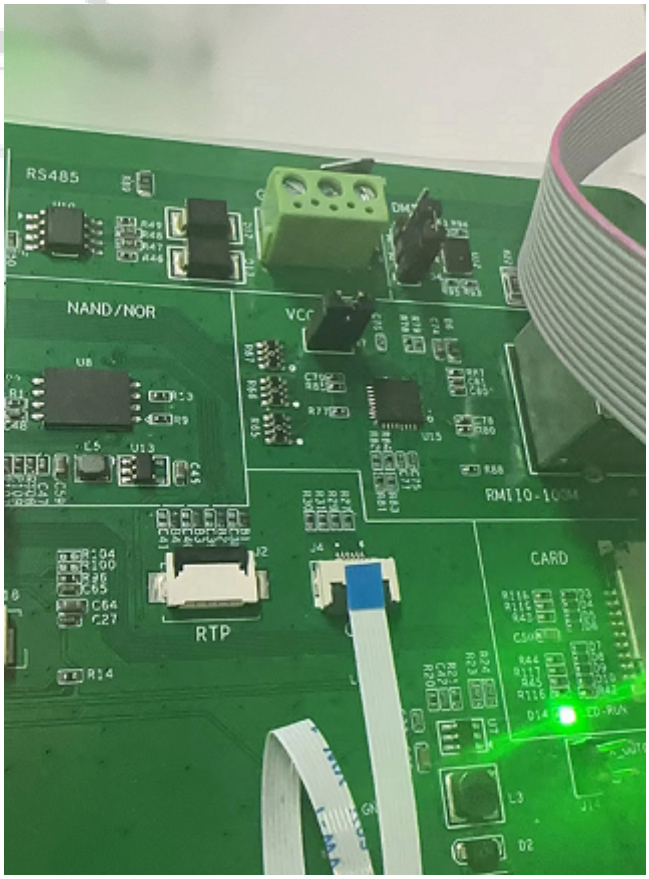
  1. 运行下列命令：

     ```
     scons --menuconfig
     ```

  2. 关闭 I2C 以及 Touch Panel

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

  使用 JTAG 口，需断开 CTP 触屏排线。



- **使用 SD 调试 口**

  1. 打开 `target/<CPU>/<board>/pack/ddr_init.json` ，配置 **`jtag_only`** 和调试口引脚：

     ```
     "jtag": {
         "jtag_only": "1", // 1: Boot code stop in PBP after DDR init and jtag init
         "main": {
     ```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
            "jtag_id": "0",
            //"jtag_do_pin_cfg_reg": "0x187000A0", // PA8
            //"jtag_do_pin_cfg_val": "0x336",
            //"jtag_di_pin_cfg_reg": "0x187000A4", // PA9
            //"jtag_di_pin_cfg_val": "0x336",
            //"jtag_ms_pin_cfg_reg": "0x187000A8", // PA10
            //"jtag_ms_pin_cfg_val": "0x336",
            //"jtag_ck_pin_cfg_reg": "0x187000AC", // PA11
            //"jtag_ck_pin_cfg_val": "0x336",

            "jtag_do_pin_cfg_reg": "0x1870028C", // PC3
            "jtag_do_pin_cfg_val": "0x336",
            "jtag_di_pin_cfg_reg": "0x18700284", // PC1
            "jtag_di_pin_cfg_val": "0x336",
            "jtag_ms_pin_cfg_reg": "0x18700280", // PC0
            "jtag_ms_pin_cfg_val": "0x336",
            "jtag_ck_pin_cfg_reg": "0x18700294", // PC5
            "jtag_ck_pin_cfg_val": "0x336",
        },
    },
```

2. 配置接口：

   a. 运行 scons --menuconfig 命令，打开配置界面。

   b. 关闭 SDMC1 选项。

```
Board options  --->
        [] Using SDMC1
```

2.7.3. 连接开发板

1. 编译 SDK。
2. 使用 AiBurn 烧录编译生成的镜像，详见配置烧录镜像。
3. 开发板连接串口、AIC-JTAG 和电源线。

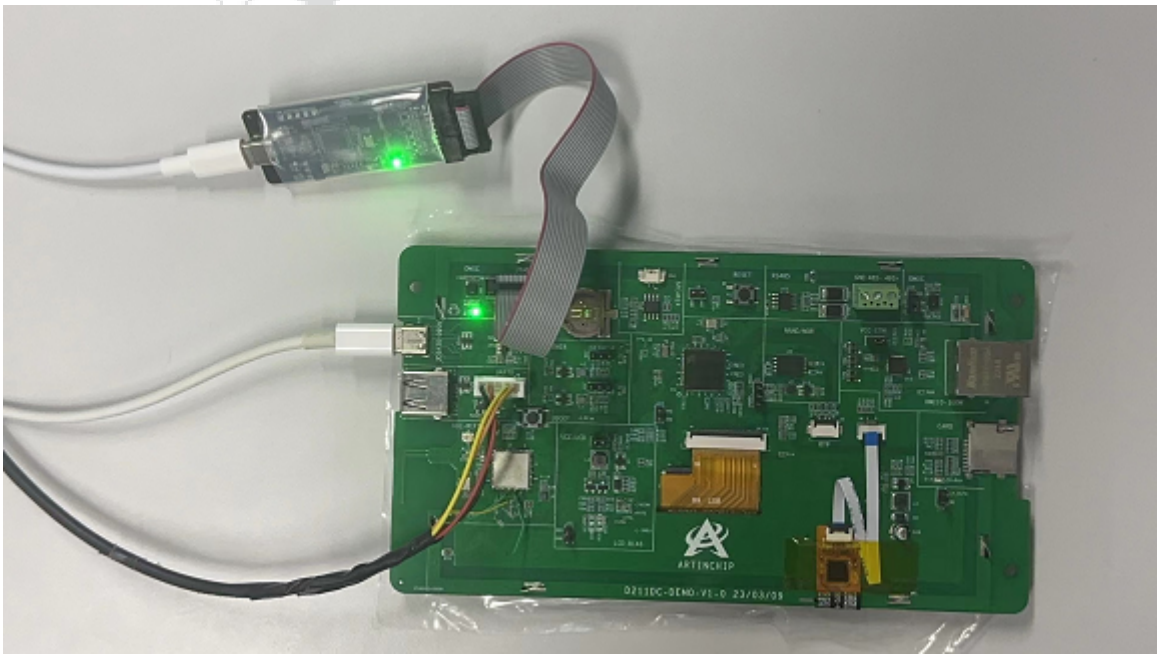


图 2-2 开发板连接 JTAG 接口

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

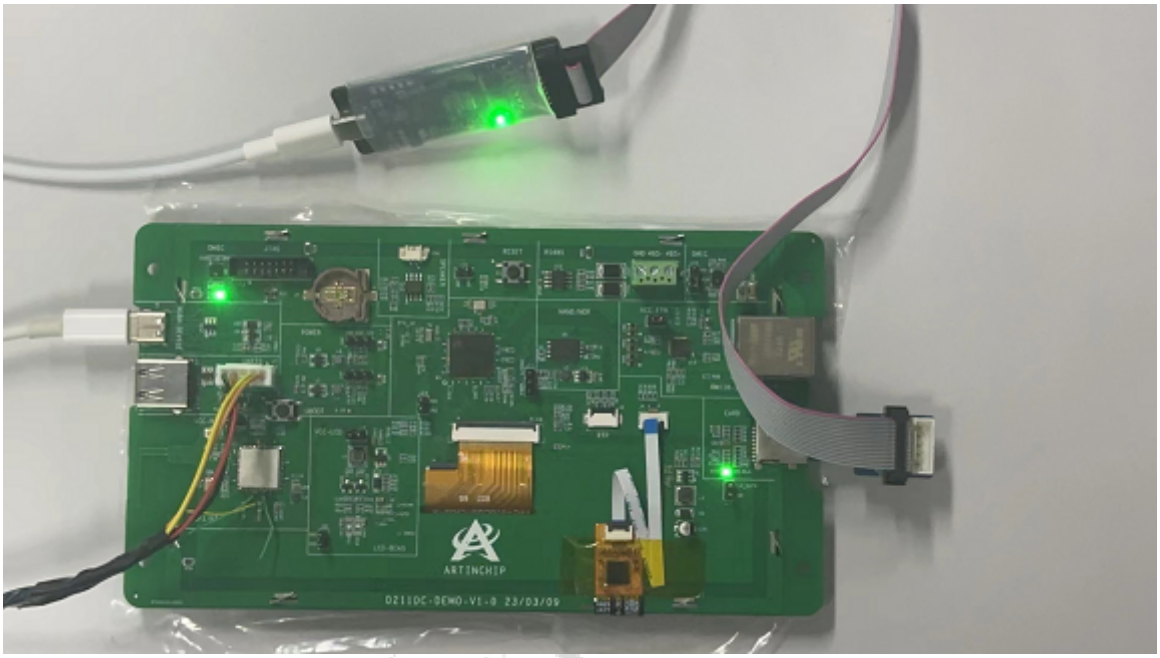
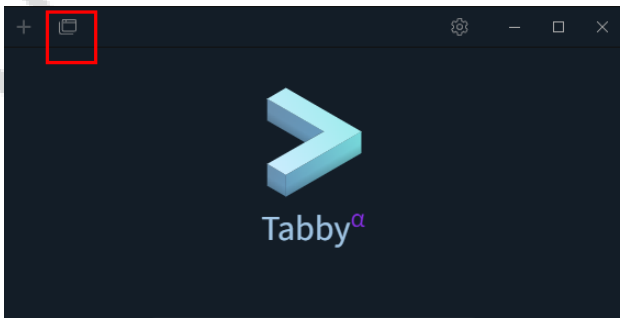


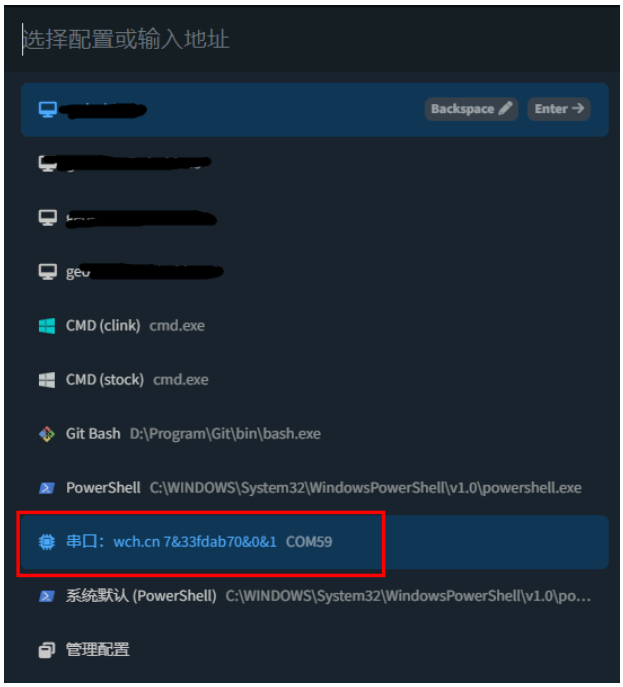
图 2–3 开发板连接 SDMC 接口

4. 使用 Tabby，打开调试串口。

   a. 打开选择页面。



   b. 选择串口 COM 口。



Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

c. 设置波特率，默认 115200。

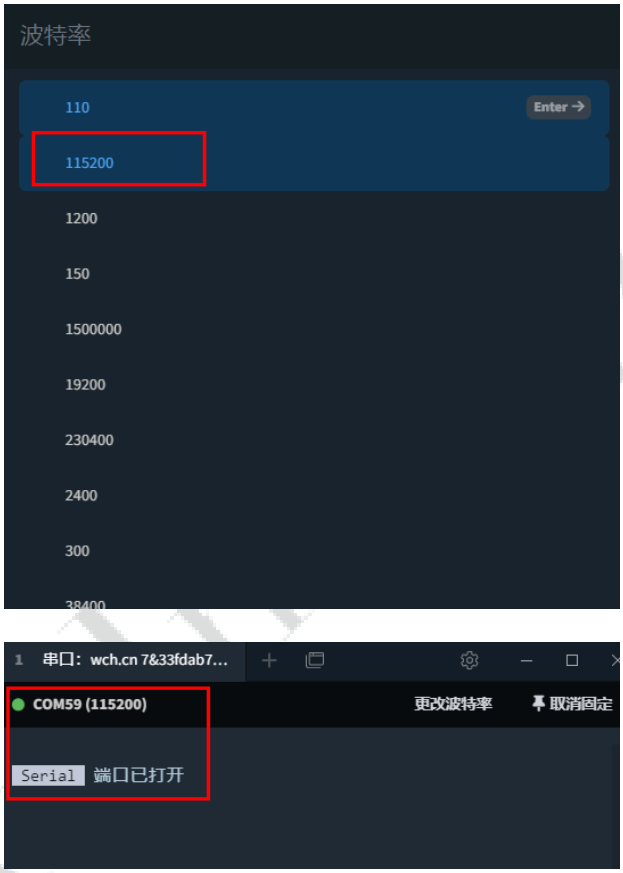


图 2–4 打开成功

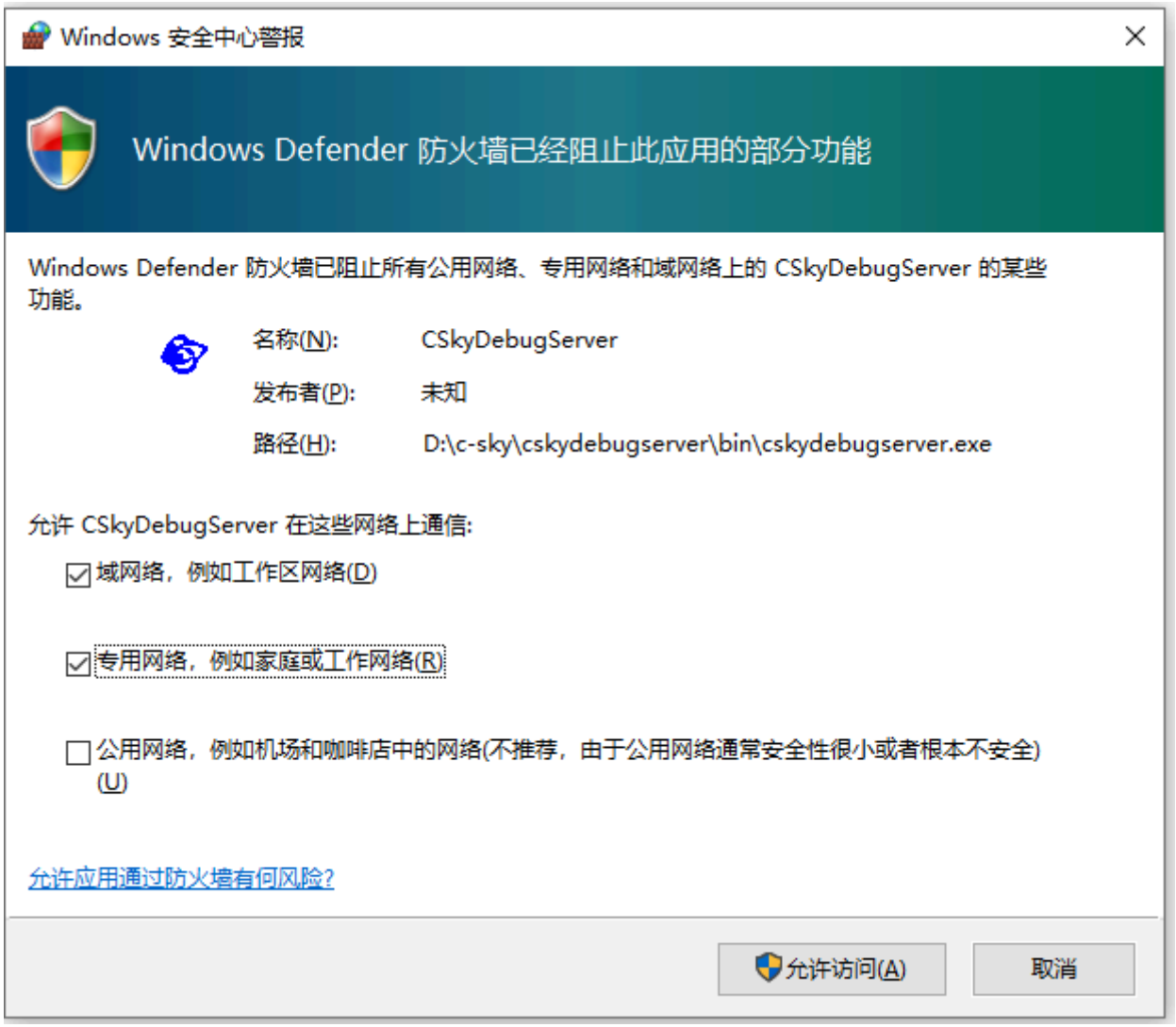
5. 将系统句柄放置在调试串口窗口，点击一下调试串口所在窗口。

6. 在键盘上按住 **Ctrl + C**，同时给开发板上电，串口打印如下：

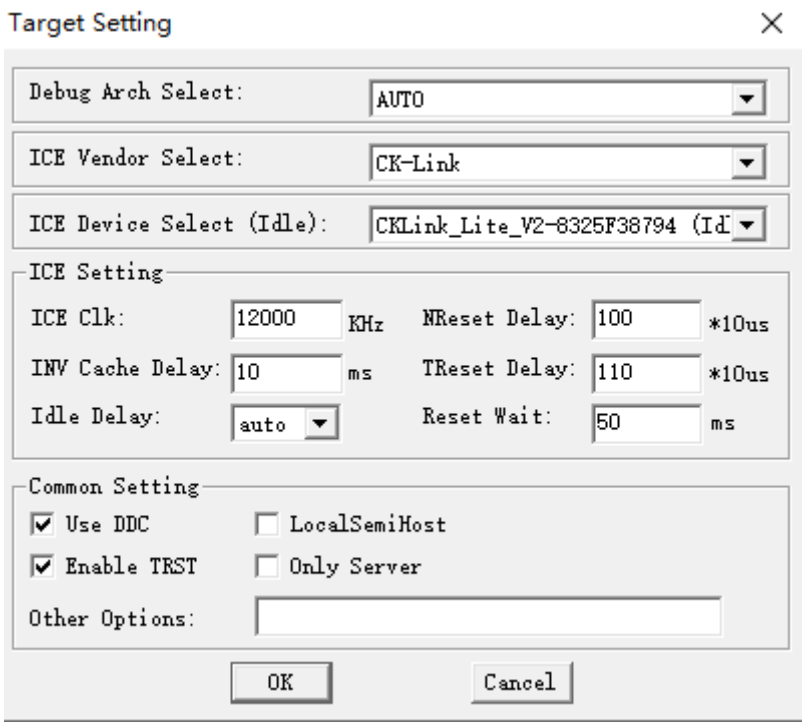
```
Pre-Boot Program ... (2023-08-08 11:29:35 13a563f)
No DDR info
Going to init DDR2. freq: 504MHz
DDR2 initialized
120186 134875 162258
PBP done

tinySPL [Built on Aug 21 2023 17:12:58]
Boot device = 5(BD_SPINAND)
nand read speed: 1346320 byte, 76998 us -> 17075 KB/s
aic@tinySPL #
aic@tinySPL #
```

7. 运行 DebugServer，首次运行会有安全警告，点击**允许访问**

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

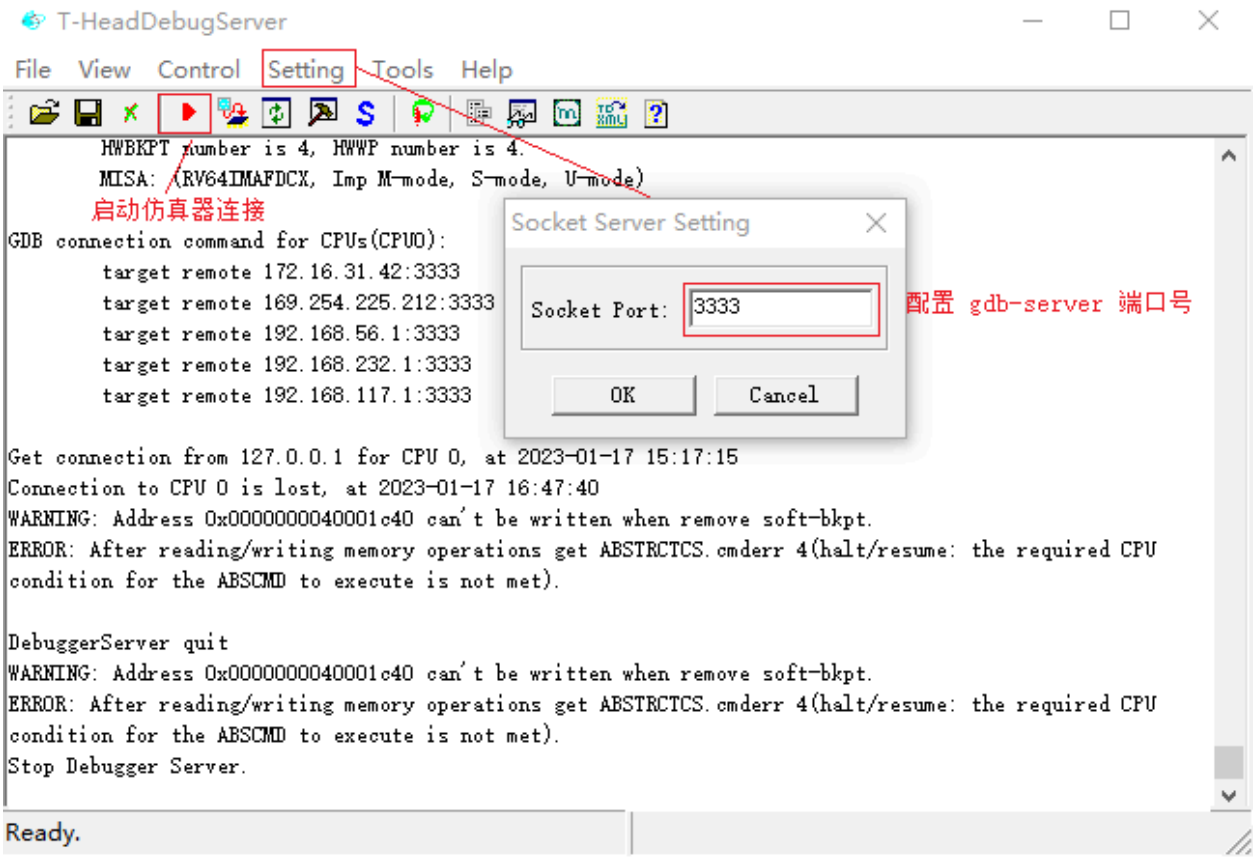


8. 配置 **DebugServer Setting > Setting > Target Setting**。



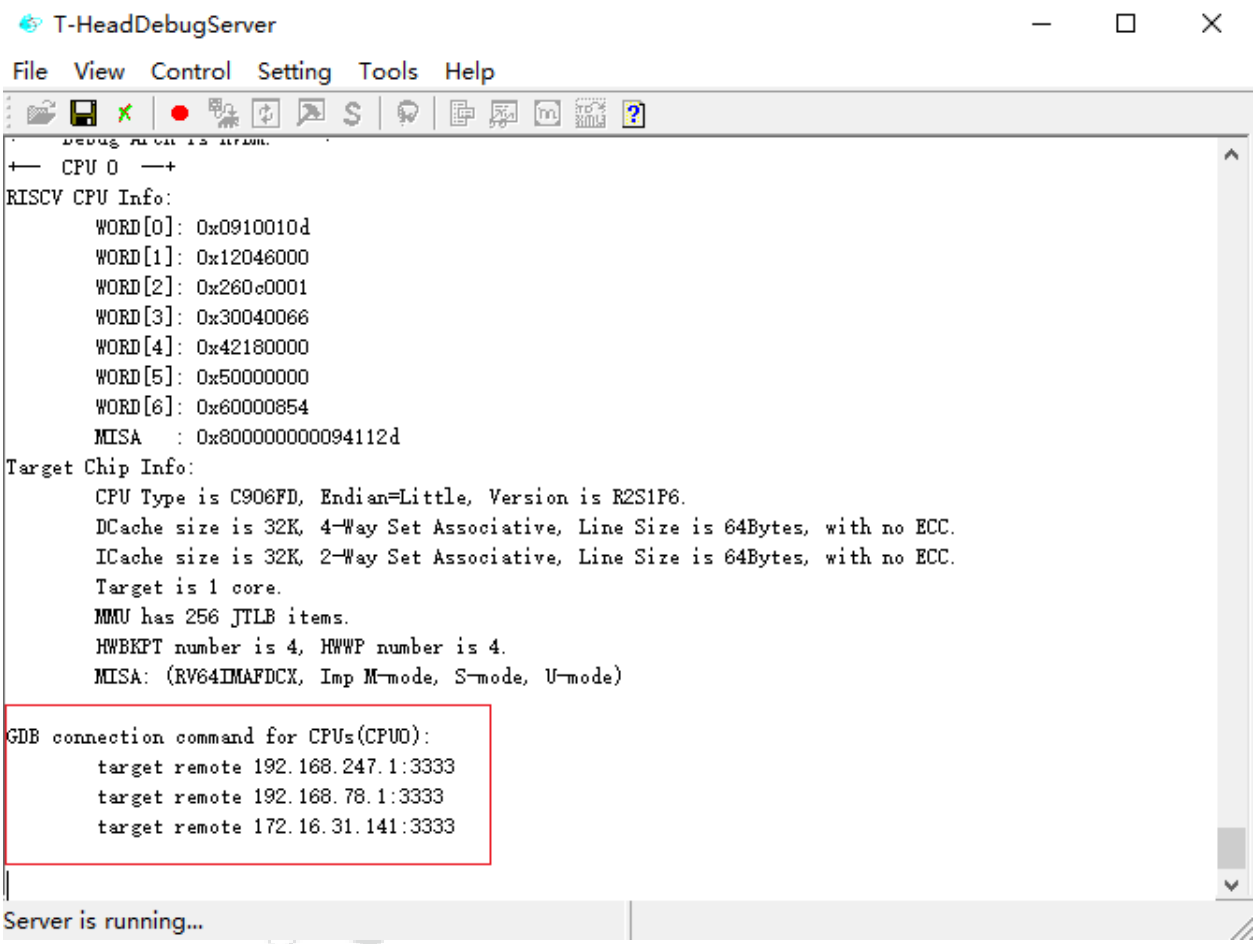
Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

9. 配置 DebugServer 的端口号，例如 3333。



10. 启动仿真器连接，显示本地 IP 及已配置的端口（3333），则表示连接成功：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



11. 连接成功后，即可通过 IDE 或命令行进行调试。

调试

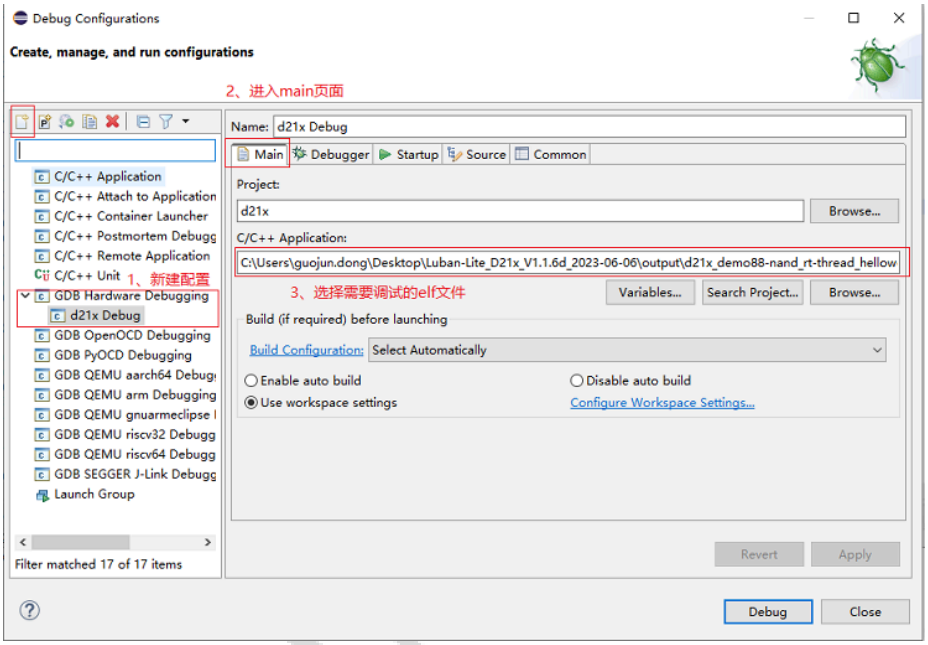
2.7.4. Windows 下使用 Eclipse 调试

采用 Eclipse 配合 GDB 的调试方案，方便用户上手：

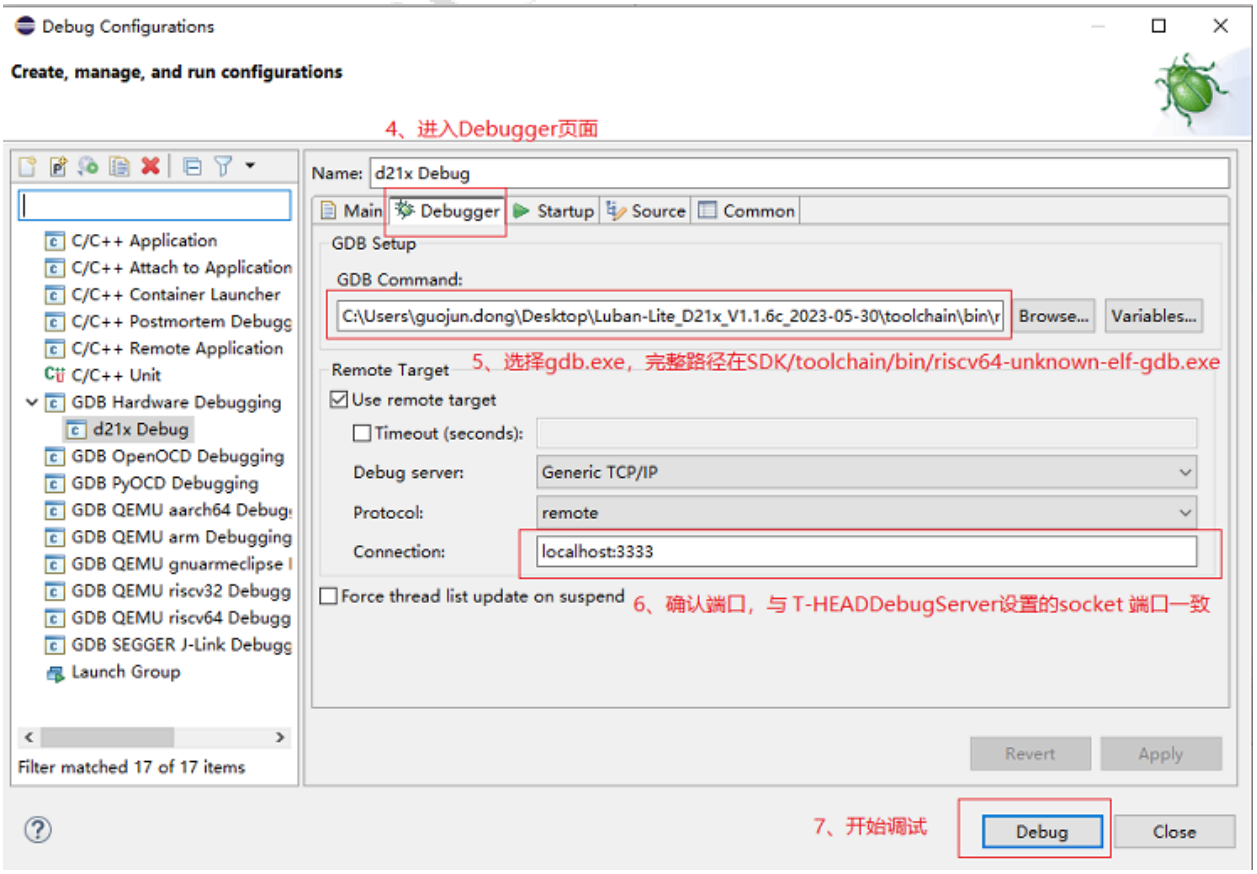
> **注：**
>
> 使用 Eclipse 调试前，需先完成 Eclipse 工程导入及编译。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

1. 配置 Debug Configurations – Main

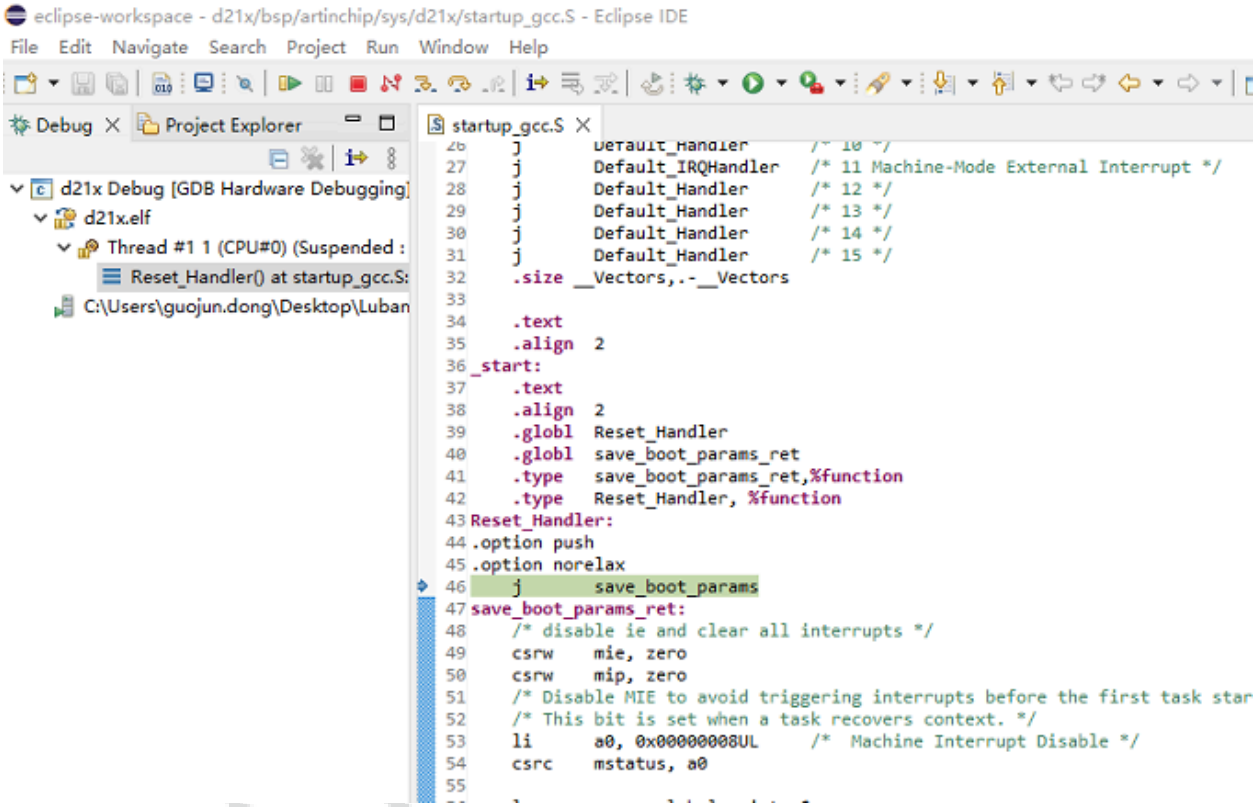


2. 配置 Debug Configurations – Debugger



Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3. Eclipse 进入 Debug 模式



Eclipse 调试常用快捷键：

- F5：单步进入，进入函数内部
- F6：下一行代码，逐行执行
- F7：返回值调用处的下一行代码
- F8：继续运行，跳过当前中断点
- F11：调试并启动程序

2.7.5. Linux 下使用 GDB 调试

> **注：**
>
> 在 Linux 下调试前，需确保与 DebugServer 所在的 Windows 系统可以 ping 通。

1. 在 SDK 根目录编写 `jtag-debug.sh` 脚本：

```
$cat jtag-debug.sh

target remote 172.16.31.141:3333    #此处 IP 为 DebugServer 运行 PC 的 IP，端口为 DebugServer 配置的端口

load ./output/d13x_demo88-nor_rt-thread_helloworld/images/d13x.elf  # 对应项目的 elf 文件
file ./output/d13x_demo88-nor_rt-thread_helloworld/images/d13x.elf
```

2. 在 SDK 根目录下运行 GDB ：

```
./toolchain/bin/riscv64-unknown-elf-gdb -x ./jtag-debug.sh
GNU gdb (Xuantie-900 elf newlib gcc Toolchain V2.6.1 B-20220906) 10.0.50.20200724-git
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x0000000000106b3a in ?? ()
Loading section .text, size 0xa4780 lma 0x40000100
        sectio--Type <RET> for more, q to quit, c to continue without paging--
--Type <RET> for more, q to quit, c to continue without paging--
        section progress:  41.5%, total progress:   inf%
        section progress:  46.2%, total progress:   inf%
        section progress:  50.4%, total progress:   inf%
        section progress:  53.9%, total progress:   inf%
        section progress:  63.3%, total--Type <RET> for more, q to quit, c to continue without paging--
        section progress:  93.5%, total progress:   inf%--Type <RET> for more, q to quit, c to continue without paging--
        section progress: 100.0%, total progress:   inf%
Loading section .rodata, size 0x843b8 lma 0x400a4880
        section progress:  27.7%, total progress:   i--Type <RET> for more, q to quit, c to continue without paging--
        sect--Type <RET> for more, q to quit, c to continue without paging--
        section progress: 100.0%, total progress:   inf%
Loading section .data, size 0x32e0 lma 0x40128c40
--Type <RET> for more, q to quit, c to continue without paging--
        section progress: 100.0%, total progress:   inf%
Start address 0x0000000040000100, load size 1228312
Transfer rate: 107 KB/sec, 3988 bytes/write.
```

显示以上信息，表示已经进入 GDB 调试模式。

2.8. 引脚配置

在 Baremetal SDK 中，修改同一个 `pinmux.c` 文件，可以配置相同板卡的 GPIO 引脚功能。

例如，对于以下 D133CBS RISC-V Kunlun Pi 开发板，可以使用`target/d13x/kunlunpi88-nor/pinmux.c` 文件来配置 GPIO 引脚功能。

> **注：**
>
> 每一个需要使用到的引脚都应显式地设置为指定功能，包括通用 GPIO、I2C、UART、SPI 等功能。

2.8.1. 配置 `pinmux.c`文件

在 `pinmux.c`中，根据需要修改结构体数组来配置每个引脚的功能。 `pinmux.c` 文件路径： `target/<CPU>/<board>/pinmux.c`。

> **注：**
>
> 项目中的 BootLoader 和 RT-Thread 共享 pinmux 配置， 因此在修改 pinmux.c 之后，必须先编译 BootLoader， 再编译 RT-Thread，否则 pinmux 配置可能无法生效。
>
> 自 V1.0.4 之后，Baremetal 支持使用 mb 命令，一键编译 BootLoader 和 RT-Thread。

在 `pinmux.c`中，通过结构体数组，对所有 pin 脚的功能进行管理，并在系统启动时统一做配置。

```
struct aic_pinmux aic_pinmux_config[] = {
    ...
};
```

每个 pin 脚的配置，使用结构体描述。例如，PA.0 和 PA.1 引脚的配置如下：

```
struct aic_pinmux
{
    unsigned char       func;    //功能编号
    unsigned char       bias;    //内部上下拉设置，一般无需修改
    unsigned char       drive;   //驱动能力，若需要修改，请联系专业人士确认
    char *              name;    //引脚名称字符串，例如 "PA.0"
};
```

```
struct aic_pinmux aic_pinmux_config[] = {
#ifdef AIC_USING_UART0
    /* uart0 */
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
    {5, PIN_PULL_DIS, 3, "PA.0"},    //PA.0 配置功能 5，用作串口 0
    {5, PIN_PULL_DIS, 3, "PA.1"},    //PA.1 配置功能 5，用作串口 0
#endif
...
};
```

PA.0 和 PA.1 引脚功能均设置为 5，用作串口 0。注意，同一个引脚只能配置一个功能。

> **注：**
>
> 若要配置某 pin 脚用于通用 GPIO 功能，则需将功能配置为 1。关于引脚功能的详细描述，可参考用户手册中的引脚复用功能表。

2.8.2. 使用检查工具 `pinmux_check`

为检查引脚功能配置，Baremetal 提供 `pinmux_check` 工具。具体使用流程如下所示：

1. 在 RT-Thread 配置中，选择启用 pinmux_check 工具，执行编译和烧录：

```
Local packages options  --->
    ArtInChip packages options  --->
        [*] aic-pinmux-check  --->
            [*]   Enable pinmux check tools
```

2. 系统启动之后，输入 pinmux_check-h 命令，查看帮助菜单。

```
pinmux_check -h
```

```
Compile time: Apr 23 2024 15:39:22
Usage: pinmux_check [options]
        -s, --start            print PIN start from PXn . Default as start from PA0
        -c, --count            print PIN count.
        -h, --help

Example:
    pinmux_check -s PA0 -c 2
```

3. 查看 PA0、PA1 的引脚配置。

```
pinmux_check -s PA0 -c 2
```

```
PA0: 0x00000035, fun[5], drv[3], pull[0], IE[0], OE[0], IE_FORCE[0]    # fun[5] 表示 PA0 配置为功能 5, drv[3]表示驱动能力为 3;
PA1: 0x00000035, fun[5], drv[3], pull[0], IE[0], OE[0], IE_FORCE[0]    # fun[5] 表示 PA1 配置为功能 5, drv[3]表示驱动能力为 3;
```

2.9. 添加驱动

本节以添加 GT911 电容屏驱动为例，介绍外设驱动的添加方法。

2.9.1. 添加外设

1. **获取源码**

   从 https://github.com/RiceChen/gt911 下载 GT911 的 RT-Thread 驱动源码位，并将其解压至 `bsp/peripheral/touch` 目录下。

   源码结构如下：

```
touch$ tree
.
├── gt911
│   ├── inc
│   │   └── gt911.h
│   ├── Kconfig
│   └── src
│       └── gt911.c
└── SConscript
```

2. **修改 `SConscript` 文件**

   在`bsp/peripheral/touch/SConscript` 文件中，添加对 GT911 驱动源码的引用，具体操作内容如下：

```
Import('AIC_ROOT')
Import('PRJ_KERNEL')
from building import *
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
cwd = GetCurrentDir()
src = Glob('*.c')
CPPPATH = []

if GetDepend('AIC_TOUCH_PANEL_GT911'):
    CPPPATH.append(cwd + '/gt911/inc')
    src += Glob('gt911/src/*.c')

group = DefineGroup('touch', src, depend = [''], CPPPATH = CPPPATH)

Return('group')
```

如果后续需要添加其他 touch 的驱动，可参考 GT911 模板将其添加到 SConscript 文件中。

3. **修改 `pinmux.c`文件**

根据原理图，在 `target/<cpu>/<board>/pinmux.c` 中添加 RST、INT、SCK 和 SDA 引脚的定义：

```
struct aic_pinmux aic_pinmux_config[] = {
...
#ifdef AIC_USING_I2C3
    {1, PIN_PULL_DIS, 3, "PA.8"},  //RST
    {1, PIN_PULL_DIS, 3, "PA.9"},  //INT
    {4, PIN_PULL_DIS, 3, "PA.10"}, //SCK
    {4, PIN_PULL_DIS, 3, "PA.11"}, //SDA
#endif
...
};
```

> **注：**
>
> 每个板卡的 `pinmux.c`都定义了各个端口的 GPIO 引脚及功能，新增接口前应检查。

根据原理图，GT911：

- 使用 I2C3 接口 作为驱动。
- 除了 I2C 的 SCK 和 SDA 引脚 以外，还需要用到 RST 和 INT 引脚。

4. **配置源码 `Kconfig`**

在驱动源码 Kconfig `bsp/peripheral/touch/gt911/Kconfig` 文件夹中，添加 GT911 驱动的配置选项。内容如下：

```
$ cat Kconfig

menu "Gt911 touch panel options"

config AIC_TOUCH_PANEL_GT911
    bool "Using touch panel gt911"
    default n
    select AIC_I2C_DRV

config AIC_TOUCH_PANEL_GT911_I2C_CHA
    string "gt911 using i2c channel index"
    default "i2c3"
    depends on AIC_TOUCH_PANEL_GT911

config AIC_TOUCH_PANEL_GT911_RST_PIN
    string "gt911 reset pin"
    default "PA.8"
    depends on AIC_TOUCH_PANEL_GT911

config AIC_TOUCH_PANEL_GT911_INT_PIN
    string "gt911 irq pin"
    default "PA.9"
    depends on AIC_TOUCH_PANEL_GT911

endmenu
```

5. 添加**外设驱动 `Kconfig`** 的引用。

在 `bsp/peripheral/Kconfig` 文件中，添加对驱动源码 Kconfig 路径的引用：

```
...
#-------------------------------------------
# touch panel driver global option
#-------------------------------------------
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
menu "Touch Panel Support"
source "bsp/peripheral/touch/gt911/Kconfig"
endmenu

...
```

6. **配置 menuconfig**

   运行 scons --menuconfig 或 me (OneStep 命令)，打开 **I2C3** 并配置为 **Master**，使能 GT911 驱动。

```
Board options  --->
    [*] Using I2c3
        I2c3 Parameter  --->
            I2c3 Master && Slave  (Master)  --->
Drivers options  --->
    Peripheral  --->
        Touch Panel Support  --->
            Gt911 touch panel options  --->
                [*] Using touch panel gt911
                (i2c0) gt911 using i2c channel index
                (PA.10) gt911 reset pin
                (PA.11) gt911 irq pin
```

7. **编译**

   完成驱动添加和配置后，使用命令 scons --menuconfig 或 m (OneStep 命令) 对 SDK 进行编译。

8. **验证**

   烧写镜像，系统启动之后，通过 list_device 命令 查看设备节点是否已成功枚举，如下所示：

```
list_device
```

```
device           type         ref count
-------- -------------------- ----------
...
gt911    Touch Device         1
...
```

   如果列表中出现 gt911 设备，表示已成功添加该设备。

2.10. 添加应用

GT911 电容屏测试程序是一个硬件与软件结合的复杂项目，主要用于测试和验证电容触摸屏的性能和功能。

本节以 GT911 电容屏测试程序为例，演示添加应用的步骤和编译流程。

2.10.1. 添加源码

在 bsp 目录下执行下列命令，可查看 GT911 电容屏测试程序的目录结构：

```
tree
```

输出示例如下：

```
.

├── Kconfig                     # 编译配置文件
...
├── SConscript                  # 固定模板
└── examples
    ├── SConscript              # 固定模板
    ...
    ├── test-ctp
    │   ├── test_gt911.c        # 源代码
    │   └── SConscript          # 源码结构文件
    ...
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- Kconfig 文件：负责编译配置，定义了需要被编译进最终操作系统镜像中的文件，允许根据需求开启或关闭特定功能，从而提供灵活性并减少不必要的资源占用。
- SConscript 文件：负责指导编译系统编译源码的脚本文件，定义了构建过程的规则，例如依赖关系、编译顺序等。
- examples 文件：包含实际测试程序源码和主要功能实现代码的示例文件。

详细配置流程如下所示：

1. 在 Kconfig 配置文件中，添加一个新的配置项，例如 **AIC_TP_DRV_TEST**，用于启用或禁用触摸面板驱动程序测试功能：

```
config AIC_TP_DRV_TEST
    bool "Enable touch panel driver test command"
    default n
    depends on AIC_TOUCH_PANEL_GT911
```

   - **AIC_TP_DRV_TEST**：新的配置项名称
   - **bool "Enable touch panel driver test command"**：用于启用或禁用触摸面板驱动程序测试功能。
   - **default n**：**AIC_TP_DRV_TEST** 设为默认关闭。
   - **depends on AIC_TOUCH_PANEL_GT911**：配置项依赖于AIC_TOUCH_PANEL_GT911。

2. 在 SConscript 文件中导入必要的模块 AIC_ROOT 和 PRJ_KERNEL。

   SConscript 文件决定参与编译的文件、目录等相关信息，具体语法请参考 SConstruct。

   本例程 SConscript 源码示例如下：

```
Import('AIC_ROOT')
Import('PRJ_KERNEL')
from building import *

cwd = GetCurrentDir()
CPPPATH = []
src = []
if GetDepend('AIC_TP_DRV_TEST'):
    src = Glob('*.c')

group = DefineGroup('test-touch', src, depend = [''], CPPPATH = CPPPATH)

Return('group')
```

   - Import('AIC_ROOT') 和 Import('PRJ_KERNEL')：导入必要模块。
   - 根据需要设置编译选项，例如 CPPPATH 等。
   - 使用 Glob('*.c') 获取所有源代码文件，并将其添加到一个名为 test-touch 组中。
   - 返回 test-touch 组，以便将其加入到构建过程中。

3. 使用宏定义将应用程序添加到 $(SDK)/bsp/test/test-touch/gt911_sample.c 的初始化函数列表中，确保系统启动时能被调用：

```
...
static void gt911_sample(void *parameter)
{
...
}

MSH_CMD_EXPORT(gt911_sample, gt911 sample);
```

   使用 RT-Thread 宏定义导入应用程序时，推荐使用 INIT_APP_EXPORT(fn) 和 MSH_CMD_EXPORT(fn) 宏定义。关于宏定义的详细说明，可查看宏接口优先级及描述。

   值得关注的是文件最后的导出命令 MSH_CMD_EXPORT，在系统启动之后，通过命令 gt911_sample 运行。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.10.2. 编译

1. 执行 scons --menuconfig 或 me（OneStep 命令），进入配置界面，按照以下步骤操作：

```
Drivers options  --->
    Drivers examples  --->
        [*] Enable touch panel driver test command
```

2. 保存配置并退出。

3. 运行 scons 命令或 m（OneStep 命令）开始编译。

   编译结束后会生成镜像文件。将生成的镜像文件烧录到目标设备上。

2.10.3. 验证

烧录镜像并运行系统后，使用 **Tab** 键查看应用命令是否添加成功。

1. 烧录镜像并运行系统，系统输出示例如下：

```
Startup reason: Power-On-Reset
Startup time: 0.601 sec (from Power-On-Reset)
os  : system memory alloc 320 bytes
flsh: UFFS consume spare data size 36
os  : system memory alloc 84480 bytes
os  : system memory alloc 53200 bytes
os  : system memory alloc 7168 bytes
tree: DIR 1, FILE 1, DATA 22
info.bits_per_pixel: 32
info.width: 1024, info.height: 600
lv_draw_aic_ctx_init:1024, 600
id = GT911
range_x = 1024
range_y = 600
point_num = 1
```

2. 使用 **Tab** 键查看应用命令是否添加成功：

```
<tab>                   # 单击 Tab 键
```

3. 系统罗列出可用命令列表。

   如果命令列表中出现 `test_gt911`，说明添加成功，输出示例如下

```
RT-Thread shell commands:
...
test_gt911     - test gt911 sample
...
```

2.10.4. 宏接口优先级及描述

RT–Thread 有一套启动优先级设置的宏，可根据不同的软件模块分类设置。具体接口及描述如下：

表 2–8 宏接口和描述

宏接口	描述
INIT_BOARD_EXPORT(fn)	非常早期的初始化，此时调度器还未启动; 使用该宏后，fn 将属于 “board init functions”
INIT_PREV_EXPORT(fn)	主要是用于纯软件的初始化、没有太多依赖的函数；使用该宏后，fn 将属于 “pre–initialization functions”
INIT_DEVICE_EXPORT(fn)	外设驱动初始化相关，比如网卡设备；使用该宏后，fn 将属于 “device init functions”
INIT_COMPONENT_EXPORT(fn)	组件初始化，比如文件系统或者 LWIP；使用该宏后，fn 将属于 “components init functions”
INIT_ENV_EXPORT(fn)	系统环境初始化，比如挂载文件系统；使用该宏后，fn 将属于 “enviroment init functions”
INIT_APP_EXPORT(fn)	应用初始化，比如 GUI 应用使用该宏后，fn 将属于 “application init functions”
MSH_CMD_EXPORT(fn)	将应用程序导出为 Msh 命令；通过手动方式运行；

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

2.11. hellowold

在 Baremetal 中有以下两种方案添加开机运行的 hellowold：

- 修改 main 函数
- 自动初始化

2.11.1. 修改 `main` 函数

SDK 默认创建了 `hellowold` 例程，可以直接在例程中添加私有函数逻辑：

- `hellowold` 例程代码为 `application/rt-thread/helloworld/main.c`。
- **`main`** 函数是 RT-Thread 的主入口函数。
- 例程代码目录的 `SConscript` 脚本默认会编译该目录中的所有 `c` 文件，因此可以直接添加用户代码文件，然后在 **`main`** 函数中调用新添加的文件中的函数。

```
application/rt-thread/helloworld/main.c |  3 +++
application/rt-thread/helloworld/test.c | 15 +++++++++++++++
2 files changed, 18 insertions(+)
create mode 100644 application/rt-thread/helloworld/test.c

diff --git a/application/rt-thread/helloworld/main.c b/application/rt-thread/helloworld/main.c
index 0dadb95e2..003233588 100644
--- a/application/rt-thread/helloworld/main.c
+++ b/application/rt-thread/helloworld/main.c
@@ -20,6 +20,8 @@
#include <boot_param.h>
#endif

+extern void helloworld();
+
int main(void)
{
#ifdef AIC_AB_SYSTEM_INTERFACE
@@ -48,5 +50,6 @@ int main(void)
#ifdef ULOG_USING_FILTER
    ulog_global_filter_lvl_set(ULOG_OUTPUT_LVL);
#endif
+    helloworld();
    return 0;
}
diff --git a/application/rt-thread/helloworld/test.c b/application/rt-thread/helloworld/test.c
new file mode 100644
index 000000000..634da82e6
--- /dev/null
+++ b/application/rt-thread/helloworld/test.c
@@ -0,0 +1,15 @@
+#include <rtthread.h>
+#include <stdio.h>
+
+void helloworld()
+{
+    printf("Hello World!\n");
+}
--
2.29.0
```

2.11.2. 自动初始化

RT-Thread 的自动初始化机制是指初始化函数不需要被显式调用，只需要在函数定义处通过宏定义的方式进行申明，则该函数在系统启动过程中将被顺序执行：

- 代码可以在任意地方声明和编译
- 通过 `INIT_APP_EXPORT` 申明

```
diff --git a/application/rt-thread/helloworld/test.c b/application/rt-thread/helloworld/test.c
new file mode 100644
index 000000000..18eb0821f
--- /dev/null
+++ b/application/rt-thread/helloworld/test.c
@@ -0,0 +1,10 @@
+#include <rtthread.h>
+#include <stdio.h>
+
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
+int helloworld(void)
+{
+    printf("Hello World!\n");
+    return 0;
+}
+
+INIT_APP_EXPORT(helloworld);
```

2.12. 烤机测试

烤机测试是一种长时间的老化测试，用于确保硬件在持续运行的情况下的稳定性和可靠性。 烤机测试具有以下特性：

- 不需要重新烧写镜像，省去量产环境的镜像文件管理麻烦。
- 只需准备一个存有测试数据的 SD 卡、或者 U 盘，即插即测。

目前已支持的测试项：

- 视频文件的循环播放

2.12.1. 源码结构说明

烤机测试的源码位于 `packages/artinchip/burn-in` 目录，结构如下：

```
packages
    |--artinchip
        |-burn-in
        |   |-burn_in.h
        |   |-burn_in.c         //主程序
        |   |-burn_in_player.c  //视频文件测试程序
        |   |-burn_in_xxx.c     //后续增加的烤机测试程序
```

`burn_in.c` 负责启动已经配置好的测试项。如需增加新的测试项，需要将其添加到 `auto_burn_in_test` 函数中。

```
static void auto_burn_in_test(void *arg)
{
    // wait some time for resource
    usleep(5*1000*1000);
#ifdef LPKG_BURN_IN_PLAYER_ENABLE
    burn_in_player_test(NULL);
#endif
}

int auto_burn_in(void)
{
    aicos_thread_t thid = NULL;
    thid = aicos_thread_create("auto_burn_in_test", 8192, 2, auto_burn_in_test, NULL);
    if (thid == NULL) {
        BURN_PRINT_ERR("Failed to create thread\n");
        return -1;
    }
    return 0;
}

INIT_APP_EXPORT(auto_burn_in);
```

2.12.2. 视频文件的循环播放

1. **`burn_in_player` 配置**

   a. 在 Baremetal 根目录下执行 scons --menuconfig 或 me 命令，打开配置菜单。在配置菜单中，依次选择以下选项，打开 player。

   ```
   local packages options--->
       ArtInChip packages options--->
           aic-mpp--->
               [*]   Enable player interface and demo
   ```

   b. 在 Baremetal 根目录下执行 scons --menuconfig 或 me命令。在配置菜单中，依次选择以下选项配置循环次数和打印输出：

   - 默认循环次数：10000000
   - log 输出目录：测试文件
   - log 名字：以测试时间命名，例如 yyyy-mm-dd-hh-mm-ss.log

Copyright © 2023-2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
local packages options--->
    ArtInChip packages options--->
        burn_in test--->
            [*]   test player
                [*]   set play file/dir loop num
                [*]   output log to serial interface
```

2. **准备视频文件**

   a. 需要准备一个 U 盘、或者 SD 卡

   b. 在 U 盘或 SD 卡的根目录中创建一个`aic_test`文件夹。

   c. 在`aic_test` 文件夹中创建一个`video` 子文件夹。

   d. 将需要测试的视频文件拷贝进 `video` 子文件夹中。

   > **注：**
   > 确保视频文件格式支持播放。

3. **执行测试**

   a. 将准备好的 U 盘或者 SD 卡插到板子上。

   b. 重启板子，确保板子能够识别到 U 盘或 SD 卡。

   c. 等待板子自动进入烤机测试，并开始持续的循环执行包括播放视频的所有测试项。

   > **注：**
   > 观察串口输出或日志文件，查看测试结果。

   根据测试结果进行相应的处理和分析。

2.12.3. 增加其他测试项

由于不同测试项的测试内容不同，后续如需添加新的测试项，需要自行添加测试步骤。参考 burn_in_player.c 文件，可了解已有测试项的实现方式和调用方式。

> **提示：**
> 对于每个测试项调用的测试命令，确保有返回值或状态码来表示测试结果，以便后续判断测试是否成功。比如 burn_in_player 调用了 player_demo 命令，player_demo 要能返回测试的结果。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3. 启动引导

> *启动引导程序 Bootloader 可以实现加载启动应用程序、烧录和升级功能。*

Baremetal 启动流程如下所示：

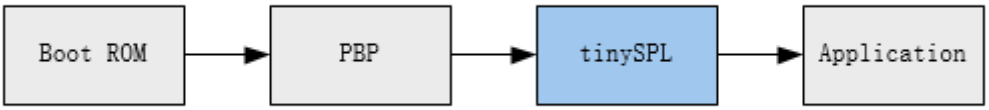


图 3-1 启动流程

- Boot ROM (BROM)
- PreBootProgram (PBP)：初始化 DRAM 。
- tinySPL：实现启动和升级。关于 tinySPL 的详细说明，查看 图 3-2 : 软件架构。

3.1. Boot 配置

使用以下命令可以进入 menuconfig 配置界面，配置 Boot 的相关参数。

```
scons --menuconfig
```

3.1.1. 配置 Bootloader 链接脚本

1. 在 **Chip options** 中，找到 **Custom link script path (relative path in sdk)** 选项。
2. 将链接脚本的相对路径填写到该选项中，以 `d21x_bootloader_gcc.ld` 链接脚本为例。

```
Chip options  --->
    (application/baremetal/bootloader/ldscript/d21x_bootloader_gcc.ld) Custom link script path (relative path in sdk)
```

3. 保存并退出 menuconfig 界面。
4. 重新编译项目，生成新的 Bootloader 固件。

3.1.2. 配置串口控制台使用 UART

1. 在 **Board Options** 中，使能本项目所使用的 UART，以 **`uart0`** 为例。

```
Board options  --->
        *** Peripheral Devices: ***
        [*] Using uart0
        [ ] Using uart1
        [ ] Using uart2
        [ ] Using uart3
```

2. 在 **Bootloader Options** 中设置控制台所使用的 UART ID。

```
Bootloader options  --->
    (0) Bootloader console UART ID
```

3. 保存并退出 menuconfig 界面。
4. 重新编译项目，生成新的 Bootloader 固件。

3.1.3. 配置 Bootloader 支持的启动介质

1. 启用 SPI NOR 和 SPI NAND 启动支持：

   在 **Bootloader options** 中，确保以下选项被选中：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
Bootloader options  --->
    [ ] MMC boot support  ----
    [*] SPI NOR boot support  --->
    [*] SPI NAND boot support  --->
```

2. 配置启动介质的相关参数。如不需要，可略过。

   部分启动介质需要进一步配置相关参数的，比如对于 SPI NAND，需要配置下列参数：

   - **SPI NAND device using QSPI controller**：指定了使用哪个 QSPI 控制器来访问 SPI NAND 设备。
   - **SPI NAND device working frequency (Hz)**：设置 QSPI 总线工作频率，例如 100000000。

   示例如下：

```
--- SPI NAND boot support
    (0)   SPI NAND device using QSPI controller
    (100000000) SPI NAND device working frequency(Hz)
```

3.1.4. 升级功能

在 **Bootloader options** 中，选中以下选项，分别启用 USB、NAND 和 NOR 升级功能：

```
Bootloader options  --->
    aicupg setting  --->
        [*] aicupg usb upgrade on
        [*] aicupg nand upgrade on
        [*] aicupg nor upgrade on
```

3.1.5. 编译 Boot 固件

新配置一个板子后，需要为该方案编译生成一个 `bootloader.bin` 文件。

在 Baremetal SDK 中，Boot 程序可以当作一个独立的裸机项目进行编译，与 SDK 中的其他方案编译方法相同。

例如：针对 D21x Demo88 的 SPI NAND 方案编译方法如下：

```
scons --apply-def d21x_demo88-nand_baremetal_bootloader_defconfig

scons
```

编译完成时，除了在 `output/d21x_demo88-nand_baremetal_bootloader/images/` 下生成 elf 文件外， 还会将 `bootloader.bin` 复制到板级目录之中，供打包烧录镜像时使用:

```
target/d21x/demo88-nand/pack/bootloader.bin
```

编译 Boot 时，不会打包烧录镜像文件，编译应用时才会打包烧录镜像。

> **提示：**
>
> 在编译应用时，如果 `bootloader.bin` 已经存在，并且分区等配置没有改变，则不需要重新编译 Bootloader。

3.2. 启用调试

遵照以下流程，启用 Bootloader 调试功能：

1. 执行下列命令，进入 menuconfig 配置界面。

```
scons --menuconfig
```

2. 在 **Bootloader Options** 中，选择以下选项，启用 Bootloader 调试功能：

```
Bootloader options  --->
    bootloader debug  --->
        [ ] aicupg debug on
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3.3. 测试指南

通过命令行可以执行 Boot 测试。使用以下任意方式，可以进入 Boot 的命令行模式：

- 启动过程中一直按住键盘上的 **CTRL+C** 键。
- 启动加载应用程序失败。
- 升级过程中按 **CTRL+C** 终止升级流程，退出到命令行模式。

3.3.1. 命令编译

在使用命令之前，首先需要确保在 、**Bootloader options** 中已经选中命令进行编译。

```
Bootloader options  --->
    Bootloader commands  --->
        [*] nor boot
        [ ] xip
        [*] nand boot
        [ ] mmc boot
        [*] spinor
        [*] spinand
        [*] mtd read/write
        [*] mem
```

3.3.2. 命令帮助

在命令行模式下，输入 help 可以获取当前支持的命令列表。

对于具体命令，通常输入 cmd help 可获取关于该命令的使用指南，例如 mtd help。

3.4. 设计说明

3.4.1. 源码说明

表 3–1 tinySPL 源码结构

Sys	bsp/artinchip/sys/<chip>/	SoC 初始化相关代码，与应用共用
Board	target/<chip>/<board>/	板子初始化相关代码，与应用共用
HAL	bsp/artinchip/hal/	HAL 驱动代码，与应用共用
Boot	application/baremetal/bootloader/	Bootloader 代码

3.4.2. 软件架构

Baremetal 使用 tinySPL 作为启动引导程序 (Bootloader)，可以实现加载启动应用程序、烧录和升级等功能。

tinySPL 是基于 Baremetal 的 HAL 构建的裸机 (Baremetal) 程序。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

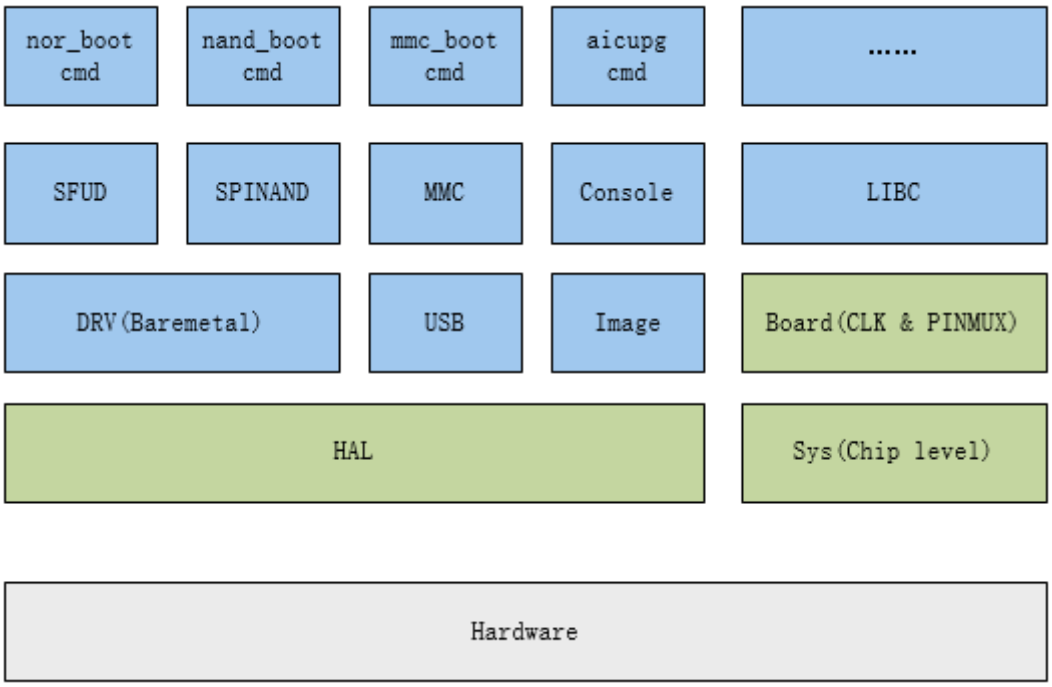


图 3–2 软件架构

上图中， `HAL` 、 `Board` 、 `Sys` 与 Baremetal SDK 中的应用程序共用同一份代码， 以简化 Boot 对不同板子的支持。

由于 Boot 的功能和应用场景比较确定和单一，因此这里将 Boot 设计为一个 Baremetal 版本的应用，具有以下特点：

1. 与应用共用部分代码，简化开发和板级配置
2. 无线程、无中断处理，只针对单一任务
3. 支持命令行模式，可根据需要，定义不同的命令

3.4.3. 关键流程

- **系统初始化**

```
_start //bsp/artinchip/sys/d21x/startup_gcc.S
|-> save_boot_params // bsp/artinchip/sys/d21x/boot_param_gcc.S
|-> icache_enable();
|-> dcache_enable();  // bsp/artinchip/sys/d21x/system.c
|-> SystemInit();
|-> main(); // application/baremetal/bootloader/main.c
    |-> board_init();
    |   |-> heap_init();
    |   |-> aic_board_sysclk_init(); // target/d21x/<board>/board.c
    |   |-> aic_board_pinmux_init();
    |   |-> uart_init(cons_uart);
    |   |-> stdio_set_uart(cons_uart);
    |-> console_init();
```

系统初始化的过程中，并不会设计不必要的驱动等模块的初始化。所有的驱动和模块， 都只在需要使用时进行初始化，这样可以减少启动过程中的不必要时间消耗，加快启动速度。

- **启动和升级**

```
main(); // application/baremetal/bootloader/main.c
|-> board_init();
|-> console_init();
|-> bd = aic_get_boot_device();
|   // 根据启动参数，判断当前的启动设备
|
|-> console_set_bootcmd("nor_boot");
|   // 此处根据 boot_device 的类型，设置不同的 bootcmd，比如 aicupg usb 0
|
|-> console_loop();
    |-> console_run_cmd(g_console->bootcmd);
    |-> _console_loop(g_console);
        // 当执行 bootcmd 失败时，会进入串口控制台，等待用户交互
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

3.5. 常见问题

ArtInChip

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

4. 命令行工具参考指南

本节详细介绍了 ArtInChip SDK 编译过程中涉及的命令行工具使用说明，仅供参考。

4.1. menuconfig 命令工具参考指南

menuconfig 是一个图形化的配置工具，主要用于配置 Linux 内核和开源项目（如 BusyBox）的编译选项。

4.1.1. 启动 Menuconfig GUI 配置界面

1. 在 SDK 根目录下，执行下列命令进入 menuconfig 的主配置界面：

   - 常规命令

     关于 make 命令的详细说明，可查看

   - OneStep 简洁命令

     ```
     me
     ```

     关于 OneStep 简洁命令的详细描述，可查看 OneStep 命令参考指南。

2. 在 SDK 根目录下，使用下列命令可以进入 menuconfig 对应的子配置选项界面，示例如下：

   - 进入内核配置界面：

     ```
     make kernel-menuconfig
     ```

     或使用下列命令：

     ```
     make linux-menuconfig
     ```

   - 进入 U-Boot 配置界面：

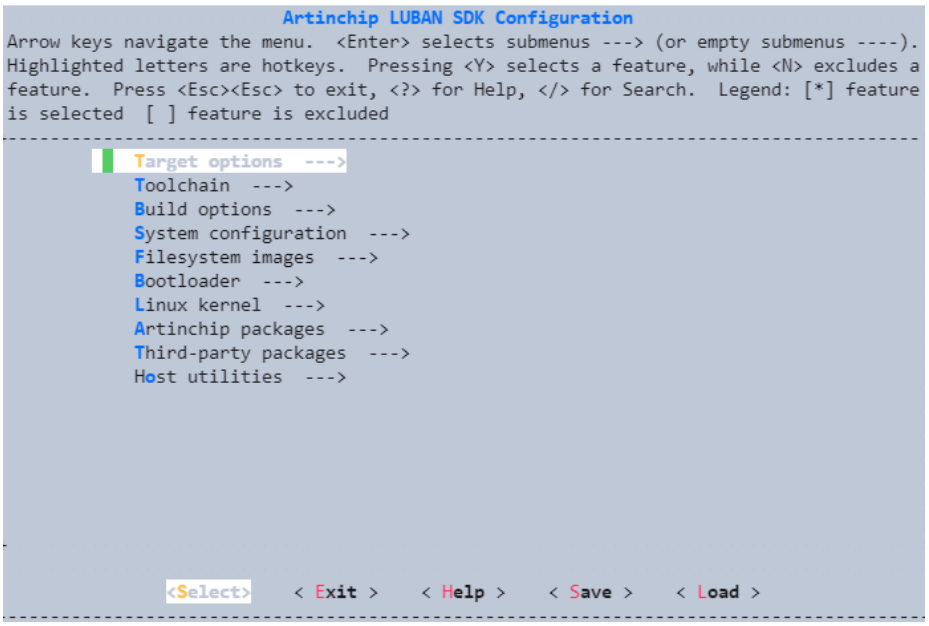
     ```
     make uboot-menuconfig
     ```

   - 进入开源包 Busybox 的配置界面：

     ```
     make busybox-menuconfig
     ```

4.1.2. Menuconfig GUI 配置选项介绍

启动 menuconfig 配置工具后，会进入一个交互式的配置界面 (GUI)。用户可以按照需要选择和调整各项参数，示例如下：



Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

4.2. MSH 命令行工具参考指南

在 Linux 系统中，MSH（Micro Shell）是一种轻量级的 shell 环境。MSH 命令提供了一套供用户在命令行调用的操作接口，主要用于调试或查看系统信息。以下是 ArtInChip SDK 编译过程中常见的 MSH 终端命令及其说明。

4.2.1. pin

在 MSH 终端使用 pin 命令，设置引脚相关功能，用于调试功能。

语法

```
pin <options>
```

参数

<num>

从硬件中获取 GPIO ID 编号。

可选编号包括 PE02, PE2, PE.02, PE.2, pe02, pe2, pe.02, pe.2。

```
pin num PA.16
```

<mode>

设置 GPIO 工作模式。

可选模式包括 output/ input/ input_pullup/ input_pulldown/ output_od

```
pin mode PA.16 output
```

<read>

读取 GPIO 引脚电平状态

```
pin read PA.16
```

<write>

写入 GPIO 引脚电平状态

可配置为 high/ low 或 on/ off

```
pin write PA.16 high
```

<help>

获取命令帮助列表

4.2.2. 控制 GPIO

- 读寄存器。

  p 调试命令，用于打印 (print) 某个变量、寄存器或内存地址的值。对于没有封装专有调试命令的功能，可使用 MSH 命令行下的 p 命令读写寄存器。

  ### 语法

  ```
  p <parameter_name>
  ```

  示例：读取 0x18700000 内存地址的值

  ```
  p 0x18700000
  ```

  输出示例如下：

  ```
  18700000: 00000100 00000100 00000200 00000101
  18700010: 00000000 00000000 00000000 00000000
  18700020: 00000000 00000000 00000000 00000000
  18700030: 00000000 00000000 00000000 00000000
  ```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- 写寄存器。

  m 调试命令，用于修改或写入内存中的某个地址的值。对于没有封装专有调试命令的功能，可使用 MSH 命令行下的 m 命令读写寄存器。

  语法

  ```
  m <parameter_name> <new_value>
  ```

  示例：将地址 `0x18700004` 处的值设置为 `0x101`

  ```
  m 0x18700000 0x101
  ```

  输出示例如下：

  ```
  value on 18700004 change from 0x100 to 0x101.
  ```

4.2.3. 自定义命令

用户可以自定义 MSH 命令，通过宏接口 `MSH_CMD_EXPORT(name, desc)` 导出函数作为命令在 MSH 模式下执行。例如，可以定义一个打印 “Hello RT-Thread!” 的命令 hello，并通过以下方式导出：

```
void hello(void) {
    rt_kprintf("Hello RT-Thread!");
}
MSH_CMD_EXPORT(hello, say hello to RT-Thread);
```

4.3. DFS 命令行工具参考指南

在 RT-Thread 操作系统中，DFS (Device File System) 提供了一组基本的文件操作命令。以下是 ArtInChip SDK 编译过程中常见的 DFS 终端命令及其说明。

4.3.1. cd

改变当前工作目录。

```
cd <filepath>
```

示例：进入 `data` 目录

```
cd data
```

4.3.2. ls

列出目录中的文件和子目录。

```
ls
```

比如，列出 `data` 目录中的文件和子目录，输出示例如下：

```
Directory /data:
fb0.argb            1536000
```

4.3.3. cp

复制文件或目录。

```
cp <orginal_filepath> <new_filepath>
```

示例：将脚本放入新建目录中：

```
cp package/artinchip/lvgl-ui/S00lvgl target/chip/board/rootfs_overlay/etc/init.d/S00lvgl
```

示例输出：

```
Copy data 1536000 B, speed 2.29 MB/s
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

4.3.4. mv

移动或重命名文件或目录。

```
mv <original_filename> <new_filename>
```

示例：重命名文件

```
mv fb0.argb <fb1>.argb
```

4.3.5. rm

删除文件或目录。

```
rm fb0.argb
```

4.3.6. mkdir

创建目录。

```
mkdir -p <new_dir>
```

示例：在 `target/<chip>/<board>/rootfs_overlay/` 目录下创建一个新目录存放脚本文件，例如 `/etc/init.d/`

```
mkdir -p target/chip/board/rootfs_overlay/etc/init.d
```

4.3.7. rmdir

删除空目录。

```
rmdir new_dir
```

4.3.8. cat

显示文件内容。

例如，使用 `cat` 命令读取 `otg_mode` 文件的内容，以查看当前的 USB OTG 模式。

```
cat /sys/devices/platform/soc/soc:usb-otg/otg_mode
```

系统输出示例如下：

```
auto
```

4.3.9. echo

向文件写入数据。

例如，将 `auto` 写入 `otg_mode` 文件，使系统根据连接的设备自动切换角色。

```
echo auto > /sys/devices/platform/soc/soc:usb-otg/otg_mode
```

4.3.10. chmod +x

更改文件权限。

示例

使用 chmod +x 命令将脚本设置为可执行文件：

```
chmod +x target/chip/board/rootfs_overlay/etc/init.d/S00lvgl
```

4.3.11. mount

```
mount -t dfs elm,data=/dev/data,noatime=y /mnt/dfs
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

4.4. Shell 命令行工具参考指南

RT-Thread Shell 命令行是 RT-Thread 操作系统中的一个重要功能，允许用户通过命令行与系统进行交互，执行各种操作和调试任务。在 RT-Thread Shell 中，用户可以输入各种命令来查看系统状态、管理线程和设备、控制 GPIO 等。

4.4.1. date

查看时区配置。

```
date
```

输出示例如下：

```
Tue Jan  601:41:27 UTC 1970
```

4.4.2. echo

```
echo 9 > brightness
```

4.4.3. free

显示系统中的内存使用情况。

4.4.4. tsen_status

4.5. hwclock 命令参考指南

Busybox 源码自带一个 `hwclock` 工具，用来读取和设置 RTC 时间。用法如下：

- 读取当前 RTC 时间（不加任何参数时就默认是读取）：

  ```
  hwclock -r
  ```

  输出示例如下：

  ```
  Thu Jan  100:00:00  19700.000000 seconds
  ```

- 读取当前 RTC 时间，并加上时区校准：

  ```
  hwclock -ru
  ```

- 将当前的系统时间同步设置到 RTC：

  ```
  hwclock -w
  ```

- 将当前的系统时间减去时区值，并同步设置到 RTC：

  ```
  hwclock -wu
  ```

- 将 RTC 时间同步到系统时间：

  ```
  hwclock -s
  ```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

5. 命令详解

本节介绍 Baremetal SDK 编译过程中涉及的命令类型以及各个命令的详细说明。

5.1. OneStep 命令参考指南

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。根据操作系统的不同，可能需要不同的设置：

- **Windows**系统设置

  在 Windows 系统中，OneStep 自动集成到 `win_cmd.bat` 和 `win_env.bat` 批处理脚本中。

  在 CMD 或者 ENV 窗口启动后，OneStep 命令已经生效，在其中可以从任意目录执行 OneStep 命令。

- **Linux** 系统设置

  在 Linux 系统中，需要将 OneStep 脚本添加到当前路径中，如执行以下命令：

  ```
  source tools/onestep.sh
  ```

  在 Ubuntu 终端中，进入 SDK 根目录后，使用 source tools/onestep.sh 命令即可查看所有常见命令。

为了方便使用并加快开发效率，ArtInChip 开发了一系列的 OneStep 子命令，如h|help显示帮助信息以及lunch选择和启动指定的项目配置。

```
h
```

```
Baremetal SDK OneStep commands:
    hmm|h                   : Get this help.
    lunch          [keyword]  : Start with selected defconfig.e.g. lunch mmc
    menuconfig|me           : Config application with menuconfig
    bm                      : Config bootloader with menuconfig
    km                      : Config application with menuconfig
    m|mb                    : Build bootloader & application and generate final image
    ma                      : Build application only
    mu|ms                   : Build bootloader only
    c                       : Clean bootloader and application
    mc                      : Clean & Rebuild all and generate final image
    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.
    godir|gd       [keyword]  : Go/jump to selected directory.
    list                    : List all SDK defconfig.
    list_module             : List all enabled modules.
    i                       : Get current project's information.
    buildall                : Build all the *defconfig in target/configs
    rebuildall              : Clean and build all the *defconfig in target/configs
    addboard|ab             : Add new board *defconfig in target/configs
    aicupg                  : Burn image file to target board
```

5.1.1. list

查看所有项目文件。

> **注：**
>
> list 命令进行了显示项目精简，隐藏了 `bootloader` 的项目文件。

```
list
```

输出示例如下：

```
scons: Reading SConscript files ...
Built-in configs:
d12x_demo68-nand_rt-thread_helloworld_defconfig
0. d12x_demo68-nor_rt-thread_helloworld_defconfig
1. d12x_hmi-nor_rt-thread_helloworld_defconfig
2. d13x_demo68-nor_rt-thread_helloworld_defconfig
3. d13x_demo88-nand_rt-thread_helloworld_defconfig
4. d13x_demo88-nor_rt-thread_helloworld_defconfig
5. d13x_kunlunpi88-nor_rt-thread_helloworld_defconfig
6. d21x_d215-demo88-nand_rt-thread_helloworld_defconfig
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
7. d21x_d215-demo88-nor_rt-thread_helloworld_defconfig
8. d21x_demo100-nor_rt-thread_helloworld_defconfig
9. d21x_demo128-nand_rt-thread_helloworld_defconfig
10.g73x_demo100-nor_rt-thread_helloworld_defconfig
11.g73x_demo68-nor_rt-thread_helloworld_defconfig
12.g73x_scan_rt-thread_helloworld_defconfig
```

5.1.2. lunch

选择、加载或应用具体配置方案。

- 示例 1：进入 Recovery 系统 SDK 生产环境

```
lunch ota_emmc
```

- 示例 2：加载 `d13x_demo88-nor-xip_rt-thread_helloworld_defconfig` 配置文件

```
lunch d13x_demo88-nor-xip_rt-thread_helloworld_defconfig
```

5.1.3. menuconfig|me

打开 menuconfig 工具界面，修改 RT-Thread 配置

```
menuconfig
```

或

```
me
```

5.1.4. bm

打开 menuconfig 工具界面，修改 BootLoader 配置

```
bm
```

5.1.5. km

打开 menuconfig 工具界面，修改应用配置

```
bm
```

5.1.6. m

编译 SDK 固件，并生成镜像文件。

```
m
```

> **注：**
> 与 mb 命令含义相同。

5.1.7. mb

编译 bootloader 和应用，并生成最终镜像文件。

```
mb
```

5.1.8. ma

仅编译应用，并生成最终镜像文件。

```
ma
```

5.1.9. mu|ms

仅编译 bootloader，并生成最终镜像文件。

```
mu|ms
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

5.1.10. c

清除 bootloader 和应用配置。

```
c
```

5.1.11. mc

清除并重新编译，重新生产所有最终镜像文件。

```
mc
```

5.1.12. crootlcr

进入 SDK 根目录。

5.1.13. coutlco

进入编译输出目录

5.1.14. ctargetlct

回到方案或目标开发板目录。

5.1.15. godirlgd

跳转到指定目录。

5.1.16. i

查看当前项目的配置信息。

```
i
```

```
scons: Reading SConscript files ...
Target app: application/rt-thread/helloworld
Target chip: d12x
Target arch: riscv32
Target board: target/d12x/demo68-nor
Target kernel: kernel/rt-thread
Defconfig file: target/configs/d12x_demo68-nor_rt-thread_helloworld_defconfig
Root directory: xxxxxxxxxxxxx
Out directory: output/d12x_demo68-nor_rt-thread_helloworld
    Toolchain: toolchain/bin\riscv64-unknown-elf-
```

5.1.17. buildall

编译目标配置文件 `target/configs` 中的所有 `*defconfig` 文件。

5.1.18. rebuildall

清除前一次配置信息，并重新编译目标配置文件 `target/configs` 中的所有 `*defconfig` 文件。

5.1.19. addboardlab

在 `target/configs` 中添加新开发板的 `*defconfig` 文件。

5.1.20. aicupg

将镜像文件烧录到目标开发板中。

5.2. scons 命令参考指南

SCons 是一个跨平台的构建系统，被广泛应用于软件开发项目中，支持在 Windows 和 Linux 上使用，且用法相同。本节列示了常用的 SCons 命令和功能。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

5.2.1. scons --list-def

解压缩 SDK 后，使用 scons --list-def 命令可以罗列 SDK 中发布的所有项目名称，方便用户查看和管理可用的项目资源。示例如下：

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

输出示例结果如下：

```
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
```

5.2.2. scons --apply-def=<项目索引或名称>

使用命令 scons --apply-def=[项目索引或名称] 可以选择特定的项目配置进行构建，命令中可以使用项目名称或项目索引完成构建：

- 使用数字索引，例如 --apply-def=3，即应用列表中的第三个配置（从 0 开始计数）。以下示例表示应用名为 d12x_demo68-nor_rt-thread_helloworld_defconfig 的配置。

```
scons --apply-def=3
```

```
scons: Reading SConscript files ...
Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig

scons --apply-def=d12x_demo68-nor_rt-thread_helloworld_defconfig
scons: Reading SConscript files ...
Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig
```

- 直接使用配置名称，例如 --apply-def=d12x_demo68-nor_rt-thread_helloworld_defconfig。

```
scons --apply-def=d12x_demo68-nor_rt-thread_helloworld_defconfig
```

```
scons: Reading SConscript files ...
Load config from target\configs\d12x_demo68-nor_rt-thread_helloworld_defconfig
```

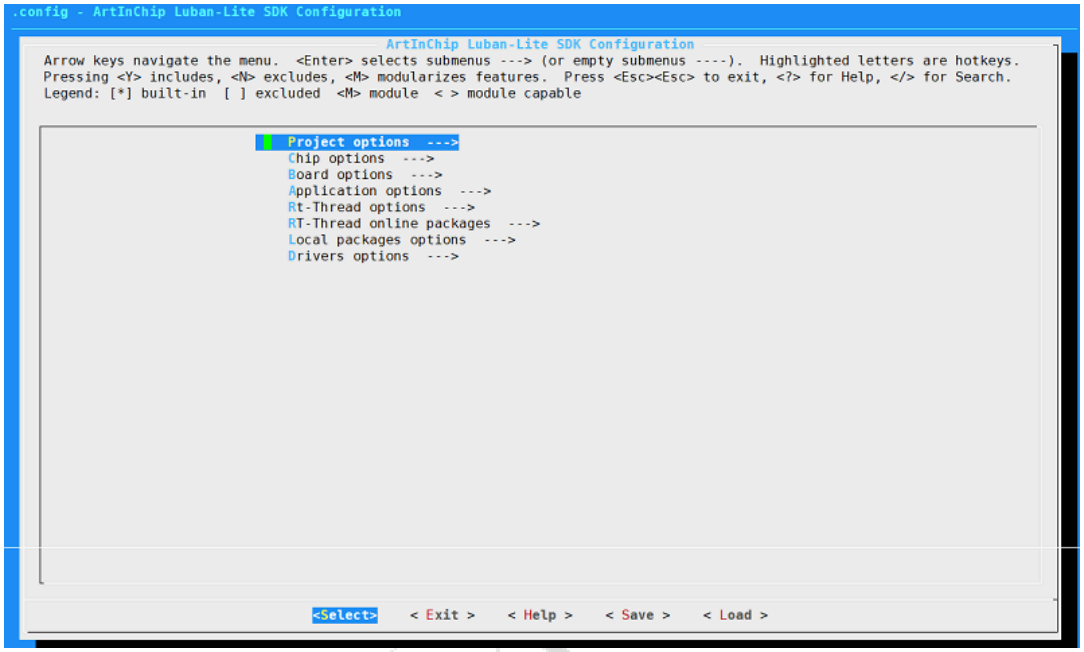
5.2.3. scons --menuconfig

使用命令 scons --menuconfig 可以调出 menuconfig 图形界面，以便根据需要调整项目设置。

```
scons --menuconfig
```

menuconfig GUI 界面示例如下：

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved



完成配置后，保存并退出配置界面，SCons 会根据所做的更改重新编译项目。

> **注：**
> 关于 menuconfig 的更多信息，可查看 menuconfig 命令工具参考指南。

5.2.4. scons

使用命令 scons 开始构建过程，编译成功后生成镜像文件。默认在简洁模式下进行构建，不会输出编译选项等信息。

```
scons
```

根据下列打印信息示例，编译成功后生成的文件路径为 `e:\luban-lite\output\d12x_demo68-nor_rt-thread_helloworld\images\d12x_demo68-nor_v1.0.0.img`：

```
Creating e:\luban-lite\output/d12x_demo68-nor_rt-thread_helloworld/images/usbupg-ddr-init.aic ...
Creating e:\luban-lite\output/d12x_demo68-nor_rt-thread_helloworld/images/bootloader.aic ...
Creating e:\luban-lite\output/d12x_demo68-nor_rt-thread_helloworld/images/os.aic ...
Image header is generated.
Meta data for image components:
            Meta for image.updater.psram       offset 0x1600     size 0x6010 (24592)
            Meta for image.updater.spl         offset 0x7e00     size 0x2d310 (185104)
            Meta for image.info                offset 0x0         size 0x800 (2048)
            Meta for image.target.spl          offset 0x35600    size 0x2d310 (185104)
            Meta for image.target.os           offset 0x62e00    size 0xd8800 (886784)
            Meta for image.target.rodata       offset 0x13b600   size 0x2b000 (176128)
            Meta for image.target.data         offset 0x166600   size 0x700000 (7340032)
    Packing file data:
            uartupg-psram-init.aic
            bootloader.aic
            bootloader.aic
            d12x_os.itb
            rodata.fatfs
            data.lfs
    Image file is generated: e:/luban-lite/output/d12x_demo68-nor_rt-thread_helloworld/images/d12x_demo68-nor_v1.0.0.img

Luban-Lite is built successfully

scons: done building targets.
```

5.2.5. scons --verbose

使用 scons --verbose 可以显示更详细的编译信息，包括编译器选项等，便于调试。

5.2.6. scons --clean

运行命令 scons --clean 清除上一次编译构建过程中生成的所有临时文件，确保在下一次编译构建时，所有源文件都会被重新编译。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
scons --clean
```

或者使用缩写命令：

```
scons -c
```

5.2.7. scons --save-def

保存当前板卡默认配置。

5.2.8. scons --info

显示当前工程信息，可使用简洁命令 scons i。

5.2.9. scons distclean

清除工具链和输出目录。

5.2.10. scons target=<*TARGET*>

生成目标工程，例如 `eclipse/eclipse_sdk`，其中 *<TARGET>* 是需要生成的目标工程名称。

5.2.11. scons genconfig

通过 `rtconfig.h` 生成 `.config` 文件。

5.2.12. scons useconfig=<*USECONFIG*>

通过配置文件生成 `rtconfig.h`，其中 *<USECONFIG>* 是需要使用的配置文件路径。

5.2.13. scons --run-qemu

运行当前编译出来的 qemu 目标文件。

5.2.14. scons --list-size

size 命令列出所有 `.o` 文件下 `text/data/bss` 中各个 section 的大小。

5.2.15. scons --pkgs-update

下载选择的在线 packages。

5.3. tree 命令参考指南

tree 命令是 Linux 系统中一个用于以树状图形式显示目录结构的命令：

- **显示当前目录结构**：直接输入`tree`命令，不跟任何参数，将显示当前目录下的文件和子目录结构。
- **指定目录显示**：通过在`tree`命令后加上路径参数，可以查看特定目录的结构。
- **递归显示**：默认情况下，`tree`命令会递归地显示目录下的所有文件和子目录。

常用选项

- -a：显示所有文件，包括隐藏文件。
- -d：只显示目录，不显示文件。
- -f：在每个文件或目录前显示完整的相对路径。
- -i：不使用缩进和线条，仅显示文件和目录名称。

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

- -l：按修改时间排序。
- -r：反转显示顺序。
- -s：显示文件大小。
- -L level：限制显示的层数。
- -P pattern：只显示符合模式的文件和目录。

查看编译完成后的目录结构

使用 SCons 命令编译 Baremetal SDK 完成后，生成的库文件位于 output 目录下，包含各个配置编译生成的目录， 例如编译后生成的 d21x_demo100-nand_rt-thread_helloworld 目录。

在 output 下，使用 tree -L 2 命令会显示当前目录及其子目录的层级结构，详情如下：

```
tree -L 2
```

```
.
├── d21x_demo100-nand_rt-thread_helloworld
│   ├── application
│   ├── bsp
│   ├── images
│   ├── kernel
│   ├── libs
│   ├── packages
│   └── target
```

```
tree -L 2
```

```
.
├── d21x_demo100-nand_rt-thread_helloworld
│   ├── application # 存放编译过程中， ``$SDK/application`` 源码目录生成的 ``*.o`` 文件;
│   ├── bsp         # 存放编译过程中， ``$SDK/bsp`` 源码目录生成的 ``*.o`` 文件，包括驱动模块、外设驱动等;
│   ├── images      # 编译生成的镜像文件、函数符号表等文件;
│   ├── kernel      # 存放 Luban-Lite 操作系统内核模块编译生成的 ``*.o`` 文件;
│   ├── libs        # ``application`` 目录里或用户自定义生成的库文件存放于此目录;
│   ├── packages    # 编译后 ``$SDK/packages`` 目录后，生成的 ``*.o`` 文件；主要包括 ``artinchip`` 和 ``third-party`` 两部分;
│   └── target      # 针对选择的板卡，生成的 ``board.o``、 ``pinmux.o`` 和 ``sys_clk.o`` ;
```

5.4. fw_ 命令参考指南

fw_setenv 和 fw_printenv 是用于在 Linux 系统下读取和修改 U-Boot 环境变量的命令工具，允许用户在应用层上对 U-Boot 的环境变量进行操作，而不必直接进入 U-Boot 命令行界面。以下是 ArtInChip SDK 编译过程中常见命令及其说明。

5.4.1. fw_setenv

示例：

- 将下次系统启动分区设置为 B。

```
fw_setenv osAB_next B
```

- 将下次只读文件系统挂载分区设置为 B。

```
fw_setenv rodataAB_next B
```

- 将下次读写文件系统挂载分区设置为 A。

```
fw_setenv dataAB_next A
```

5.4.2. fw_printenv

查看当前的环境变量设置。

输出示例如下：

```
MTD=spi0.0:1m(spl),256k(env),256k(env_r),4m(os),4m(os_r),12m(rodata),12m(rodata_r),40m(data),40m(data_r)
bootlimit=5
rodata_partname=blk_rodata
rodata_partname_r=blk_rodata_r
data_partname=blk_data
data_partname_r=blk_data_r
```


Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved

```
bootcount=1
upgrade_available=0
osAB_now=A # 当前系统启动分区为 A 分区
rodataAB_now=A # 当前只读系统挂载为 A 分区
dataAB_now=B # 当前读写系统挂载为 B 分区
osAB_next=B # 下次系统启动分区为 B 分区
rodataAB_next=B # 下次只读系统挂载为 B 分区
dataAB_next=A # 下次读写系统挂载为 A 分区
```

ArtInChip

Copyright © 2023–2025 ArtInChip Technology Co., Ltd.
All rights reserved