



CV180X & CV181X & CV184X & CV186AH/BM1688 量产烧写使用手册

Version: 1.0.8

Release date: 2025-03-19

©2025 北京晶视智能科技有限公司
本文件所含信息归北京晶视智能科技有限公司所有。
未经授权，严禁全部或部分复制或披露该等信息。

目录

| | | |
|----------|------------------------------|-----------|
| 1 | 声明 | 2 |
| 2 | 概述 | 3 |
| 2.1 | 介绍 | 3 |
| 2.2 | 烧录前的准备工作 | 3 |
| 3 | USB 烧录/UART 烧录 | 4 |
| 3.1 | 概述 | 4 |
| 3.2 | 安装 usb 驱动 (USB 烧录) | 4 |
| 3.3 | 卸载 usb 驱动 (USB 烧录) | 6 |
| 3.4 | cviDownloadTool 工具 | 7 |
| 3.5 | cviDownloadTool 工具 | 18 |
| 4 | SD 卡烧录 | 20 |
| 4.1 | 概述 | 20 |
| 4.2 | 使用 SD 卡裸烧 | 20 |
| 5 | TFTP 烧录 | 24 |
| 5.1 | 使用 TFTP 加载固件到 DDR | 24 |
| 5.2 | 烧写固件到 Flash | 26 |
| 6 | 烧录器烧录 | 30 |
| 6.1 | 概述 | 30 |
| 6.2 | 预烧程序 | 30 |

修订记录

| Revision | Date | Description |
|----------|------------|----------------------------|
| 1.0.0 | 2023/08/30 | 初稿 |
| 1.0.1 | 2023/09/14 | 添加 usb 驱动安装使用指南 |
| 1.0.2 | 2023/09/22 | 完善一拖多流程内容 |
| 1.0.3 | 2023/09/27 | 修改一拖多界面图片，增加注意事项 |
| 1.0.4 | 2023/10/20 | 增加 ini 配置文件说明 |
| 1.0.5 | 2023/12/10 | 修改一拖多图标与进度条显示说明 |
| 1.0.6 | 2024/03/29 | 更新驱动安装章节内容 |
| 1.0.7 | 2024/11/08 | 重构工具，更新新的设置选项和图片 |
| 1.0.8 | 2025/03/19 | 集成 uart、usb、sd、tftp、烧录器的文档 |

1 声明



法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

联系我们

地址 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

电话 +86-10-57590723 +86-10-57590724

邮编 100094（北京）518100（深圳）

官方网站 <https://www.sophgo.com/>

技术论坛 <https://developer.sophgo.com/forum/index.html>

2 概述

2.1 介绍

本文介绍通过 USB、UART、SD 卡、烧录器等方法对单板进行烧录。

2.2 烧录前的准备工作

- 烧录前的准备工作如下：
- USB/UART 烧录：
 - 准备待烧写的原始文件，可选的格式有：
 - * 文件系统镜像文件。
 - * 文件系统镜像文件的压缩 zip 文件。
 - * 文件系统镜像文件的 img/tar/tar.gz 格式压缩文件。（需要自备解包工具，解包工具必须为可直接在命令行执行的 exe 文件）
 - 准备 cviDownloadTool 工具。
 - 准备双 USB 接口数据线。
- SD 卡烧录：
 - 文件系统镜像文件。
 - SD 卡一张。
- TFTP 烧录：
 - 文件系统的 rawimage 镜像文件。
 - tftpd64。
- 烧录器烧录：
 - 文件系统的 rawimage 镜像文件。
 - 烧录器。

3 USB 烧录/UART 烧录

3.1 概述

3.1.1 介绍

本文介绍如何使用 cviDownloadTool 烧录整个单板系统文件，该方案通过 USB 通信来完成烧录，具有成本低，烧录速度快等特点。

3.1.2 烧录前的准备工作

- 烧录前的准备工作如下：
- 准备待烧写的原始文件，可选的格式有：
 - 文件系统镜像文件。
 - 文件系统镜像文件的压缩 zip 文件。
 - 文件系统镜像文件的 img/tar/tar.gz 格式压缩文件。（需要自备解包工具，解包工具必须为可直接在命令行执行的 exe 文件）
- 准备 cviDownloadTool 工具。
- 准备双 USB 接口数据线。

3.2 安装 usb 驱动 (USB 烧录)

以 windows 10 系统下驱动安装为例

步骤 1

双击运行 CviUsbDownloadInstallDriver.exe ，点选下一步直至安装完成

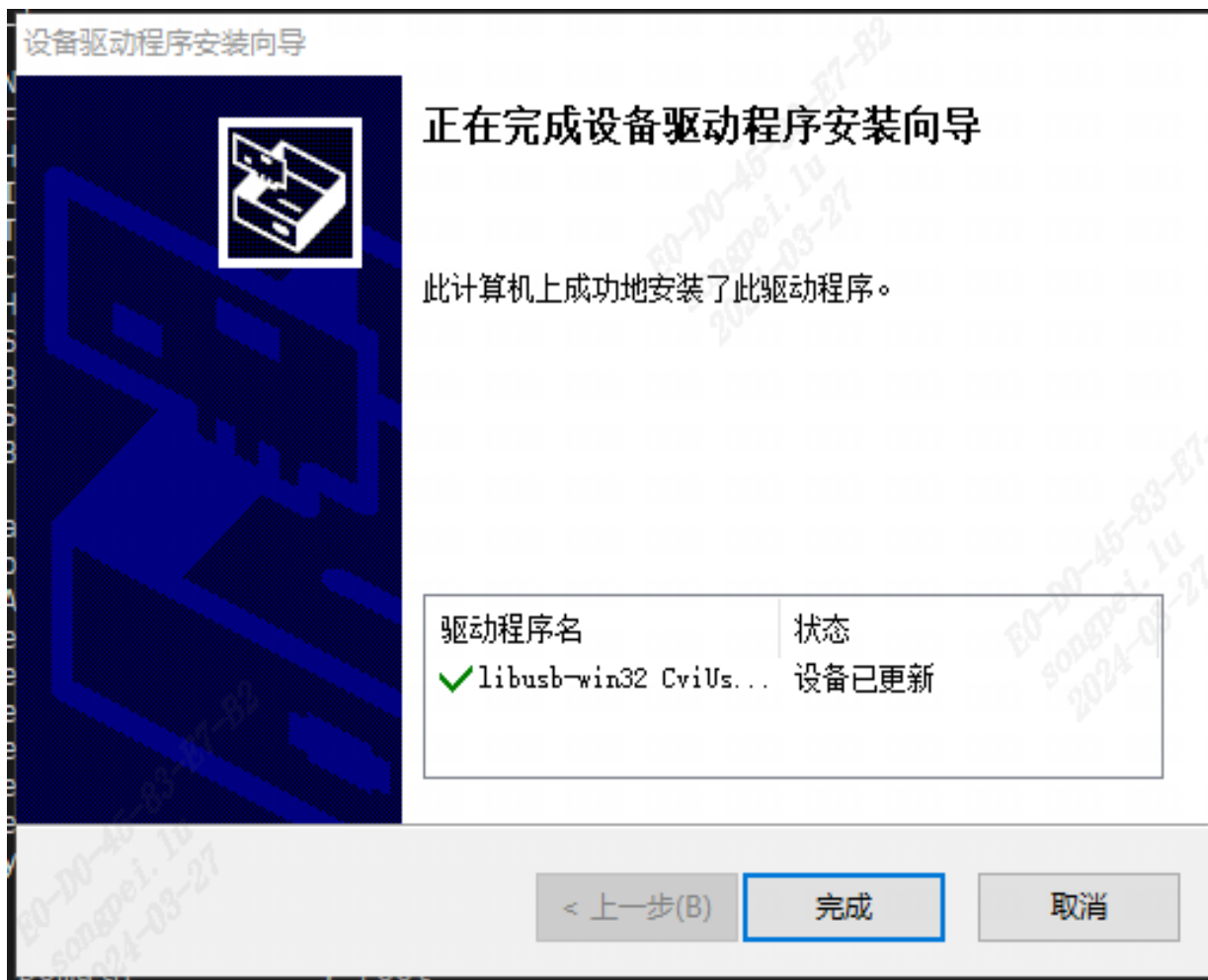


图 3.1: 执行 CviUsbDownloadInstallDriver.exe

驱动安装完成后，即可使用 CviDownload 工具来烧录固件，具体烧录步骤请参考《量产烧写使用指南》的“cviDownloadTool 工具”章节。

步骤 2

找一片烧过 u-boot 的板子，进入到 uboot 后敲 `cvi_utask vid 0x3346 pid 0x1000`，让 cv18xx 板子当作 device 接入电脑，在电脑上打开“设备管理器”，看到“libusb-win32 devices”下面出现一个“CviUsbDownload”的设备，表示安装正常。

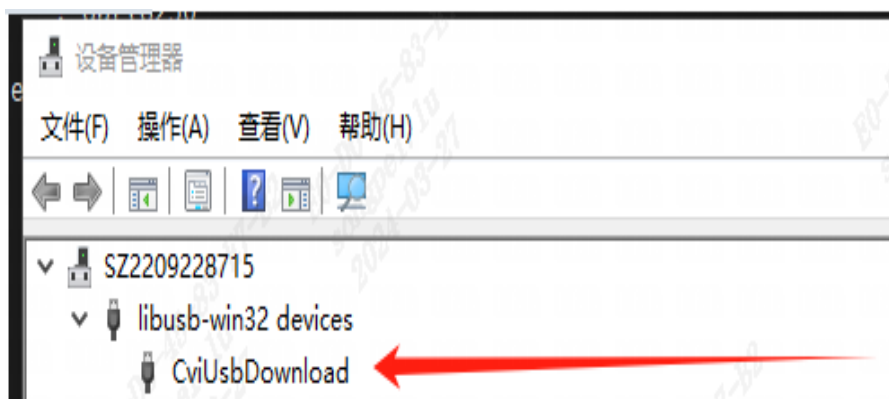


图 3.2: 如何验证驱动安装是否正确

注解: 不要重复安装驱动

3.3 卸载 usb 驱动 (USB 烧录)

Windows 提供了 pnputil 命令行工具，可以用来管理驱动程序包。可以使用以下命令列出所有第三方驱动程序：

```
pnputil /enum-drivers
```

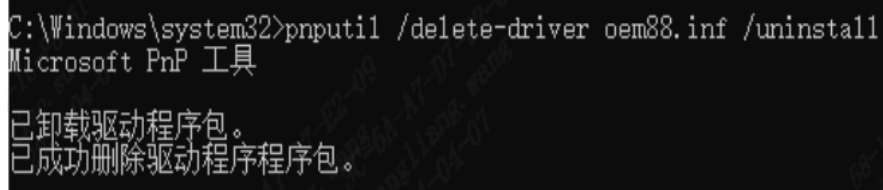
从输出的结果中筛选出来安装过的 cviusbdownload 的驱动：

```
发布名称: oem88.inf
原始名称: cviusbdownload.inf
提供程序名称: libusb-win32
类名: libusb-win32 Usb Devices
类 GUID: {eb781aaf-9c79-4523-a5df-642a87eca567}
驱动程序版本: 03/26/2024 1.0.0.0
签名者姓名: libusb-win32 (CviUsbDownload.inf) [Self]
```

```
发布名称: oem136.inf
原始名称: cviusbdownload.inf
提供程序名称: libusb-win32
类名: libusb-win32 Usb Devices
类 GUID: {eb781aaf-9c79-4523-a5df-642a87eca567}
驱动程序版本: 03/26/2024 1.0.0.0
签名者姓名: libusb-win32 (CviUsbDownload.inf) [Self]
```

从上面的例子，cviusb 驱动重复安装了两次，可以选择其中一个卸载掉（注意：需要用管理员权限打开命令行工具）：

```
pnputil /delete-driver oem88.inf /uninstall
```

```
C:\Windows\system32>pnputil /delete-driver oem88.inf /uninstall
Microsoft PnP 工具

已卸载驱动程序包。
已成功删除驱动程序包。
```

图 3.3: 如何验证驱动卸载是否正确

3.4 cviDownloadTool 工具

3.4.1 cviDownloadTool 工具介绍

cviDownloadTool 工具是一款可视化烧录工具，支持裸片烧录，也支持 U-Boot 烧录（需设备硬件支持）最多可以支持 8 台设备同时烧录。

注解： 通过 USB 来烧录单板需要满足以下条件：

- PC 机 USB 接口与单板的 USB2.0 口对接
- 单板必须满足一次系统复位，可以上电复位或者系统软复位

以上条件必须同时满足时，单板才能进入 USB 烧录流程

3.4.2 cviDownloadTool 使用方法

烧录工具操作流程

步骤 1.1

打开烧录工具，进入到主界面。

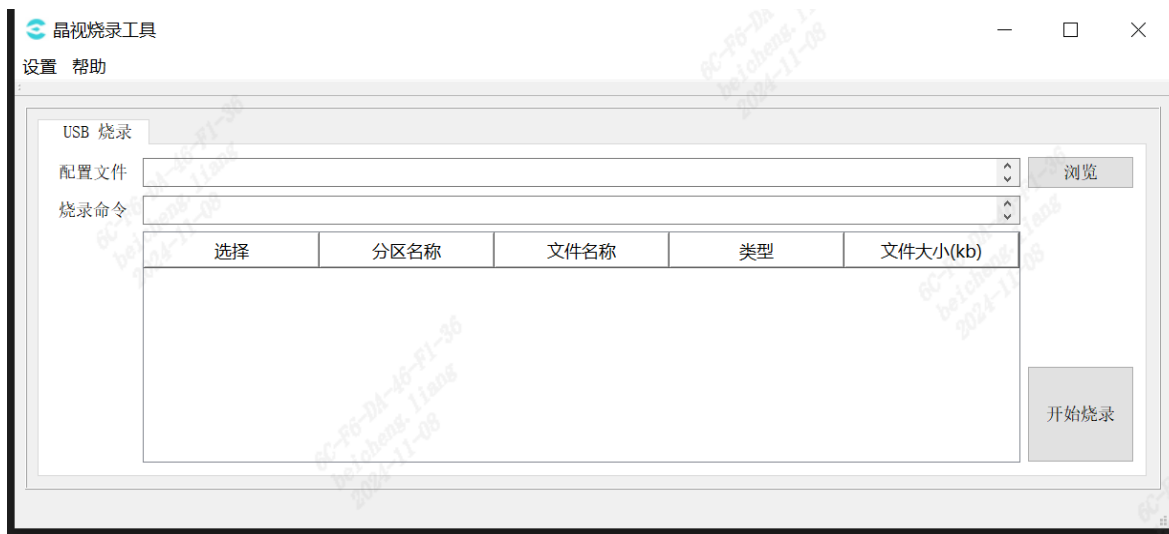


图 3.4: 晶视烧录工具界面

步骤 1.2

点击设置-> 语言进行语言选择。

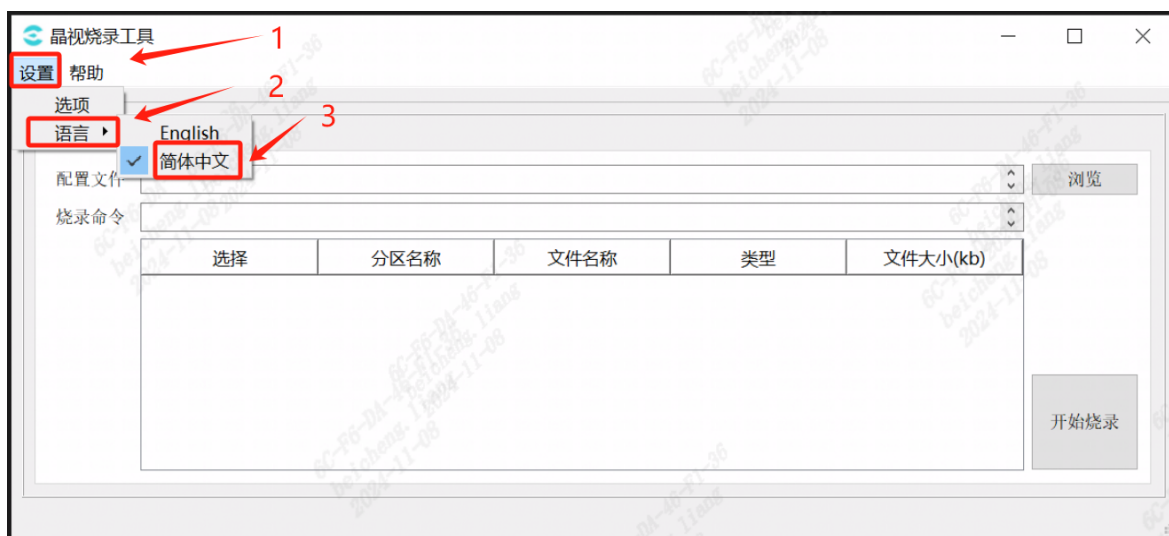


图 3.5: 晶视烧录工具语言选择

步骤 1.3

配置烧录工具参数

用户在烧录前的第一个步骤应当是打开设置栏，按照当次烧录的需求进行配置。打开烧录工具左上角的菜单栏，依次选择设置-> 选项，配置烧录所需的各个配置项。



图 3.6: 设置按钮



图 3.7: 设置界面

· 设置选项

– 烧录方式:

- * UART 烧录: 使用 UART 口进行烧录。
- * USB 烧录: 使用 USB 口进行烧录。

- Uboot 烧录: 若勾选此项, 使用板端自带的 U-Boot 进行烧录, 否则通过 USB 口加载镜像文件的 U-Boot 进行烧录。
- 校验模式: 若勾选此项, 在烧录时进行数据读回并进行校验。
- Raw 文件烧录: 若输入的镜像文件不具备文件头, 需勾选此项。
- SN 信息: 烧录时读取 sn 码, 并在烧录时打印到详细日志中。
- gpt 分区烧录: 勾选此选项, 烧录时包含 gpt 分区。
- 芯片类型: 需烧录设备对应的芯片类型, 当镜像包的 fip 文件中包含了该参数时, 优先使用 fip 中的参数。
- vendor 分区信息: 输入值为分区下标, 若该值非空时, 烧录时读取对应的 vendor 分区值并打印到详细日志中。
- 解包选项:
 - * 使用自定义解包工具: 勾选此项, 即可自定义解包工具。
 - * 解包工具: 选择自定义的解包工具, 该工具必须为 exe 文件, 且可以通过命令行执行的方式进行解包。
 - * 解包参数: 解包工具的命令行参数, 使用 [IMAGE] 和 [DESTDIR] 占位符来作为命令行中的镜像路径和目的路径, 如 -unpack [IMAGE] [DESTDIR]。
 - * 文件格式: 镜像压缩包的文件格式后缀。
- 烧录文件选择: 允许在主界面自定义分区表上的哪些文件不需要被烧录。
- 烧录详情: 在勾选此项后, 主界面会显示烧录时使用的命令行参数。
- 按键烧录: 若需烧录的板子需要按下烧录按键才进入烧录模式, 则需勾选此选项。

注解: 当不清楚设备的软硬件环境是否支持指定选项时, 请勿随意勾选该选项, 否则可能导致无法成功烧录。

步骤 1.4

点击浏览按钮, 找到需要被烧录的文件系统配置 xml 文件, 或镜像文件的压缩包。



图 3.8: 点击浏览按钮

注解: 若选择的是压缩包，在解压之前会检测同级目录下是否存在 `cviBurnUnpackDir` 文件夹，若存在会将其删除，以删除旧的镜像文件！

该文件用于保存从镜像中解压出来的所有文件，请在放置烧录固件文件的时候尽量避免所在路径存在同名文件夹。

请确保镜像压缩包文件的内容仅有一个 `xml` 或 `yaml` 文件，当前工具会检索 `xml` 或 `yaml` 文件的路径作为烧录路径。

若目录下必须同时存在 `xml` 和 `yaml` 文件，请手动解压镜像包，并点击浏览按钮选中 `xml` 或 `yaml` 文件。

步骤 1.5

选好文件后，点击确认，便会出现如 图 3.9，表格中会列出需要被烧录进去的文件以及他们的属性，用户可以通过手动取消勾选，表示哪些文件不需要被烧录。

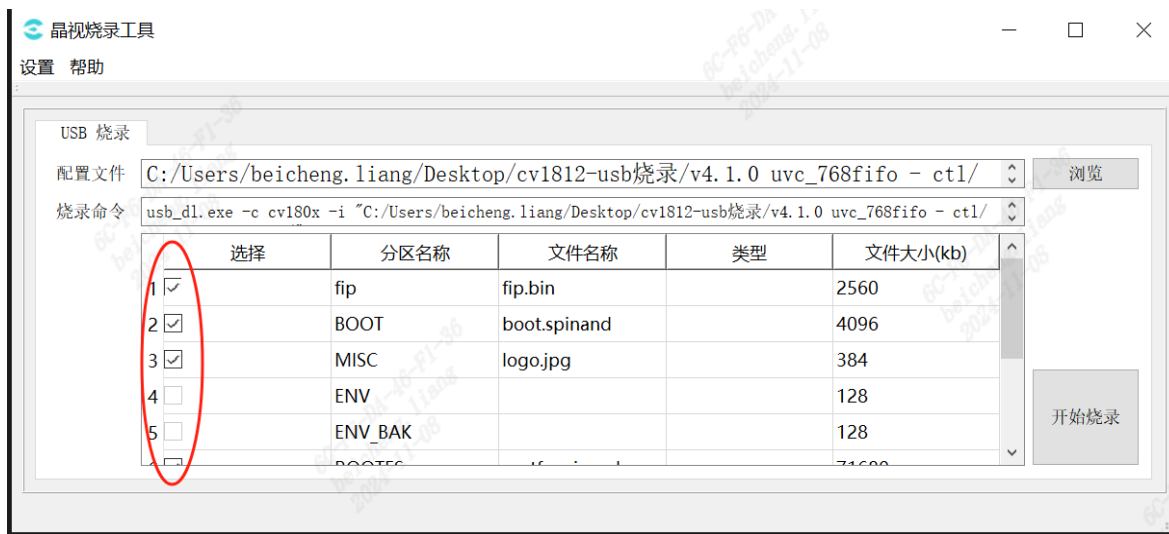


图 3.9: 选择需要下载的文件

步骤 1.6

点击开始烧录按钮，进入烧录子页面。

此时已经是就绪状态，可以直接开始烧录。

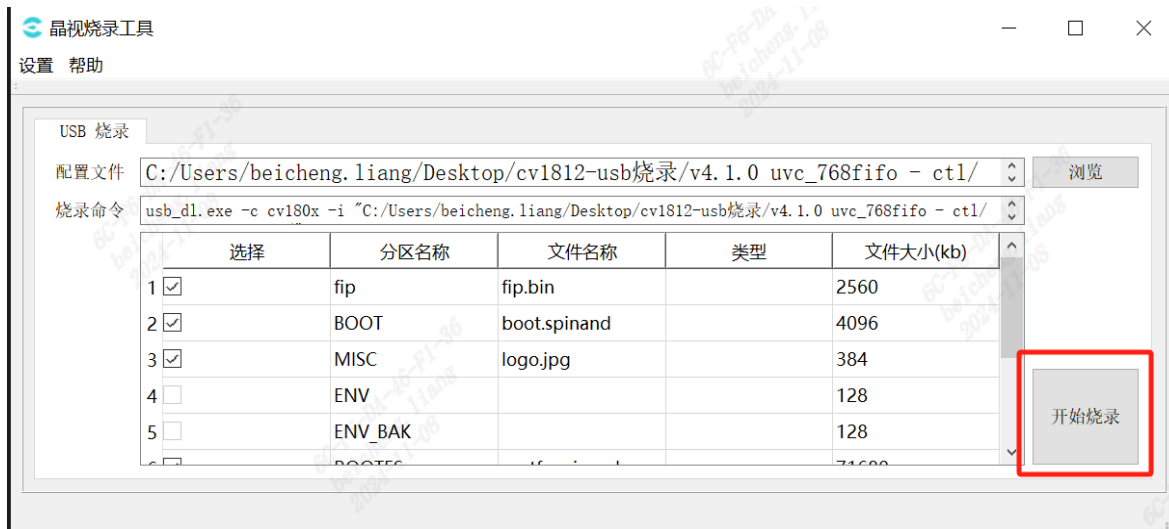


图 3.10: 点击开始烧录按钮

可以通过下方的按钮来改变烧录工具当前的状态。



图 3.11: 点击开始烧录按钮

步骤 1.7

进入就绪状态之后:

- UART 烧录: 插入 UART 口, 待进度条出现后, 设备上电即可进入烧录。
- USB 烧录: 可以将设备插入到 USB HUB 中并开机, 烧录程序会自动启动。

简要日志中打印的是当前插入的 USB 设备, 并且会在上方的方框标题显示插入板子对应的 USB 端口。

烧录开始了之后进度条会展示当前设备的烧录进度。



图 3.12: 插入 USB 设备开始烧录

附：USB 端口的对应关系：

将 USB 接口插入电脑之后，软件会自动识别插入的 USB 设备并为其编号，无需用户设置。

在电脑与单板建立连接之后，会在对应的 Board 窗口展示其端口号。

根据插入的顺序，依从左往右、从上至下的顺序排列在软件界面上。

USB 端口由两部分组成，例如端口：4,1 中，

4 表示当前 HUB 的编号，

1 表示该 HUB 上的 USB 编号

（实物 HUB 上标记的编号和电脑识别的可能不一致，这里以电脑标记的 HUB 号为主）。

建议用户在插入 USB 之后为其做好标记，以免造成混淆。

这样在量产时便可以通过软件显示的 USB 端口与实际单板一一对应，方便查看每个单板的状态。

注解： 为了使 USB 端口能够依次正确读取，请先将 USB HUB 插入电脑后再依次插入单板。

请勿先将单板插入 USB HUB 之后再一次性将 HUB 插入电脑，此操作可能造成无法正确读取 USB 端口。

步骤 1.8

烧录成功：

烧录完成后状态栏会有一个“版 x 烧录完成.”提示，图标栏也会显示如下图标。

烧录成功后只需拔掉设备，然后插入新的设备即可进行新一轮的烧录。

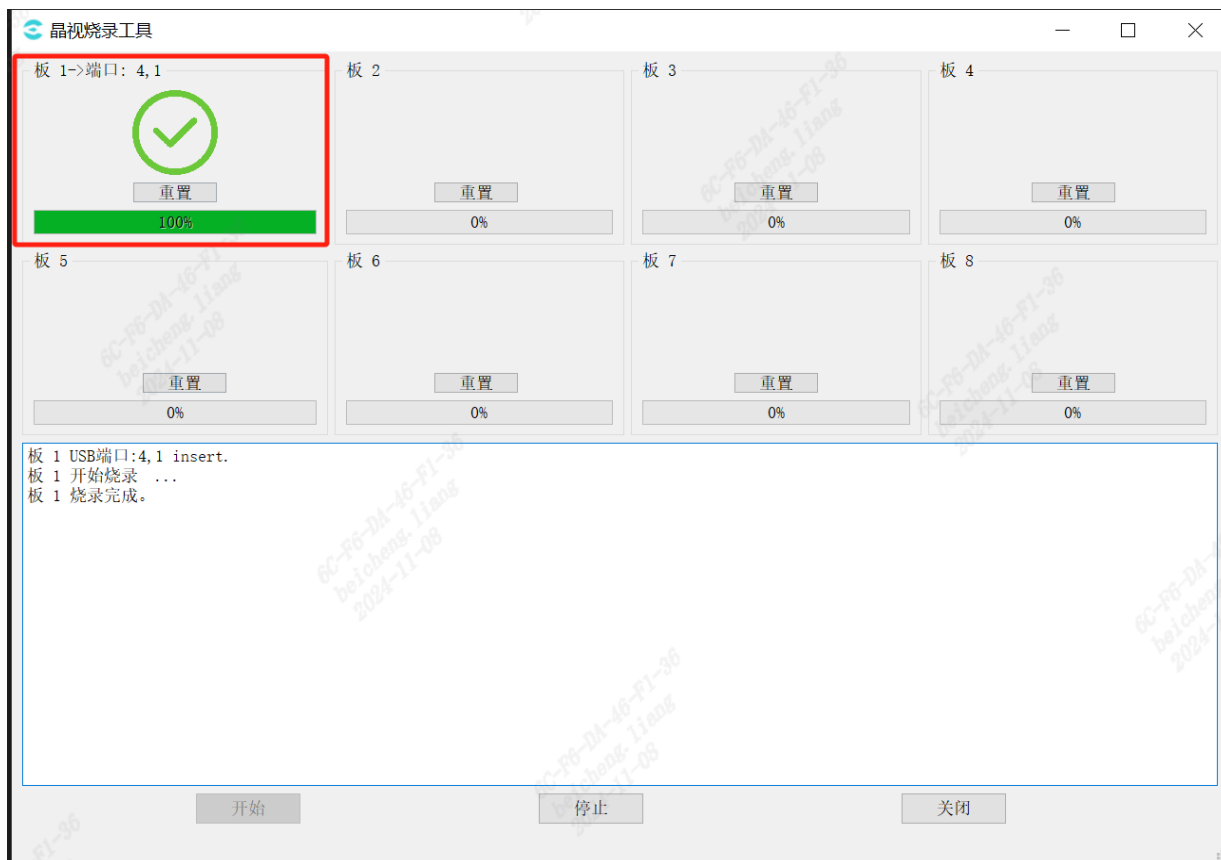


图 3.13: 烧录成功

烧录失败：

若烧录过程中因各种原因导致烧录失败的话，对应烧录失败的 Board 框内图标会显示“F”，日志会提示“版 x 烧录失败，请检查连接并点击重置按钮.”，

如 图 3.14。此时可点击“重置”按钮对单个板子重新启动烧录程序。



图 3.14: 烧录失败

重置:

每个板子对应的框内都含有重置按钮，点击重置按钮可将对应的烧录程序重置，重新进入就绪状态，此时将未烧录成功的板子需要将设备拔掉、关机、再重新插入、开机，烧录程序会重新启动对板子进行烧录。

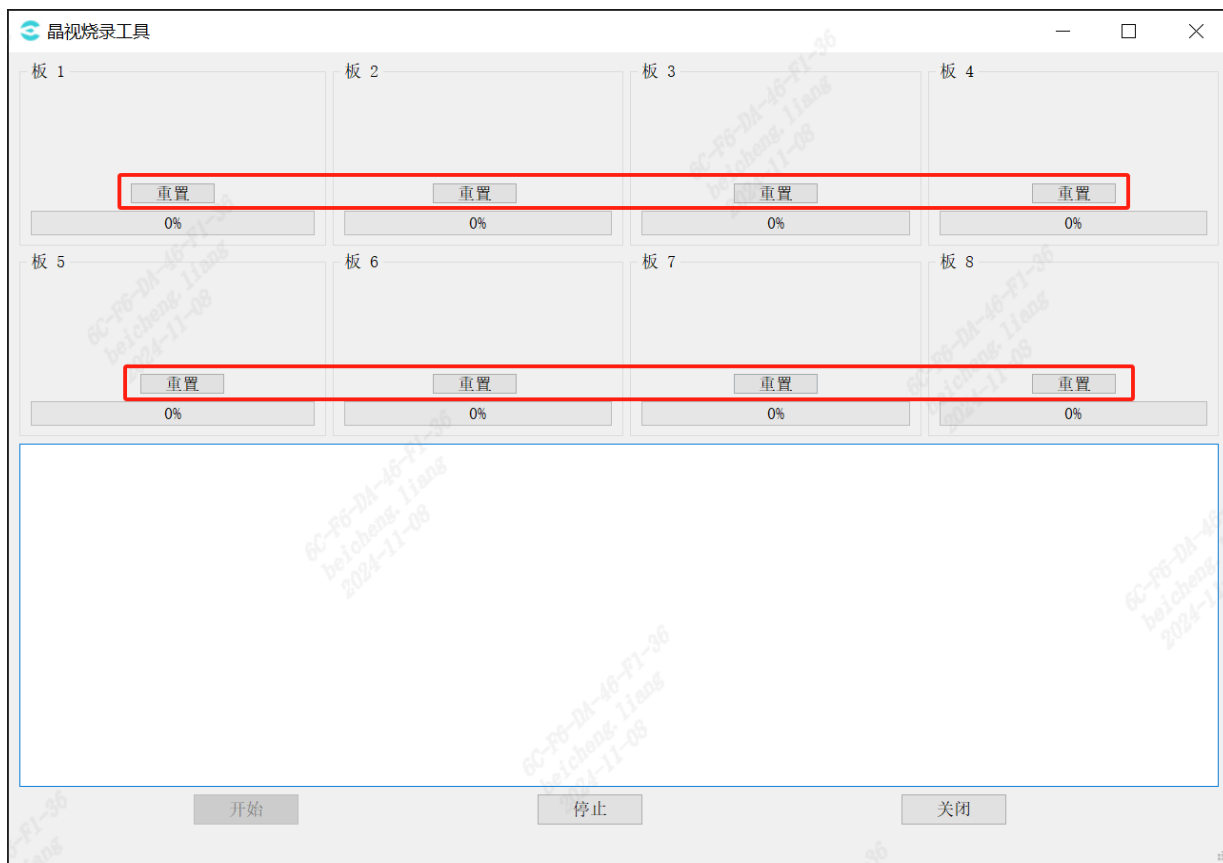


图 3.15: 重置按钮

3.5 cviDownloadTool 工具

3.5.1 cviDownloadTool 工具介绍

cviDownloadTool 工具是一款可视化烧录工具，支持裸片烧录，也支持 U-Boot 烧录（需设备硬件支持）最多可以支持 8 台设备同时烧录。

注解： 通过 USB 来烧录单板需要满足以下条件：

- PC 机 USB 接口与单板的 USB2.0 口对接
- 单板必须满足一次系统复位，可以上电复位或者系统软复位

以上条件必须同时满足时，单板才能进入 USB 烧录流程

3.5.2 cviDownloadTool 使用方法

烧录工具操作流程

步骤 1.1

在工具所在路径中打开 cmd 命令行窗口，将 upgrade.zip 压缩包解压到同级目录。

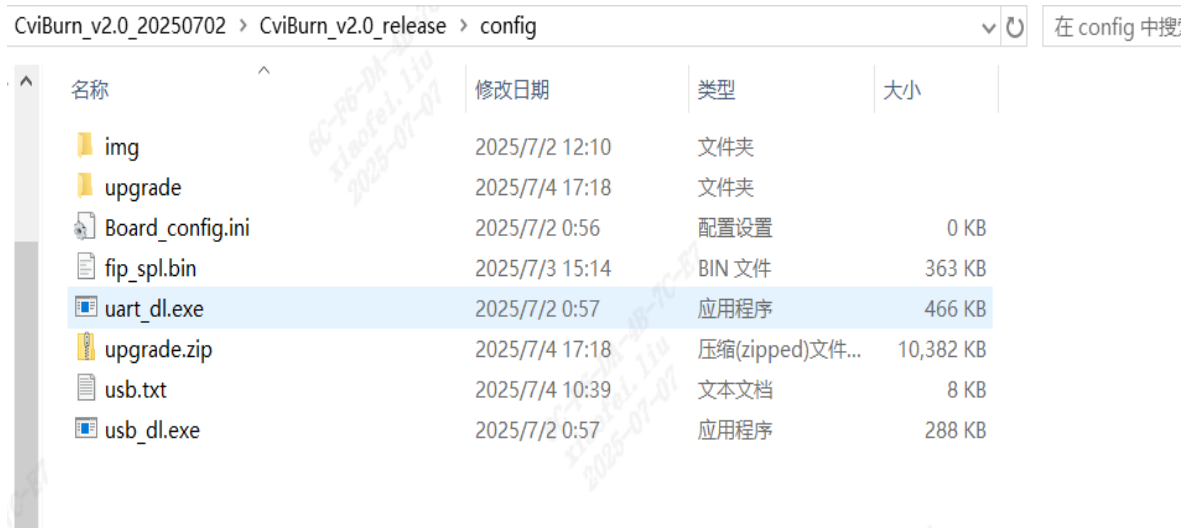


图 3.16: usb_dl.exe 和 uart_dl.exe 所在路径

usb_dl.exe 和 uart_dl.exe 使用

1. usb_dl.exe - usb_dl.exe -c cv184x -i .upgrade
2. uart_dl.exe - uart_dl.exe -c cv184x -p COM3 -i .upgrade

4 SD 卡烧录

4.1 概述

4.1.1 介绍

本章介绍如何使用 SD 卡烧录整个单板系统文件，

4.1.2 烧录前的准备工作

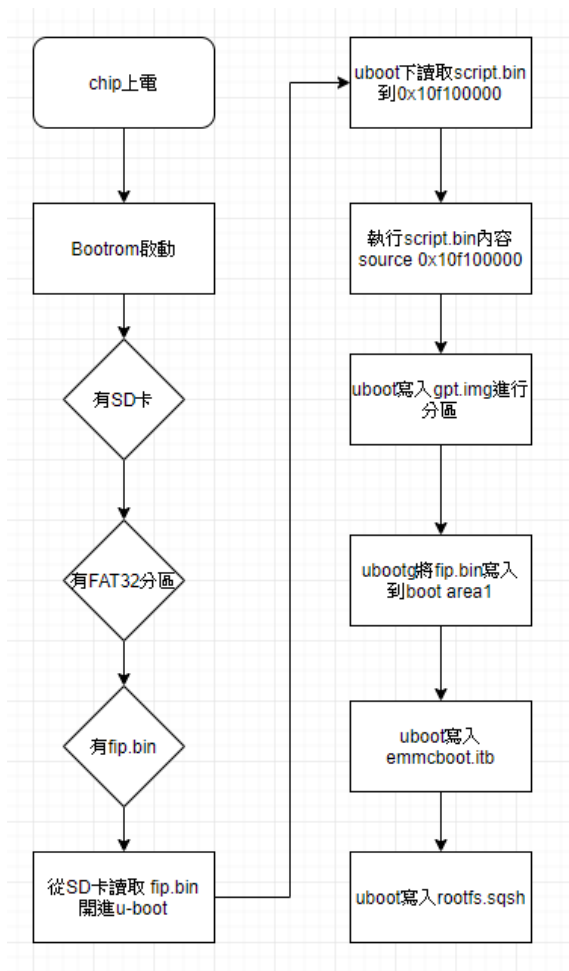
- 烧录前的准备工作如下：
 - 文件系统镜像文件。
 - 准备 SD 卡一张。

4.2 使用 SD 卡裸烧

4.2.1 使用前准备

1. 参考【Linux 开发环境用户指南】【1.2 如何编译 BSP】编译下列档案：
 - fip.bin - bootloader + uboot
 - boot.emmc/boot.spinand/boot.spinor - minimal Linux image(可选)
 - rootfs.emmc/rootfs.spinand/rootfs.spinor - rootFS(可选)
 - system.emmc/system.spinand/system.spinor - rw 分区 (可选)
 - cfg.emmc/cfg.spinand/cfg.spinor - config rw 分区 (可选)
2. 一张 FAT32 格式的 TF 卡 (micro SD)

4.2.2 裸烧流程解释



4.2.3 操作过程

- 将 fip.bin, *.emmc/*.spinand/*.spinor 放到 SD 卡中
- 将 SD 卡插入 Cvitek EVB 的 SD 卡槽中
- 将 Cvitek EVB 平台开机

4.2.4 操作实例

使用前确认档案

SPINAND

| 名稱 | 修改日期 | 類型 | 大小 |
|--|-------------------|------------|-----------|
|  boot.spinand | 2021/6/2 上午 04:13 | SPINAND 檔案 | 7,213 KB |
|  cfg.spinand | 2021/6/2 上午 04:13 | SPINAND 檔案 | 2,049 KB |
|  fip.bin | 2021/6/2 上午 04:10 | BIN 檔案 | 374 KB |
|  rootfs.spinand | 2021/6/2 上午 04:13 | SPINAND 檔案 | 28,929 KB |
|  system.spinand | 2021/6/2 上午 04:13 | SPINAND 檔案 | 1,921 KB |

插入 SD 卡, 并将 Cvitek EVB 平台接上电源开机后, 自动启动刻录程序

平台刻录完成时, 可于 UART 端口看到以下讯息.

将平台断电再重开机即刻录完成

```
## Resetting to default environment
Start SD downloading...
switch to partitions #0, OK
mmc0 is current device
403968 bytes read in 19 ms (20.3 MiB/s)
spinor id = C2 20 18
SF: Detected MX25L12835F with page size 256 Bytes, erase size 4 KiB, total 16 MiB
device 0 offset 0x0, size 0x62a00
403968 bytes written, 0 bytes skipped in 3.73s, speed 134480 B/s
sf update speed 0.131 MB/s
64 bytes read in 2 ms (31.3 KiB/s)
Header Version:1
2999536 bytes read in 135 ms (21.2 MiB/s)
device 0 offset 0x100000, size 0x2dc4b0
2999472 bytes written, 0 bytes skipped in 22.529s, speed 136315 B/s
sf update speed 0.133 MB/s
64 bytes read in 2 ms (31.3 KiB/s)
Header Version:1
3100736 bytes read in 139 ms (21.3 MiB/s)
SF: 10485760 bytes @ 0x420000 Erased: OK
device 0 offset 0x420000, size 0x2f5000
SF: 3100672 bytes @ 0x420000 Written: OK
sf write speed 0.649 MB/s
64 bytes read in 1 ms (62.5 KiB/s)
Header Version:1
228 bytes read in 2 ms (111.3 KiB/s)
SF: 524288 bytes @ 0xe20000 Erased: OK
device 0 offset 0xe20000, size 0xa4
SF: 164 bytes @ 0xe20000 Written: OK
sf write speed 0.23 MB/s
Saving Environment to SPIFlash... Erasing SPI flash...Writing to SPI flash...done
Valid environment: 2
OK
cv181x_c906#
```


4.2.5 使用 upgrade.zip 进行升级

1. 参考【Linux 开发环境用户指南】【1.2 如何编译 BSP】编译 upgrade.zip
2. 将 upgrade.zip 拷贝至 SD 卡
3. 解压缩 upgrade.zip (请将档案解压至 SD 卡的根目录)

4.2.6 注意事项

请确认 SD Card 有被正确格式化成 FAT32

4.2.7 设置 eMMC ECSD register

当使用 SD 卡进行裸烧时，会用 u-boot 内建的 eMMC 驱动来对 ECSD 进行修改，主要是针对 ECSD[162] bit，也就是将 n_RST 的功能开启，具体刻录方式如下：

1. 在 u-boot 下输入以下指令来启 n_RST 功能
- u-boot # mmc fuse_rstn 0

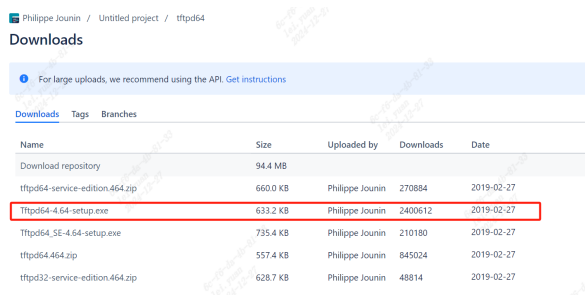
5 TFTP 烧录

5.1 使用 TFTP 加载固件到 DDR

5.1.1 使用前准备

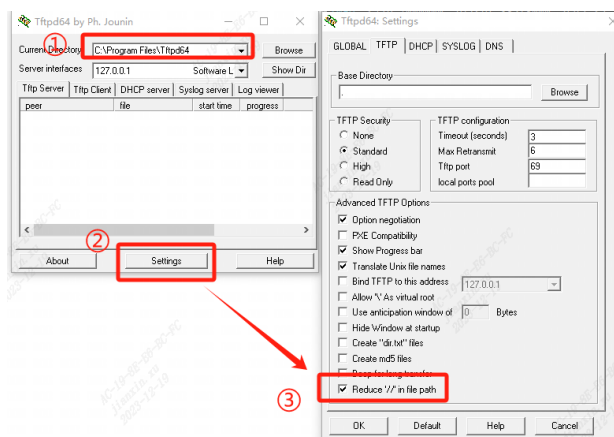
1. 下载 tftp 上位机软件:

- <https://bitbucket.org/phjounin/tftpd64/downloads/>



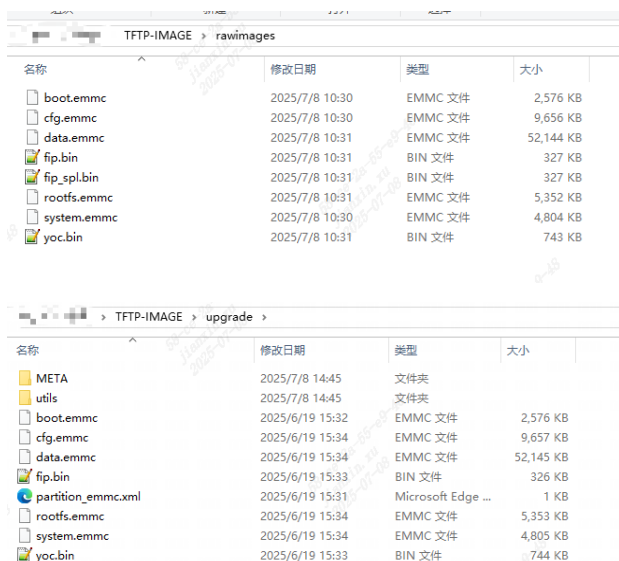
2. 对 tftpd64 工具进行以下配置:

- 进入 setting 界面, 勾选 TFTP 页面中的 Reduce ‘//’ in file path。



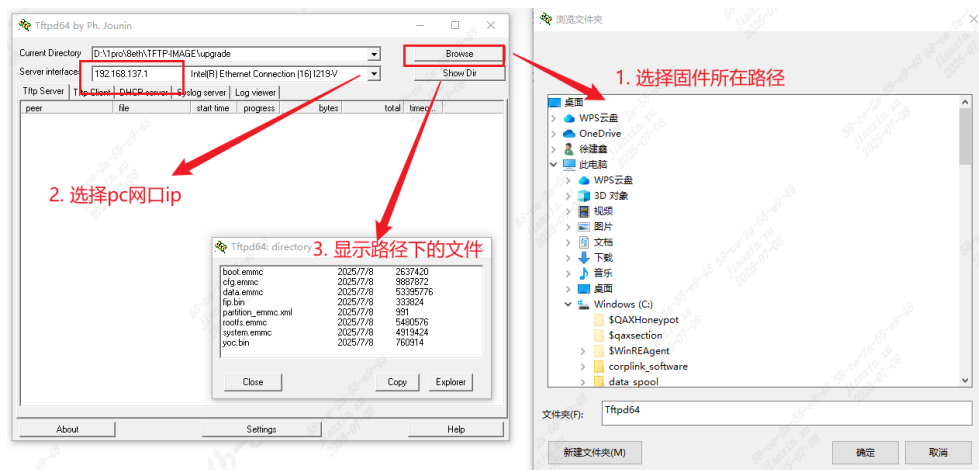
3. 准备固件包

- 如果是使用源码自行编译的固件, “sdk/install/soc_芯片名”路径下有名为 upgrade.zip 的压缩包和 rawimages 的文件夹。如下图所示:



5. 设置 TFTP 软件

- 将升级包放到 tftp 目录下，或者使用上位机软件“Browse”定位到固件路径，路径不要有中文和空格等非法字符。
- 点击 PC server 的“Show Dir”按钮，可以显示当前路径下的文件。
- Server interfaces 设置为本机的网络接口，例如 192.168.137.1;



4. 配置 PC 和板卡在网段

- 通过网线连接 PC 端网口和板卡端 LAN 口
- 配置 PC 端 ip 为 192.168.137.1
- 配置板卡端 ip 为 192.168.137.11
- 通过在 uboot 命令行下键入以下命令来设置板卡 ip，执行 saveenv 保存设置：

```

1  setenv ipaddr 192.168.137.11
2  setenv gatewayip 192.168.137.1
3  setenv serverip 192.168.137.1
4  ...

```

ipaddr 是板端 IP，serverip 是 tftp server 所在 PC 的 IP，gatewayip 是网关地址。

5. 测试连通性 - 在 uboot 下直接 ping PC 端 ip 地址，下图表示可以正常通信

```
1 ping 192.168.137.1
```

```
cv184x# setenv serverip 192.168.137.1
cv184x# setenv ipaddr 192.168.137.100
cv184x# setenv gatewayip 192.168.137.1
cv184x#
cv184x# ping 192.168.137.1
Speed: 100, full duplex
Using ethernet@4070000 device
host 192.168.137.1 is alive
cv184x#
```

5.1.2 TFTP 加载固件

1. 从 tftp 服务器下载文件到指定内存地址

- tftp 下载固件的时候，要先下载到 DDR 内存中，为了避免踩到 uboot 的内存空间，可以从 0x82000000 开始写入。

```
1 # 以emmc 举例如下
2 # 每次加载固件前将ddr 目标区域写0
3 mw.b 0x82000000 0 0x20000000
4 # 加载文件到DDR,
5 # 可以加载文件到同一位置，并参考下一节把DDR内文件写入Flash，然后重复操作下一个文件
6 # 也可以加载到DDR不同位置，并参考下一节把DDR内文件写入Flash，请注意不要踩内存
7 tftpboot 0x82000000 fip.bin
8 # boot等文件需要使用rawimages下的文件
9 tftpboot 0x82100000 yoc.bin
10 tftpboot 0x82280000 boot.emmc
11 tftpboot 0x82580000 rootfs.emmc
12 tftpboot 0x82d80000 cfg.emmc
13 tftpboot 0x83780000 system.emmc
```

2. 头部处理

- 如果使用 Upgrade 压缩包中的固件就需要删除固件的前 128 字节头 (boot.emmc 和 rootfs.emmc 头部，而且 rootfs.emmc 除了开头 128 字节头外，每隔 16M 都有 64 字节的头部)
- 使用 rawimages 文件夹中的固件就不用删除头部

5.2 烧写固件到 Flash

5.2.1 烧写前准备

1. 查看加载进来的固件的大小

- 在使用 TFTP 加载固件到 DDR 时，可以从打印中看到文件的大小

```

cv184x# tftpboot 0x82000000 fip.bin
Speed: 100, full duplex
Using ethernet@4070000 device
TFTP from server 192.168.137.1; our IP address is 192.168.137.11
Filename 'fip.bin'.
Load address: 0x82000000
Loading: #####
2.4 MiB/s
done
Bytes transferred = 333824 (51800 hex)
cv184x#

```

2. 查看每个固件在 Flash 中的偏移地址

- 使用 U-BOOT 的 printenv 命令可以查看这部分信息，或者打开 SDK/u-boot-2021.10/include/cvippart.h，此文件内也包含了这部分信息

```

cv184x# printenv
2nd_PART_OFFSET=0x0
2nd_PART_SIZE=0x1800
BOOT_PART_OFFSET=0x1800
BOOT_PART_SIZE=0x4000
CFG_PART_OFFSET=0x3c500
CFG_PART_SIZE=0x7710
DATA_PART_OFFSET=0x43c10
DATA_PART_SIZE=0x600000
ENV_PART_OFFSET=0x5c000
ENV_PART_SIZE=0x100
MISC_PART_OFFSET=0x5800
MISC_PART_SIZE=0x400
ROOTFS_PART_OFFSET=0x5000
ROOTFS_PART_SIZE=0x22800
SYSTEM_PART_OFFSET=0x28500
SYSTEM_PART_SIZE=0x14000
baudrate=115200
board=cv184x
board_name=cv184x
bootm=mmc1 update || run emmcboot
bootdelay=0
consoledev=tttyS0
cpus=armv8
dl_flag=prog
emmcboot=setenv bootargs $(root) $(mtddparts) console=${consoledev},$baudrate $othbootargs;mmc dev 0 ;mmc read $(uimage_addr) $(BOOT_PART_OFFSET) $(BOOT_PART_SIZE) ;boot
eth=ethernet@4070000
fileaddr=0x1800000
filesize=2ec040
gatewayip=192.168.137.1
ipaddr=192.168.137.11
mtddparts=blkdevparts=mmcblk0:3072K(2nd),8192K(BOOT),512K(MISC),128K(ENV),70656K(ROOTFS),40960K(SYSTEM),15240K(CFG),3145728K(DATA);mmcblk0boot0:1M(fip),1M(fip_bak);
netdev=eth0
netmask=255.255.255.0
othbootargs=earlycon=${} loglevel=0
root=roothfstype=squashfs rootwait ro roots=/dev/mmcblk0p5
sdboot=setenv bootargs $(root) $(mtddparts) console=${consoledev},$baudrate $othbootargs;echo Boot from SD with ramboot.itb;mmc dev 1 66 fatload mmc 1 $(uimage_addr) ram
u_bootcmd=emmc;
serverip=192.168.137.1

```

```

#ifndef CVIPART_H
#define CVIPART_H
#define CONFIG_ENV_IS_IN_MMC
#define CONFIG_ENV_SECT_SIZE 0x40000
#define CONFIG_SYS_MMC_ENV_DEV 0
#define CONFIG_SYS_MMC_ENV_PART 0
#define PART_LAYOUT "blkdevparts=mmcblk0:3072K(2nd),8192K(BOOT),512K(MISC),128K(ENV),70656K(ROOTFS),40960K(SYSTEM),15240K(CFG),3145728K(DATA);mmcblk0boot0:1M(fip),1M(fip_bak);"
#define ROOTFS_DEV "/dev/mmcblk0p5"
#define CONFIG_ENV_OFFSET 0x80000
#define CONFIG_ENV_SIZE 0x20000
#define SECOND_OFFSET 0x0
#define SECOND_SIZE 0x30000
#define LOGO_OFFSET 0xB0000
#define LOGO_SIZE 0x80000
#define PARTS_OFFSET \
"2nd_PART_OFFSET=0x0\0" \
"2nd_PART_SIZE=0x1800\0" \
"BOOT_PART_OFFSET=0x1800\0" \
"BOOT_PART_SIZE=0x4000\0" \
"MISC_PART_OFFSET=0x5800\0" \
"MISC_PART_SIZE=0x400\0" \
"ENV_PART_OFFSET=0x5c00\0" \
"ENV_PART_SIZE=0x100\0" \
"ROOTFS_PART_OFFSET=0x5d00\0" \
"ROOTFS_PART_SIZE=0x22800\0" \
"SYSTEM_PART_OFFSET=0x28500\0" \
"SYSTEM_PART_SIZE=0x14000\0" \
"CFG_PART_OFFSET=0x3c500\0" \
"CFG_PART_SIZE=0x7710\0" \
"DATA_PART_OFFSET=0x43c10\0" \
"DATA_PART_SIZE=0x600000\0"
#define SPL_BOOT_PART_OFFSET 0x1800
#endif

```

5.2.2 烧写固件

1. 从上述内存地址把固件加载到 emmc 指定块位置

- 写文件进 emmc 时需要注意，文件大小的单位为“块”，即实际大小/块大小 512，请对齐后再进行 emmc 操作

```

1  # 1. 清空ddr对应区域，并加载文件到ddr
2  # 2. 写fip.bin到emmc boot0分区，
3  #   fip.bin 在boot分区，其他文件在user分区；选择 emmc 并切换到 boot 分区
4  mmc dev 0 1
5  #   写fip.bin进flash，参数为DDR地址，flash地址，文件大小
6  mmc write 0x82000000 0x0 0x800
7  #   对fip.bin进行双备份
8  mmc write 0x82000000 0x800 0x800
9  # 3. 写其他文件到emmc boot0分区
10 #   选择 emmc 并切换到 user 分区
11 mmc dev 0 0
12 # 写yoc.bin
13 mmc write 0x82100000 0x0 0xc00
14 # 写boot.emmc
15 mmc write 0x82280000 0x1800 0x1800
16 # 写rootfs.emmc
17 mmc write 0x82580000 0x5d00 0x4000
18 # 写cfg.emmc
19 mmc write 0x82d80000 0x3c500 0x5000
20 # 写system.emmc
21 mmc write 0x83780000 0x28500 0x2800

```

2. 从上述内存地址把固件加载到 spinand 指定位置

```

1  # 使用mtd 命令也可以查看当前spinand 分区情况
2  mtd

```

```

cvl84x# mtd

device nand0 <cvsnfc>, # parts = 10
#: name          size          offset          mask_flags
0: fip            0x00280000      0x00000000      0
1: 2nd            0x00300000      0x00280000      0
2: BOOT           0x00800000      0x00580000      0
3: MISC           0x00060000      0x00d80000      0
4: ENV            0x00020000      0x00de0000      0
5: ENV_BAK        0x00020000      0x00e00000      0
6: ROOTFS         0x04600000      0x00e20000      0
7: SYSTEM         0x01400000      0x05420000      0
8: CFG            0x00400000      0x06820000      0
9: DATA          0x01000000      0x06c20000      0

active partition: nand0,0 - (fip) 0x00280000 @ 0x00000000

defaults:
mtdids : nand0=cvsnfc
mtdparts:

```

```

1  # 擦除fip.bin
2  nand erase.part fip
3  # 写fip 要使用cvi_sd_update 其他的使用nand write
4  # cvi_sd_update 可以对fip.bin 做双备份
5  cvi_sd_update 0x82000000 spinand fip

```

(下页继续)

(续上页)

```
6
7 # 擦除yoc.bin
8 nand erase.part 2nd
9 # 写yoc.bin offset和size要按照 page size 对齐，单位为字节
10 nand write 0x82100000 2nd 0x180000
11 # 擦除boot.spinand
12 nand erase.part BOOT
13 # 写boot.spinand
14 nand write 0x82280000 BOOT 0x300000
15 # 擦除rootfs.pinand
16 nand erase.part ROOTFS
17 # 写rootfs.spinand
18 nand write 0x82580000 ROOTFS 0x800000
19 # 擦除cfg.spinand
20 nand erase.part CFG
21 # 写cfg.spinand
22 nand write 0x82d80000 CFG 0xA00000
23 # 擦除system.spinand
24 nand erase.part SYSTEM
25 # 写system.spinand
26 nand write 0x83780000 SYSTEM 0x300000
```

3. 从上述内存地址把固件加载到 spinor 指定位置

```
1 # 探测并初始化 SPI-NOR
2 sf probe
3 # 擦除flash，不然无法写入，以16MB容量为例，0x0是起始地址，16M = 0x1000000
4 sf erase 0x0 0x1000000
5 # 根据printenv命令得到的信息将DDR的数据写入Flash，按page 对齐。单位为字节。
6 # 写fip.bin
7 sf write 0x82000000 0x0 0x100000
8 # 写yoc.bin
9 sf write 0x82100000 0x200000 0x100000
10 # 写boot.spinor
11 sf write 0x82280000 0x300000 0x300000
12 # 写rootfs.spinor
13 sf write 0x82580000 0x640000 0x480000
```

6 烧录器烧录

6.1 概述

6.1.1 介绍

本章介绍如何使用烧录器烧录整个单板系统文件，

6.1.2 烧录前的准备工作

- 烧录前的准备工作如下：
 - 文件系统 rawimage 镜像文件。
 - 烧录器。

6.2 预烧程序

6.2.1 使用前准备

1. 以 SPINAND 为例，从 SDK 编译下列档案:

```
fip.bin - bootloader + uboot  
boot.spinand - Linux image  
logo.jpg - boot logo (Optional)  
rootfs.spinand - root file system  
system.spinand - system partition (Optional)  
cfg.spinand - encrypted ISP PQ partition (Optional)
```

fip.bin 从 install/<board name> 目录下取得:

```
$ ls -al install/soc_cv1820_wevb_0005b_spinand
```



```
total 66588
drwxr-xr-x 8 alec alec 4096 八 16 22:28 ./
drwxr-xr-x 21 alec alec 4096 七 21 13:53 ../
-rw-rw-r-- 1 alec alec 7134746 八 16 22:28 boot.spinand
-rw-rw-r-- 1 alec alec 1966208 八 16 22:28 cfg.spinand
drwxrwxr-x 2 alec alec 4096 八 16 22:28 elf/
-rw-rw-r-- 1 alec alec 385024 八 16 22:11 fip.bin
drwxr-xr-x 2 alec alec 4096 七 21 14:12 fip_pre/
-rw-r--r-- 1 alec alec 660 八 16 22:28 partition_spinand.xml
drwxrwxr-x 2 alec alec 4096 八 16 22:28 rawimages/
drwxrwxr-x 18 alec alec 4096 八 16 22:28 rootfs/
-rw-rw-r-- 1 alec alec 24510592 八 16 22:28 rootfs.spinand
drwxrwxr-x 2 alec alec 4096 八 16 22:28 system/
-rw-rw-r-- 1 alec alec 1966208 八 16 22:28 system.spinand
drwxrwxr-x 3 alec alec 4096 八 16 22:28 tools/
-rw-rw-r-- 1 alec alec 32172224 八 16 22:29 upgrade.zip
```

*.spinand 从 install/<board name>/rawimages 目录下取得:

```
$ ls -al install/soc_cv1820_wevb_0005b_spinand/rawimages
```

```
total 34748
drwxrwxr-x 2 alec alec 4096 八 16 22:28 ./
drwxr-xr-x 8 alec alec 4096 八 16 22:28 ../
-rw-rw-r-- 1 alec alec 7134618 八 16 22:28 boot.spinand
-rw-rw-r-- 1 alec alec 1966080 八 16 22:28 cfg.spinand
-rw-rw-r-- 1 alec alec 24510464 八 16 22:28 rootfs.spinand
-rw-rw-r-- 1 alec alec 1966080 八 16 22:28 system.spinand
```

注意: 注意: rawimages 目录下的 *.spinand 文件是用于烧录器烧录的裸 images; 上一层的目录下的 *.spinand 文件为 CVITEK SD card/USB 烧录专用的格式, 基于裸的 image 多加了 128 bytes header 信息

2. 进入 build/tools/common/spinand_tool/fip_maker, 执行如下操作:

```
make clean; make
```

3. 拷贝 fip.bin 至该目录下, 执行 ./fip_maker {pagesize} {DID/MID} {input_path} {output_path}, 其中 {pagesize} 和 {DID/MID} 参数值请从 spi nand 颗粒 datasheet 中获取

示例:

```
./fip_maker 2048 0x71e5 ./fip.bin ./fip_out.bin
```

1. 若无错误, 会产出 fip_out.bin。此 fip_out.bin 即为烧录器烧录所需的文件备注: 烧录器以及 tftp 烧录均需要使用上述 fip_out.bin 文件

执行上述三个步骤准备好二进制文件后, 可进行烧录器预烧。

6.2.2 分区表

CVITEK 方案 Flash 分区表以 xml 格式定义，细节请参考《Flash 分区工具使用指南》。

Flash 分区以 xml 格式定义，以 boards/default/partition/partition_spinand_page_2k.xml 为例：

```
<physical_partition type="spinand">
  <partition label="fip" size_in_kb="2560" file="fip.bin"/>
  <partition label="BOOT" size_in_kb="8192" file="boot.spinand"/>
  <partition label="MISC" size_in_kb="384" file="logo.jpg" />
  <partition label="ENV" size_in_kb="128" file="" />
  <partition label="ROOTFS" size_in_kb="71680" file="rootfs.spinand" />
  <partition label="SYSTEM" size_in_kb="20480" file="system.spinand" mountpoint="" type=
  ↪ "ubifs" />
  <partition label="CFG" size_in_kb="4096" file="cfg.spinand" mountpoint="/mnt/cfg" type=
  ↪ "ubifs" />
  <partition label="DATA" file="" mountpoint="/mnt/data" type="ubifs" />
</physical_partition>
```

以 2KB page size 128KB blocksize 的 NAND flash 为例：由 xml 文件上数据，将各分区大小换算成 block 大小后（公式：block 个数 = 分区大小 / 单一 block 大小），如下所示：

| Partition | Start block offset | Number of blocks | Binary files |
|-----------|--------------------|------------------|----------------|
| FIP | 0 | 20 | fip.bin |
| BOOT | 24 | 64 | boot.spinand |
| MISC | 顺排（遇坏块则跳过） | 3 | logo.jpg |
| ENV | 顺排（遇坏块则跳过） | 1 | Null（无内容） |
| ENV_BAK | 顺排（遇坏块则跳过） | 1 | Null（无内容） |
| ROOTFS | 顺排（遇坏块则跳过） | 560 | rootfs.spinand |
| SYSTEM | 顺排（遇坏块则跳过） | 160 | system.spinand |
| CFG | 顺排（遇坏块则跳过） | 32 | cfg.spinand |
| DATA | 顺排（遇坏块则跳过） | Don' t Care | Null（无内容） |

6.2.3 烧写规则

FIP 分区

FIP 分区包含两个部分： 处理器相关的 Bootloader(无开源)，u-boot. CVITEK 编译流程会自动将两者打包成一个 fip.bin nand 烧录时烧录逻辑从 block 0~19 之间依序挑选好块，总共烧写两份，第一份会写在 block 0~9, 第二份会烧写在 block 10~19，互为备份。

fip.bin 本身烧入进 spinand 大概会使用到 3~4 个 blocks，但因 spinand 特性问题，block 可能会出现坏块状况，所以剩余未使用的 block 预留用于出现坏块时使用。

例一

没有坏块的话，将第一份 fip.bin 烧写至 block 0, 1, 2, 3, 4；第二份 fip.bin 烧写至 block 9, 10, 11, 12, 13。

例二

若 block 4, 11 为坏块, 请将第一份 fip.bin 烧写至 block 0, 1, 2, 3, 5; 第二份 fip.bin 烧写至 block 9, 10, 12, 13, 14

其他分区

照分区表配置, 依序烧写, 遇到坏块则略过, 跳下一个好块再烧写。