



# T113 Longan linux SDK

## 开发环境配置手册

**版本号: 0.8**

**发布日期: 2021.9.2**

## 版本历史

| 版本号 | 日期        | 制/修订人   | 内容描述                |
|-----|-----------|---------|---------------------|
| 0.1 | 2021.6.21 | KPA0526 | 初建版本                |
| 0.2 | 2021.6.22 | KPA0526 | 修改部分内容，添加挂载 U 盘内容   |
| 0.8 | 2021.9.2  | KPA0526 | 修改内容，适配 V0.8 版本 SDK |

## 目 录

|                              |           |
|------------------------------|-----------|
| <b>1 前言</b>                  | <b>1</b>  |
| 1.1 文档简介                     | 1         |
| 1.2 目标读者                     | 1         |
| 1.3 适用范围                     | 1         |
| 1.4 相关术语                     | 1         |
| <b>2 目录结构</b>                | <b>2</b>  |
| 2.1 brandy                   | 2         |
| 2.2 buildroot                | 2         |
| 2.3 kernel                   | 3         |
| 2.4 platform                 | 4         |
| 2.5 tools                    | 4         |
| 2.6 device/config/chips/t113 | 5         |
| <b>3 开发环境配置</b>              | <b>7</b>  |
| 3.1 Linux 服务器开发环境搭建          | 7         |
| 3.1.1 硬件配置                   | 7         |
| 3.1.2 系统版本                   | 7         |
| 3.1.3 网络环境                   | 8         |
| 3.1.4 软件包                    | 8         |
| 3.2 Windows PC 环境搭建          | 9         |
| 3.2.1 开发工具安装                 | 9         |
| 3.2.2 开发板驱动安装                | 9         |
| 3.2.3 烧录软件安装                 | 10        |
| 3.3 开发板介绍                    | 10        |
| 3.3.1 使用准备                   | 12        |
| 3.3.2 开发板供电                  | 12        |
| 3.3.3 串口连接                   | 12        |
| 3.3.4 USB 调试连接               | 12        |
| <b>4 编译代码和打包固件</b>           | <b>13</b> |
| 4.1 编译基础                     | 13        |
| 4.1.1 基本编译命令                 | 13        |
| 4.1.2 编译选项                   | 13        |
| 4.1.3 扩展编译命令                 | 14        |
| 4.2 编译示例                     | 15        |
| 4.2.1 EMMC 板型全编译过程           | 15        |
| 4.2.2 SPI NAND 板型全编译过程       | 16        |
| 4.2.3 SPI NOR 编译             | 16        |
| 4.2.4 T113 Qt 编译详解           | 17        |
| 4.2.5 工具链简介                  | 18        |
| 4.2.5.1 Kernel 工具链           | 18        |

|                        |           |
|------------------------|-----------|
| 4.2.5.2 Buildroot 工具链  | 18        |
| <b>5 固件烧写</b>          | <b>19</b> |
| 5.1 USB 烧录             | 19        |
| 5.1.1 运行 PhoenixSuit   | 19        |
| 5.1.2 连接设备             | 19        |
| 5.1.3 选择 img 文件        | 20        |
| 5.1.4 开始烧录             | 20        |
| 5.2 SD 卡烧录             | 21        |
| 5.2.1 制作升级卡            | 21        |
| 5.2.2 插入平台上电升级         | 22        |
| 5.2.3 制作启动卡            | 22        |
| <b>6 系统调试</b>          | <b>23</b> |
| 6.1 串口调试               | 23        |
| 6.2 ADB 调试             | 24        |
| 6.2.1 adb 简介           | 24        |
| 6.2.2 运行 ADB           | 24        |
| 6.2.3 adb 常用命令         | 25        |
| <b>7 常见问题</b>          | <b>26</b> |
| 7.1 uboot 编译报错         | 26        |
| 7.2 adb 概率性断开          | 26        |
| 7.3 编译 QT 后，镜像中没有      | 27        |
| 7.4 插入 U 盘没有自动挂载       | 27        |
| 7.5 SPI NAND 硬浮点工具链问题  | 27        |
| 7.5.1 通过 menuconfig 配置 | 27        |
| 7.5.2 写入最简配置文件         | 28        |
| 7.5.3 验证硬浮点工具链         | 28        |
| 7.6 部分编译命令不支持          | 29        |
| 7.7 串口无打印、输入无反应等       | 29        |
| <b>8 附录</b>            | <b>30</b> |
| 8.1 在线帮助文档             | 30        |

## 插图

|                              |    |
|------------------------------|----|
| 3-1 交叉编译环境 . . . . .         | 7  |
| 3-2 开发板驱动 . . . . .          | 9  |
| 3-3 Win10 ADB 设备截图 . . . . . | 10 |
| 3-4 开发板介绍 . . . . .          | 11 |
| 3-5 串口接口 . . . . .           | 12 |
| 5-1 连接设备 . . . . .           | 19 |
| 5-2 选择固件 . . . . .           | 20 |
| 5-3 开始烧录 . . . . .           | 20 |
| 5-4 制作升级卡 . . . . .          | 21 |
| 5-5 SD 卡槽 . . . . .          | 22 |
| 6-1 串口连接 . . . . .           | 23 |

# 1 前言

## 1.1 文档简介

本文档用于介绍全志科技 T113 芯片的 Longan linux SDK。

Longan 是 linux SDK 开发包，它集成了 BSP、构建系统、linux 应用、测试系统、独立 IP、工具和文档，既可作为 BSP、IP 的开发、验证和发布平台，也可作为嵌入式 Linux 系统。



说明  
请重点关注环境配置编译章节，熟悉 **T113 SDK** 编译、**T113 Qt** 编译的过程。

## 1.2 目标读者

T113 平台开发人员。

## 1.3 适用范围

本文档适用于 V0.8 版本 SDK，后续更新的 SDK 版本在编译配置选项、功能配置的细节上可能与 V0.8 版本 SDK 不同。

本文档适用于的 T113 开发板见“开发板介绍”章节。

## 1.4 相关术语

表 1-1: 术语介绍

| 术语          | 说明              |
|-------------|-----------------|
| Longan      | 全志提供的 Linux SDK |
| APST        | 全志量产软件中心        |
| PhoenixSuit | 全志平台 USB 烧录软件   |
| PhoenixCard | 全志平台卡烧录软件       |

## 2 目录结构

longan 主要由 brandy、buildroot、kernel、platform 组成。其中 brandy 包含 uboot2018，buildroot 负责 ARM 工具链、应用程序软件包、Linux 根文件系统生成；kernel 为 linux 内核；platform 是平台相关的库和 sdk 应用。

```
.
├── 3rd
├── brandy
├── build
├── buildroot
├── build.sh
├── device
├── kernel
├── platform
├── test
└── tools
```

### 2.1 brandy

brandy 目录下有 brandy2.0 版本，目前 T113 使用 brandy2.0 版本，其目录主要结构为：

```
brandy/brandy-2.0/
├── build.sh -> tools/build.sh
├── opensbi
├── tools
└── u-boot-2018
```

📖 说明

**PS：**默认的代码编译流程中会编译 **uboot**，用户修改 **uboot** 后无需额外编译命令，使用默认编译命令即可。

### 2.2 buildroot

使用的版本是 buildroot-201902，主要功能如下：

- 管理包之间的依赖关系
- 生成 ARM 工具链
- 制作根文件系统，可以包含 strace, directfb, oprofile 等非常丰富的应用软件和测试软件
- 生成最终用于烧写的固件包

目录主要结构如下：

```
buildroot/buildroot-201902/
├── arch
├── board
├── boot
├── build.sh
├── CHANGES
├── Config.in
├── configs
├── COPYING
├── DEVELOPERS
├── dl
├── docs
├── fs
├── linux
├── Makefile
├── package
├── README
├── scripts
├── support
├── system
├── toolchain
└── utils
```

其中 configs 目录里存放预定义好的配置文件，比如我们的 sun8iw20p1\_t113\_defconfig，dl 目录里存放已经下载好的软件包，scripts 目录里存放 buildroot 的编译脚本，mkcmd.sh，mkcommon.sh，mkrule 和 mksetup.sh 等。target 目录里存放用于生成根文件系统的一些规则文件，该目录对于代码和工具的集成非常重要。对于我们来说最为重要的是 package 目录，里面存放了将近 3000 个软件包的生成规则，我们可以在里面添加我们自己的软件包或者是中间件。更多关于 buildroot 的介绍，可以到 buildroot 的官方网站<http://buildroot.uclibc.org/>获取。

## 2.3 kernel

linux 内核源码目录。当前使用的内核版本是 linux5.4.61。

目录主要结构如下：

```
kernel/linux-5.4/
├── abi_gki_whitelist
├── android
├── arch
├── block
├── certs
├── COPYING
├── CREDITS
├── crypto
├── Documentation
├── drivers
├── fs
├── include
├── init
├── ipc
├── Kbuild
└── Kconfig
```



```
kernel
lib
LICENSES
MAINTAINERS
Makefile
mm
modules
net
README
README.md
rootfs_32bit.cpio.gz
rootfs.cpio.gz
rootfs_rv64.cpio.gz
samples
scripts
security
sound
tools
usr
virt
```

除了 modules 目录，以上目录结构跟标准的 linux 内核一致。modules 目录是我们用来存放没有跟内核的 menuconfig 集成的外部模块的地方。

## 2.4 platform

平台私有软件包目录。

```
platform/
├── apps
├── base
├── config
├── core
├── external
├── framework
└── tools
```

其中 rootfs 会在每次顶层执行 build.sh 的时候强制覆盖到 out 目录相应的 target 下（适用于 EMMC 板型，target 为机器的根文件系统目录）。

## 2.5 tools

编译打包工具，tools\_win 是 PC 端烧录等工具目录。

```
tools/
├── build
├── codecheck
├── pack
└── tools_win
```

## 2.6 device/config/chips/t113

芯片配置目录，包含多个板级配置，每个板级配置都有不同的 board.dts, sys\_config.fex 等配置文件。

主要内容如下：

|               |                                  |                       |
|---------------|----------------------------------|-----------------------|
| bin           | 打包时使用的启动文件                       |                       |
|               | boot0_nand_sun8iw20p1.bin        | nand用的boot0启动文件       |
|               | boot0_sdcard_sun8iw20p1.bin      | emmc用的boot0启动文件       |
|               | dsp0.bin                         |                       |
|               | fes1_sun8iw20p1.bin              | 烧录工具用的初始化文件           |
|               | optee_sun8iw20p1.bin             | optee                 |
|               | sboot_sun8iw20p1.bin             | 安全启动的bin，暂不支持         |
|               | u-boot-sun8iw20p1.bin            | uboot的bin             |
| boot-resource |                                  |                       |
|               | boot-resource                    |                       |
|               | bat                              |                       |
|               | bootlogo.bmp                     | 启动画面，优先级低于板级目录下logo图片 |
|               | fastbootlogo.bmp                 |                       |
|               | wavefile                         |                       |
|               | boot-resource.ini                |                       |
| configs       |                                  |                       |
|               | default                          | 正常情况下不生效              |
|               | evb1                             | 不支持                   |
|               | evb1_auto                        | 默认的EMMC板型             |
|               | board.dts -> linux-5.4/board.dts | 板级dts配置               |
|               | bsp                              |                       |
|               | BoardConfig.mk                   |                       |
|               | BoardConfig_nor.mk               |                       |
|               | bootlogo.bmp                     |                       |
|               | env.cfg                          |                       |
|               | env_nor.cfg                      |                       |
|               | sys_partition.fex                |                       |
|               | sys_partition_nor.fex            |                       |
|               | linux-5.4                        |                       |
|               | board.dts                        | 板级dts配置               |
|               | config-5.4                       |                       |
|               | longan                           |                       |
|               | BoardConfig.mk                   | 内核，buildroot，工具链等配置   |
|               | BoardConfig_nor.mk               |                       |
|               | bootlogo.bmp                     | EMMC板型bootlogo图片      |
|               | env.cfg                          | EMMC板型环境变量            |
|               | env_nor.cfg                      |                       |
|               | sys_partition.fex                |                       |
|               | sys_partition-recovery.fex       | EMMC板型默认分区文件          |
|               | sys_config.fex                   | EMMC板型sys_config配置    |
|               | uboot-board.dts                  | EMMC板型uboot使用的dts文件   |
|               | evb1_auto_nand                   | SPI NAND板级目录          |
|               | BoardConfig.mk                   |                       |
|               | board.dts -> linux-5.4/board.dts |                       |
|               | bsp                              |                       |
|               | bootlogo.bmp                     |                       |
|               | env.cfg                          |                       |
|               | sys_partition.fex                |                       |
|               | env.cfg                          |                       |
|               | linux-5.4                        |                       |
|               | board.dts                        | SPI NAND板级dts配置       |



## 3 开发环境配置

本章主要介绍了如何在本地搭建编译环境来编译 Longan SDK 源代码。目前 SDK 只支持在 linux 环境下编译 Ubuntu 14.04(64 bit)。一个典型的嵌入式开发环境通常包括 linux 服务器、Windows PC 和目标硬件板。linux 服务器上建立交叉编译开发环境，为软件开发提供代码更新下载，代码交叉编译服务。

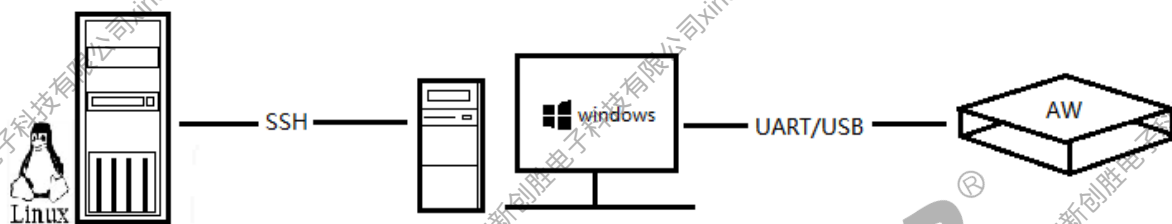


图 3-1: 交叉编译环境

Windows PC 和 Linux 服务器共享程序，Windows PC 并安装 SecureCRT 或 puTTY，通过网络远程登陆到 Linux 服务器，在 linux 服务器上进行交叉编译和代码的开发调试。Windows PC 通过串口和 USB 与目标开发板连接，可将编译后的镜像文件烧写到目标开发板，并调试系统或应用程序。

### 3.1 Linux 服务器开发环境搭建

linuxSDK 推荐的开发环境如下。

#### 3.1.1 硬件配置

推荐 64 位系统，硬盘空间大于 30G。如果您需要进行多个构建，请预留更大的硬盘空间。

#### 3.1.2 系统版本

本 SDK 开发环境安装如下版本 linux 系统，在默认情况下，SDK 均以此 linux 系统进行编译。Ubuntu 14.04.5 LTS Linux version 3.19.0-80-generic (buildd@lcy01-33) (gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04.3)) #88~14.04.1-Ubuntu SMP Fri Jan 13 14:54:07 UTC 2017



说明

注意：如用其他版本的 **linux**，请自行处理可能出现的软件包或环境设置问题。

### 3.1.3 网络环境

请自行安装安装 nfs、samba、ssh 等网络组件，并自行配置网络。

### 3.1.4 软件包

**警告**

下面的命令请手动安装，并确认每一个都成功，必须每个都成功。

除了 gcc, ncurses, bison, autoconf, wget, patch, texinfo, zlib, dos2unix 之外，还需要安装一些额外的软件包。配置好网络环境之后，则可以通过如下命令安装编译 SDK 需要的软件包：

```
sudo apt-get install git
sudo apt-get install gnupg
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install gperf
sudo apt-get install build-essential
sudo apt-get install zip
sudo apt-get install curl
sudo apt-get install libc6-dev
sudo apt-get install libncurses5-dev:i386
sudo apt-get install x11proto-core-dev
sudo apt-get install libx11-dev:i386
sudo apt-get install libreadline6-dev:i386
sudo apt-get install libgl1-mesa-glx:i386
sudo apt-get install libgl1-mesa-dev
sudo apt-get install g++-multilib
sudo apt-get install mingw32
sudo apt-get install tofrodos
sudo apt-get install python-markdown
sudo apt-get install libxml2-utils
sudo apt-get install xsltproc
sudo apt-get install zlib1g-dev:i386
sudo apt-get install gawk
sudo dpkg-reconfigure dash 选择no
sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

若编译遇到报错，请再根据报错信息，安装对应的软件包。

## 3.2 Windows PC 环境搭建

本节介绍 Windows PC 端需要的环境配置。

### 3.2.1 开发工具安装

请自行选择 SourceInsight, Notepad++, Qt Creator, VS code 等 IDE 或其它编辑软件, 以及 Xshell 或 puTTY 等串口通讯软件。

### 3.2.2 开发板驱动安装

一般在 Windows7 的环境下, 当目标板设备上电并插上 USB 线之后, 会自动安装 USB 设备驱动程序。如果安装成功, 则会在 Windows 管理器中出现下图中红色椭圆形标识的设备 Android Phone。



图 3-2: 开发板驱动

有些电脑在设备上电并插上 USB 线之后, 自动安装 USB 设备驱动程序会失败。推荐使用驱动人生等软件, 自动检索安装驱动程序。

Windows10 系统一般是自带 ADB 驱动的, 如果无法识别到 ADB 设备, 请手动更新驱动程序 (使用 PhoenixSuit 目录下的 Drivers/ADB\_Driver 文件夹)

Windows10 系统下 T113 的 ADB 设备在设备管理器中截图如下:



图 3-3: Win10 ADB 设备截图

不同版本的 Windows10 系统（比如家庭版与专业版、不同版本号的差别）、不同的驱动下，**T113 的 ADB 的设备名可能不同**，测试 adb 功能正常即可。

### 3.2.3 烧录软件安装

请从 APST 平台下载最新版本的 PhoenixSuit、PhoenixCard 软件。当 sdk 编译打包后，就可以通过 PhoenixSuit 烧录，详细步骤将在后文介绍。

APST 下载方法：登录全志“一号通”平台，点击页面上面的“开发工具”，点击“Windows 工具下载”，安装下载的“.msi”安装包。

烧录软件下载：启动 APST 软件并登录（需要管理员权限），点击左侧的“全部”，查看可用工具并下载，点击“运行”启动工具软件。

如果终端客户没有 APST 权限，请找代理商导出所需的最新版工具的压缩包，可免安装使用。

首次运行 APST 时，会检测当前机器是否包含全志驱动程序，如果驱动程序不存在，会自动进行驱动程序的安装。（USB 烧录驱动，不同于 ADB 驱动）

**(PS: 请注意 T113 要使用新的 PhoenixSuit 和 PhoenixCard，否则可能会导致无法进行 USB 烧录、无法卡升级等问题。最新版本的工具请使用 APST 下载)**

## 3.3 开发板介绍

AW T113evb1\_auto 开发板如下，板上丝印为：T113\_EVB\_V1\_0。



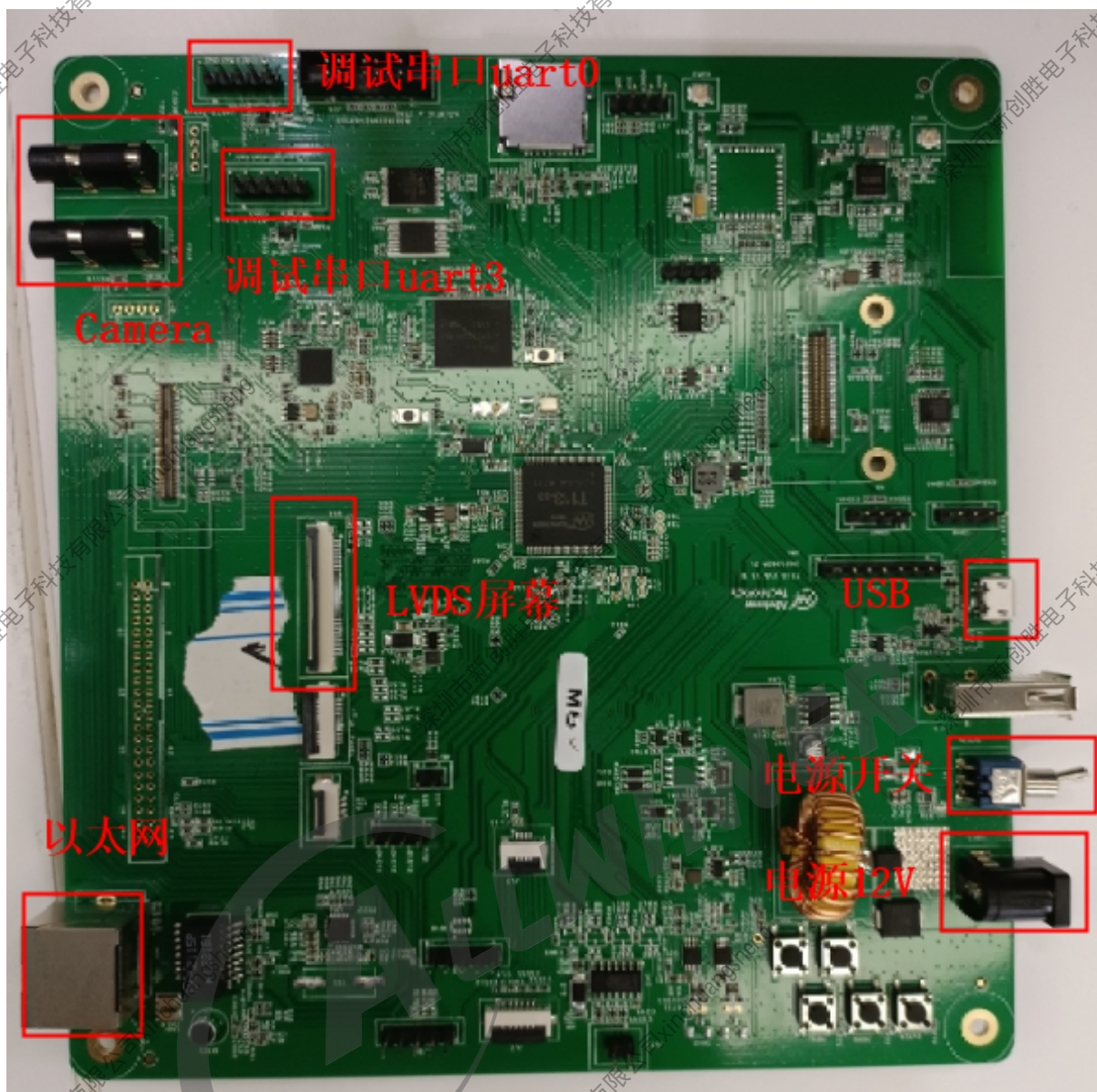


图 3-4: 开发板介绍

本手册重点关注 DCIN12V（连接 12V 直流电源），SOC 调试串口（连接串口通讯），USB-OTG micro 端口（用于烧录和 ADB）。

此开发板上接口丰富，可以通过连接不同的电阻来配置不同功能，使用某一功能前需要先确定开发板硬件连接正确，具体细节请查看随 SDK 发布的文档《T113\_Linux\_配置指南.pdf》。

开发板支持 EMMC、SPI NAND、SPI NOR 三种 flash，也支持 TF 卡启动。默认情况下连接的是 EMMC，使用 SPI NAND 或者 SPI NOR 需要修改硬件，并按下文“编译示例”章节内容编译固件。



### 3.3.1 使用准备

请检查串口硬件工具以及串口连接线、12V 直流电源、以及 microUSB 线等是否就绪。

### 3.3.2 开发板供电

请使用 12V 直流电源为开发板供电，供电电流推荐 2A 左右。

### 3.3.3 串口连接

默认的调试串口用的是 uart3，数字电平为 3.3v，连接如下图：（板上丝印有误，实际为 uart3，连接到 PG8&PG9）

使用此调试串口需要硬件上连接 R844、R847。

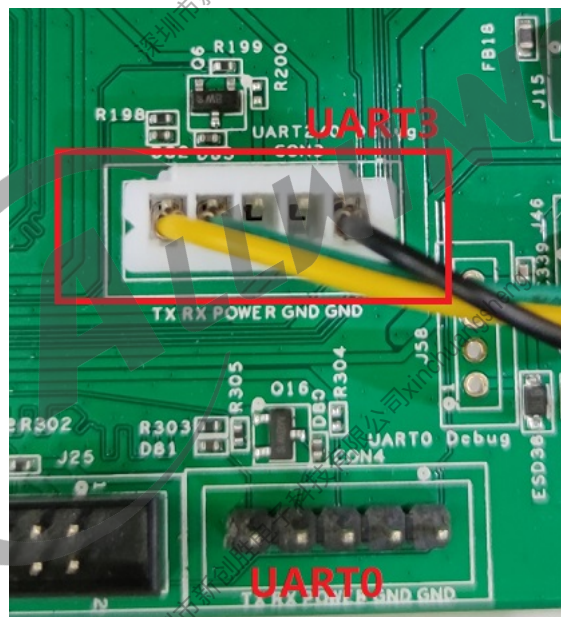


图 3-5: 串口接口

### 3.3.4 USB 调试连接

请使用 USB Micro 数据线，连接开发板的 micro USB 口和 windows PC 的 USB 端口。

## 4 编译代码和打包固件

本章介绍全编译和部分编译的详细步骤。编译完成后，通过打包，生成最终的 img。

### 4.1 编译基础

#### 4.1.1 基本编译命令

进入 longan 顶级目录，执行如下命令即可。

初次进行 SDK 环境配置需要进行多个选项的选择，为避免选错，建议使用 ./build.sh autoconfig 功能，详情见下文“编译示例”章节。

```
source build/envsetup.sh      #进行sdk环境配置（重要）每次开新的shell都做此操作，再进行开发
./build.sh                    #编译整个sdk
./build.sh pack                #打包固件

或者使用传统命令如下（不建议）：
./build.sh config              #进行sdk环境配置
./build.sh                     #编译整个sdk
./build.sh pack                #打包固件
```

#### 4.1.2 编译选项

只有执行了 source build/envsetup.sh 命令进行了编译配置，如下命令才能生效。

表 4-1：编译命令说明

| 类别   | 命令                    | 说明                    |
|------|-----------------------|-----------------------|
| 整体编译 | ./build.sh config     | 编译配置，弹出编译选择           |
|      | ./build.sh autoconfig | 根据传入的参数进行编译配置，不弹出编译选择 |
|      | ./build.sh            | 根据编译配置，编译 SDK         |
|      | ./build.sh clean      | 清除过程文件和目标文件           |
|      | ./build.sh distclean  | 清除所有生成的文件             |
| 局部编译 | ./build.sh brandy     | 编译 brandy             |
|      | ./build.sh kernel     | 编译 kernel             |
|      | ./build.sh buildroot  | 编译 buildroot          |
|      | ./build.sh qt         | 编译 qt                 |

| 类别 | 命令                           | 说明                            |
|----|------------------------------|-------------------------------|
| 打包 | ./build.sh dragonboard       | 编译 dragonboard                |
|    | ./build.sh sata              | 编译 sata                       |
|    | ./build.sh pack              | 打包命令，调试串口为 uart0              |
|    | ./build.sh pack_debug        | 打包命令，调试串口为 card0              |
|    | ./build.sh pack_secure       | 打包命令，生成 secure 固件，调试串口为 uart0 |
|    | ./build.sh pack_debug_secure | 打包命令，生成 secure 固件，调试串口为 card0 |

### 4.1.3 扩展编译命令

执行 source build/envsetup.sh 命令进行了编译配置，如下命令生效。

表 4-2: 扩展编译命令说明

| 类别   | 扩展命令                     | 说明   |
|------|--------------------------|--|
| 整体编译 | source build/envsetup.sh | 编译配置，弹出编译选择，将扩展命令导出到环境变量中                            |
|      | build                    | 根据编译配置，编译 SDK  |
|      | build clean              | 清除过程文件和目标文件  |
|      | build distclean          | 清除所有生成的文件  |
| 局部编译 | build brandy             | 编译 brandy  |
|      | build kernel             | 编译 kernel  |
|      | build buildroot          | 编译 buildroot   |
|      | build qt                 | 编译 qt  |
|      | build dragonboard        | 编译 dragonboard                                       |
|      | build sata               | 编译 sata  |
|      | build                    | 编译   |
| 打包   | pack                     | 打包命令，调试串口为 uart0                                     |
|      | pack -d                  | 打包命令，调试串口为 card0                                     |
|      | pack -s                  | 打包命令，生成 secure 固件，调试串口为 uart0                        |
|      | pack -sd                 | 打包命令，生成 secure 固件，调试串口为 card0                        |
| 调试   | build help               | 打印各个命令的使用方法  |
|      | build printconfig        | 打印编译使用到的全局变量   |
| 文件跳转 | cbrandy                  | 跳转到 brandy/brandy-xxx 目录                             |
|      | cbr                      | 跳转到 buildroot/buildroot-xxx 目录                       |
|      | cconfigs                 | 跳转到 device/config/chips/{chip_id}/configs/{board} 目录 |
|      | cdevice                  | 跳转到 device/target/{product}/{board} 目录               |
|      | cdts                     | 跳转到 dts 目录   |
|      | ckernel                  | 跳转到 kernel/linux-xxx 目录                              |
|      | cout                     | 跳转到 out/{chip_id}/{board}/{platform} 目录              |
|      | croot                    | 跳转到 longan 顶级目录                                      |

## 4.2 编译示例

建议的编译步骤如下：（最简编译过程，EMMC 板型默认配置）

```
./build.sh autoconfig -c linux -o longan -k linux-5.4 -i t113 -b evb1_auto -n default
#使用autoconfig进行配置，避免选错
source build/envsetup.sh      #设置环境变量，可使用快速跳转命令、简化编译命令
./build.sh      #一键编译
./build.sh pack   #打包生成固件
```

最终生成的 img 参考：out/t113\_linux\_evb1\_auto\_uart0.img。（文件名与调试串口不符，实际调试串口默认是 UART3）

### 4.2.1 EMMC 板型全编译过程

按如下步骤进行编译，可将 qt、cedar 编进固件中。

```
./build.sh autoconfig -c linux -o longan -k linux-5.4 -i t113 -b evb1_auto -n default
source build/envsetup.sh
# build all
./build.sh
# build cedar
croot
cd platform/framework/auto/buildcedar
./T113_cedar_compile.sh
# buld qt
croot
./build.sh qt
cd platform/framework/auto/qt_demo
./build.sh
croot
./build.sh
# pack image
croot
./build.sh pack
```

按上面步骤编译出的固件，路径默认为：out/t113\_linux\_evb1\_auto\_uart0.img。使用 EMMC flash，调试串口在 UART3，可用功能有：LVDS LCD、录像、CSI camera、USB camera、U 盘、TF 卡、RTP、AudioCodec。默认是 DVR 场景。

其余功能与上述功能用到的 IO 口存在冲突，需要单独配置，配置方法请查看随 SDK 发布的文档《T113\_Linux\_配置指南.pdf》

如果需要烧号固件，请修改 device/config/chips/t113/configs/evb1\_auto/sys\_config.fex 中 burn\_key = 1 后重新执行如下命令编译打包：

```
./build.sh
./build.sh pack
```

如果需要安全固件，首先执行正常编译步骤，然后进行如下操作：

```
source build/envsetup.sh
cd build
echo -e "\n" | ./createkeys
croot
./build.sh bootloader
pack -s
```

生成安全固件路径：out/t113\_linux\_evb1\_auto\_uart0\_secure\_v0.img。

## 4.2.2 SPI NAND 板型全编译过程

SPI NAND 板型编译过程如下：

```
./build.sh autoconfig -c linux -o longan -k linux-5.4 -i t113 -b evb1_auto_nand -n default
source build/envsetup.sh
./build.sh
./build.sh pack
```

使用 SPI NAND 需要硬件修改 flash 选择的电阻。按上面步骤编译出的固件，路径默认为：out/t113\_linux\_evb1\_auto\_nand\_uart0.img。使用 SPI NAND flash，调试串口在 UART3，默认可用的功能有：LVDS LCD、TF 卡、以太网。

使用以太网需要硬件焊接 25M 外部时钟给 ethernet phy。

其余功能与上述功能用到的 IO 口存在冲突，需要单独配置，配置方法请查看随 SDK 发布的文档《T113\_Linux\_配置指南.pdf》。

## 4.2.3 SPINOR 编译

V0.8 版本 SDK 中，SPI NOR 的板级目录目前与 EMMC 的相同，存在部分文件共用的情况，而 EMMC 板型默认使用 UART3，所以在编译之前需要进行如下修改：

仓库路径：device/config/chips/t113

```
diff --git a/configs/evb1_auto/linux-5.4/board.dts b/configs/evb1_auto/linux-5.4/board.dts
index 0b55f03..2db4bb7 100644
--- a/configs/evb1_auto/linux-5.4/board.dts
+++ b/configs/evb1_auto/linux-5.4/board.dts
@@ -520,7 +520,7 @@
     pinctrl-names = "default", "sleep";
     pinctrl-0 = <uart0_pins_a>;
     pinctrl-1 = <uart0_pins_b>;
-    status = "disabled";
+    status = "okay";
 };

 &uart1 {
@@ -541,7 +541,7 @@
     pinctrl-names = "default", "sleep";
     pinctrl-0 = <uart3_pins_a>;
     pinctrl-1 = <uart3_pins_b>;
```

```

-       status = "okay";
+       status = "disabled";
};

&sdrc2 {
diff --git a/configs/evb1_auto/sys_config.fex b/configs/evb1_auto/sys_config.fex
index c146e8a..3aba0af 100755
--- a/configs/evb1_auto/sys_config.fex
+++ b/configs/evb1_auto/sys_config.fex
@@ -208,9 +208,9 @@ twi_sda = port:PC1<3><1><default><default>
;uart_debug_rx |Boot串口接收的GPIO配置 |
;-----
[uart_para]
-uart_debug_port = 3
-uart_debug_tx   = port:PG08<5><1><default><default>
-uart_debug_rx   = port:PG09<5><1><default><default>
+uart_debug_port = 0
+uart_debug_tx   = port:PF02<3><1><default><default>
+uart_debug_rx   = port:PF04<3><1><default><default>
;-----

```

以上修改为：修改 board.dts，修改 uart0 状态为“okay”，uart3 状态改为“disabled”。修改 sys\_config.fex，调试串口改到 uart0。

修改完成后使用如下命令配置、编译：

```

./build.sh autoconfig -c linux -o bsp -k linux-5.4 -i t113 -b evb1_auto -n nor
source build/envsetup.sh
./build.sh
./build.sh pack

```

使用 SPI NOR 需要硬件修改 flash 选择的电阻。按上面步骤编译出的固件，路径默认为：out/t113\_linux\_evb1\_auto\_uart0\_nor.img。使用 SPI NOR flash，调试串口在 UART0，默认可用的功能有：LVDS LCD 等。

#### 4.2.4 T113 Qt 编译详解

Qt 源代码位于：platform/framework/qt/qt-everywhere-src-5.12.5。其下有两个脚本：

```

buildsetup_t113.sh 设置Qt编译环境参数，
qtenv.sh           设置Qt目标平台运行环境参数
当执行build qt选项时，会配置buildsetup.sh：
platform/framework/qt/qt-everywhere-src-5.12.5/buildsetup_t113.sh

***** useage *****
please use:
qtmakeconfig:      config qt env.
qtmakeall:         build qt
qtmakeinstall:     install qt-lib and cp to target dir.
qtmakecleanall:    clean qt and rm all target.

```

如果单独编译，可以在 qt 目录下 source 该脚本，按上述命令执行即可。当执行 qtmakeinstall 时，重点会执行以下三个步骤：



1. 将目标文件安装到 platform/framework/qt/qt-everywhere-src-5.12.5/Qt\_5\_12\_5 目录，
2. 将运行时所需的 Qt 库文件拷贝到 out 目录。将 qtenv.sh 文件拷贝到 quto/rootfs/etc/，由 rcS 上电时执行。
3. 脚本会重新再次执行 build，重新编译，以加入 qt 库，供打包程序使用。

编译 Qt 库之后，再运行一次 build.sh，然后重新打包，就可以将 Qt 库文件打包进 img 了。

```
./build.sh qt  
./build.sh  
./build.sh pack
```

## 4.2.5 工具链简介

### 4.2.5.1 Kernel 工具链

Kernel 工具链位于：

build/toolchain/gcc-linaro-5.3.1-2016.05-x86\_64\_arm-linux-gnueabi.tar.xz

T113 是 32 位 IC，采用该工具链来编译内核。

### 4.2.5.2 Buildroot 工具链

Buildroot 工具链位于：

buildroot/buildroot-201902/dl/toolchain-external-linaro-armsf/gcc-linaro-7.3.1-2018.05-x86\_64\_arm-linux-gnueabi.tar.xz

在 buildroot 的 defconfig 中有相关定义。

如果使用硬浮点，则工具链为：

buildroot\buildroot-201902\dl\toolchain-external-linaro-arm\gcc-linaro-7.3.1-2018.05-x86\_64\_arm-linux-gnueabihf.tar.xz

需要硬浮点工具链，可以在 ./build.sh config 的最后一步选择“gnueabihf”，注意目前默认只有 **EMMC 板型支持硬浮点工具链**。此工具链是 buildroot 的，对 rootfs 生效。如果 SPI NAND 板型需要使用硬浮点工具链，请参考后文“硬浮点工具链问题”章节。

## 5 固件烧写

本章介绍如何将编译好的固件，烧写到开发板的步骤。烧录软件前文已经介绍安装方式。

### 5.1 USB 烧录

这种烧录方式方便开发人员进行软件的开发以及调试，具体步骤如下。

#### 5.1.1 运行 PhoenixSuit

启动 PhoenixSuit，目前最新版本为 1.19，旧版本工具可能不支持 T113。

#### 5.1.2 连接设备

开发板上电开机，用 microUSB 线连接到电脑，查看是否检测到设备。检测到设备之后的界面如下，下方会提示设备连接成功：



图 5-1: 连接设备

**注意：**如未检测到设备，可能驱动未正常安装。可以使用 **PhoenixSuit** 安装目录、**APST** 安装目录、**SDK** 的 **tools/tools\_win** 等目录下的驱动手动安装。



### 5.1.3 选择 img 文件

点击上方【一键刷机】图标，选择编译生成的 img 文件：



图 5-2: 选择固件

**注意：V0.8 版本的 SDK 已经支持 PhoenixSuit“立即升级”按钮一键烧录。前提是 ADB 连接正常。**

### 5.1.4 开始烧录

关闭 T113 开发板电源，待所有的指示灯都灭干净。按住机器的 FEL 按键，重新给 T113 供电。或者直接按住 FEL 键，再按一下 reset。或者在重启/上电时串口不断发送 2（在串口软件按住键盘的 2 不放），也可进入烧录模式。此外，输入 reboot efex 命令也可以进入烧录模式。

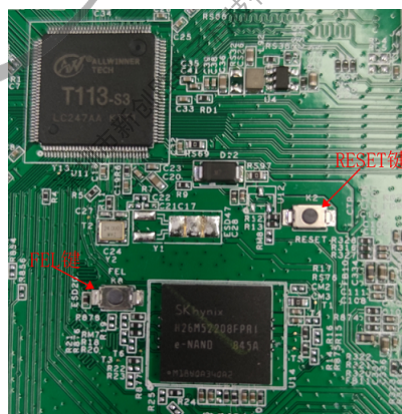


图 5-3: 开始烧录

**注意：在开始菜单，或者安装目录，可以找到《PhoenixSuit 用户指南.pdf》，里面介绍了烧录软件的整个流程。更具体的信息请参考该文档。**

请等待烧录软件提示烧录成功后再断开电源或者 USB 线，否则会导致固件烧录不完整，系统无法启动。出现烧录失败问题请检查固件与板子硬件上连接的 flash 是否匹配。

## 5.2 SD 卡烧录

此种方式常用于量产或售后软件升级（注：**V0.8 版本 SDK 的 EMMC 板型已支持卡启动、卡量产，前提是当前配置 sdc0 接口的 IO 无冲突**）

### 5.2.1 制作升级卡

从 APST 下载并打开 PhoenixCard 软件（目前最新版本为 4.2.7），并安装 APST 中提供的“VS Runtime Collection”，制作 sd 升级卡的相关信息可以点 PhoenixCard 软件的“帮助”查看。参考下图，首先选择固件，然后选择“量产卡”，再选择盘符，最后点击“烧卡”：

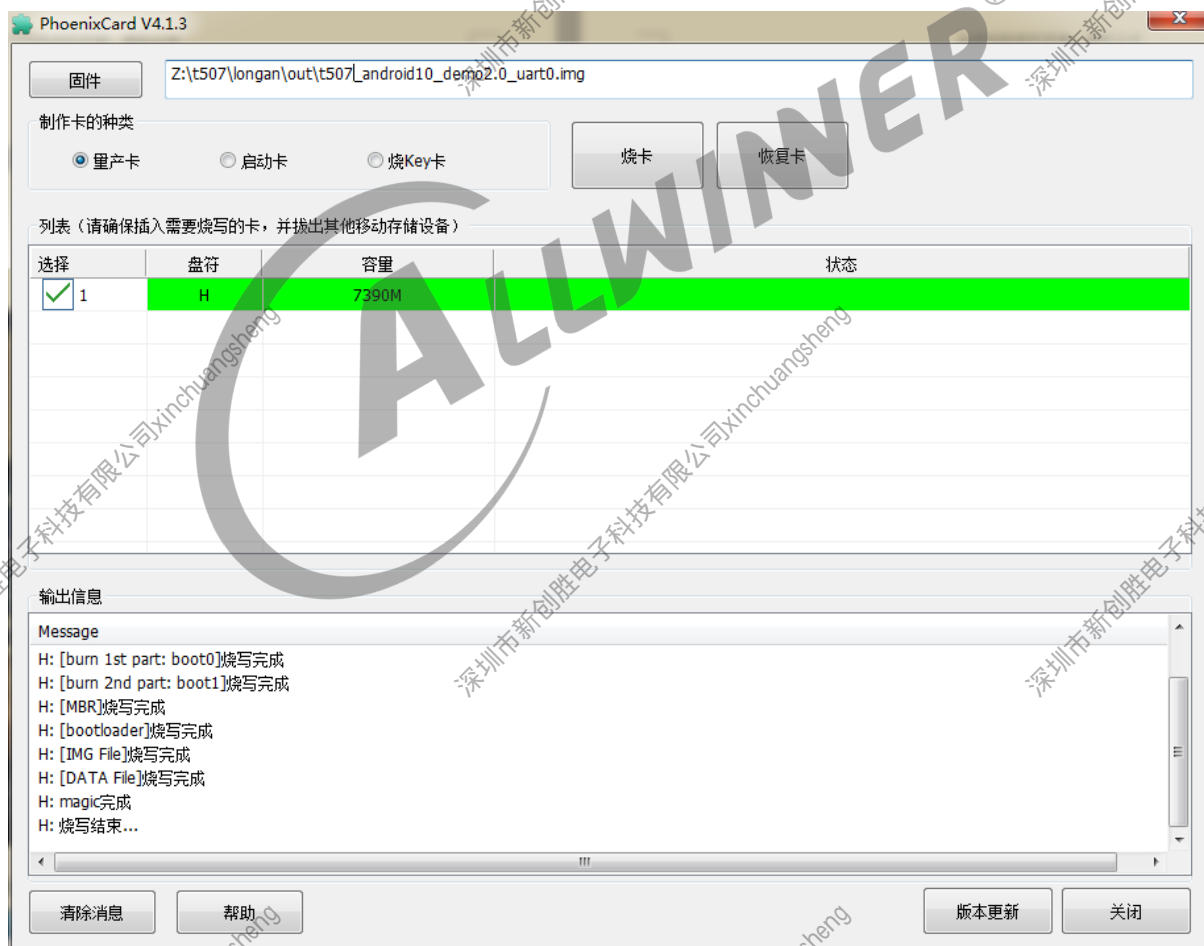


图 5-4: 制作升级卡

如果卡里面已经有多个分区，第一次操作可能会失败，重新点击“烧卡”即可，如果多次烧卡失败，请检查是否安装了“VS Runtime Collection”。

(注意：若是 v4.2 以下的软件，则有可能无法进行正常的卡升级/卡烧录动作。)

## 5.2.2 插入平台上电升级

卡烧录好后，插入机器 TF 卡槽，如下图：

(PhoenixCard 软件执行完后，Windows 系统可能会提示无法识别磁盘询问是否格式化，请不要做格式化操作，因为 Linux 的分区 Windows 系统不能识别，如果被 Windows 系统格式化则 PhoenixCard 软件写入数据会全部丢失，这个卡也就白做了)

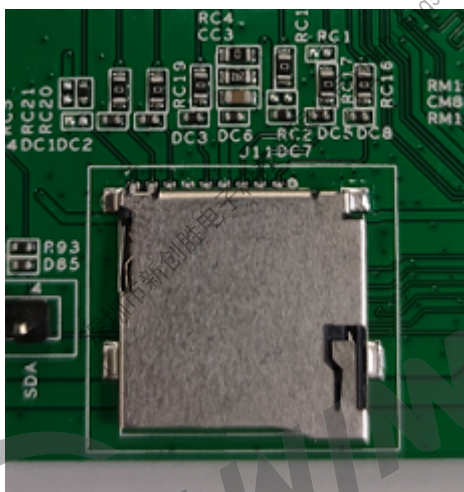


图 5-5: SD 卡槽

重新上电，机器就会自动升级了。可以看到屏幕上有进度条，调试串口有相应输出。整个过程大概 1~2 分钟，具体视固件大小而定。升级完毕后，串口会打印提示信息，拔掉 TF 卡，然后重新上电。

## 5.2.3 制作启动卡

打开 PhoenixCard 软件，首先选择固件，然后选择“启动卡”，再选择盘符，最后点击“烧卡”，如果卡里面已经有多个分区，第一次操作可能会失败，重新点击“烧卡”即可，如果多次烧卡失败，请检查是否安装了“VS Runtime Collection”。

使用 PhoenixCard 软件制作的量产卡或者启动卡，里面有多个分区，而且部分分区不能被 Windows 系统所识别，文件浏览器中看到的容量会偏小，如果要恢复作为存储卡，使用 PhoenixCard 软件的“恢复卡”功能进行格式化即可，卡会被格式化为单分区的，容量也会正常显示。也可使用 Windows 自带的磁盘分区工具进行分区删除操作，只是略微繁琐。

## 6 系统调试

支持串口和 adb 方式来和 windows PC 通讯。

### 6.1 串口调试

通过 windows PC 端串口通讯工具，连接开发板串口。配置参数：波特率：115200，数据位：8，奇偶校验位：无，停止位：1。

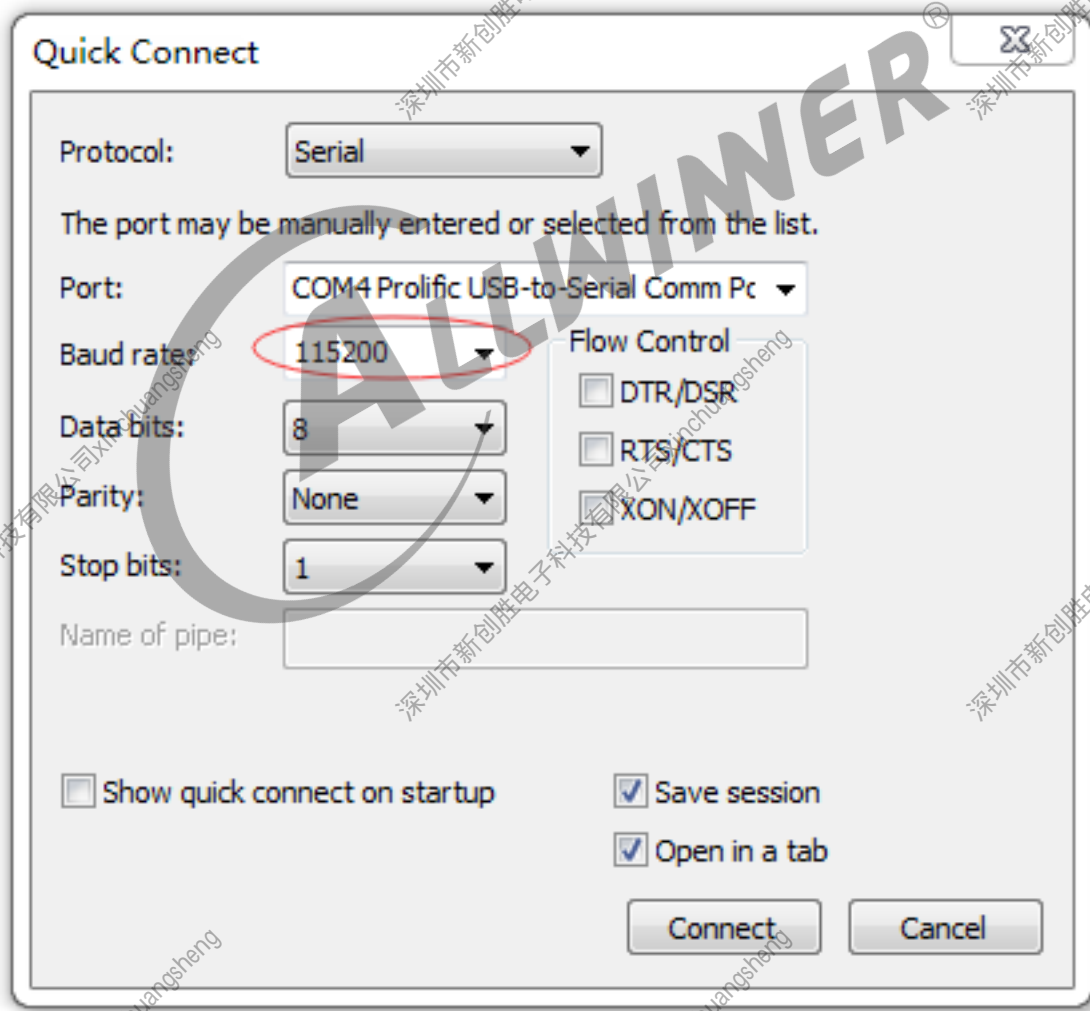


图 6-1：串口连接

## 6.2 ADB 调试

### 6.2.1 adb 简介

adb 全称为 Android Debug Bridge，是 Android SDK 里的一个工具，用于操作管理 Android 模拟器或真实的 Android 设备。其主要功能有：

- 运行设备的 shell（命令行）
- 管理模拟器或设备的端口映射
- 在计算机和设备之间上传/下载文件

### 6.2.2 运行 ADB

Windows PC 端的 adb 使用方法和 adb 应用程序，请自行从网络搜索。上电会自动加载 adb 脚本，如果有问题，请手动运行/etc/adb\_conf.sh。

```
# /etc/adb_conf.sh
Starting adb addb: /usr/lib/libcrypto.so.1.1: no version information available (required by
addb)
install_listener('tcp:5037','*smartsocket*')
cannot bind 'tcp:5037'
find: ./proc/21747: No such file or directory
[ 4374.153373]
[ 4374.153373] rmmod_device_driver
[ 4374.153373]
[ 4374.160259] rmmod_device_driver()223 WARN: get power supply failed
[ 4374.167398] android_work: sent uevent USB_STATE=DISCONNECTED
[ 4374.168429]
[ 4374.168429] insmod_device_driver
[ 4374.168429]
device_chose finished!
# [ 4374.336191] sunxi_set_cur_vol_work()397 WARN: get power supply failed
[ 4374.423310] android_work: sent uevent USB_STATE=CONNECTED
[ 4374.437568] ERR: dev->driver->setup failed. (-22)
[ 4374.443118] ERR: dev->driver->setup failed. (-22)
[ 4375.129101] configfs-gadget gadget: high-speed config #1: c
[ 4375.135873] regulator-dummy: Underflow of regulator enable count
[ 4375.135925] android_work: sent uevent USB_STATE=DISCONNECTED
[ 4375.143311] read descriptors
[ 4375.152348] read strings
[ 4375.443891] sunxi_set_cur_vol_work()397 WARN: get power supply failed
[ 4375.530285] android_work: sent uevent USB_STATE=CONNECTED
[ 4375.560081] configfs-gadget gadget: high-speed config #1: c
[ 4375.566737] android_work: sent uevent USB_STATE=CONFIGURED
```

如果出现 android\_work: sent uevent USB\_STATE=CONNECTED，则说明已经可以了，如果没有出现，请再次执行/etc/adb\_start.sh 并检查接线是否正常。这样就可以在 Windows PC 直接通过 adb 来更新平台的应用程序或者库文件，不用重新烧录固件。

## 6.2.3 adb 常用命令

- pc 端查看当前连接的设备

```
adb devices
```

- PC 端进入设备 shell 命令行模式

```
adb shell
```

- 将电脑上的文件上传至设备

```
adb push <local path> <remote path>
```

- 下载设备里的文件到电脑

```
adb pull <remote path> <local path>
```



## 7 常见问题

本章主要介绍平台环境搭建的常见问题及解决办法。

### 7.1 uboot 编译报错

如果出现如：

```
brandy/brandy-2.0/spl/include/config.h:27:13: error: missing whitespace after the macro
name [-Werror]
#define CFG_本_dts 3
```

那应该是 device/config/chips/t113/configs 下是否出现中文名字的文件，如果出现则会有这样的报错。解决方法是去掉中文文件名。

### 7.2 adb 概率性断开

手动在串口终端执行一下/etc/adb\_conf.sh

```
# /etc/adb_conf.sh
Starting adb abdb: /usr/lib/libcrypto.so.1.1: no version information available (required by
adb)
install_listener('tcp:5037','*smartsocket')
cannot bind 'tcp:5037'
find: ./proc/21747: No such file or directory
[ 4374.153373]
[ 4374.153373] rmmmod_device_driver
[ 4374.153373]
[ 4374.160259] rmmmod_device_driver()223 WARN: get power supply failed
[ 4374.167398] android_work: sent uevent USB_STATE=DISCONNECTED
[ 4374.168429]
[ 4374.168429] insmod_device_driver
[ 4374.168429]
device_chose finished!
# [ 4374.336191] sunxi_set_cur_vol_work()397 WARN: get power supply failed
[ 4374.423310] android_work: sent uevent USB_STATE=CONNECTED
[ 4374.437568] ERR: dev->driver->setup failed. (-22)
[ 4374.443118] ERR: dev->driver->setup failed. (-22)
[ 4375.129101] configfs-gadget gadget: high-speed config #1: c
[ 4375.135873] regulator-dummy: Underflow of regulator enable count
[ 4375.135925] android_work: sent uevent USB_STATE=DISCONNECTED
[ 4375.143311] read descriptors
[ 4375.152348] read strings
[ 4375.443891] sunxi_set_cur_vol_work()397 WARN: get power supply failed
[ 4375.530285] android_work: sent uevent USB_STATE=CONNECTED
```

```
[ 4375.560081] configfs-gadget gadget: high-speed config #1: c  
[ 4375.566737] android_work: sent uevent USB_STATE=CONFIGURED
```

看到最后一句，就说明 adb 重新连上了，如果没有出现这一句，请检查硬件线路是否正确，或者更换较好的 usb 线。

## 7.3 编译 QT 后，镜像中没有

编译 QT 成功后需要重新 build 一次，然后打包，这样 QT 就会被打包到镜像中。

## 7.4 插入 U 盘没有自动挂载

V0.8 版本 SDK 的 EMMC 板型插入 U 盘会自动挂载，而 NAND 和 NOR 暂不支持，需要手动输入命令挂载：

```
ls /dev/sd* #插入U盘，查看设备节点  
mount /dev/sda /mnt #挂载到/mnt目录  
ls -al /mnt #查看U盘内容  
df -h #查看挂载情况
```

## 7.5 SPI NAND 硬浮点工具链问题

需要硬浮点工具链，可以在./build.sh config的最后一步选择“gnueabihf”，注意**目前只有 EMMC 板型默认支持硬浮点工具链**。此工具链是 buildroot 的，对 rootfs 生效。

如果需要在 SPI NAND 板型使用硬浮点工具链，请参考如下两种方法。

### 7.5.1 通过 menuconfig 配置

```
#执行如下命令前，先进入SDK根目录  
cd buildroot/buildroot-201902/configs  
cp sun8iw20p1_t113_nand_defconfig sun8iw20p1_t113_nand_hf_defconfig
```

修改 sun8iw20p1\_t113\_nand\_hf\_defconfig 的 154 行中BR2\_DEFCONFIG的定义，改为如下内容：

```
BR2_DEFCONFIG="${TOPDIR)/configs/sun8iw20p1_t113_nand_hf_defconfig"
```

然后返回 buildroot 根目录，重新生成配置文件，接着进行 menuconfig：

```
cd ..  
make ARCH=arm sun8iw20p1_t113_nand_hf_defconfig
```



menuconfig 配置方法：选择 Target options，选择 Target ABI，选择 EABIhf；回到最上层，选择 Toolchain，选择第三行的 Toolchain，选择 Linaro ARM 2018.05；然后 save，就是硬浮点的配置。

可以使用两种方法保存配置文件，最简保存和完全保存。

```
#最简保存，不包含未配置的软件包，在buildroot/buildroot-201902目录执行：
make ARCH=arm savedefconfig
#完全保存，将生成的配置文件copy为defconfig文件，在SDK根目录执行
cp out/t113/evb1_auto_nand/longan/buildroot/.config buildroot/buildroot-201902/configs/
sun8iw20p1_t113_nand_hf_defconfig
```

保存配置文件后，进行./build.sh config操作，在最后一步选择“gnueabihf”，并按下文“验证硬浮点工具链”章节，确认完全生效。

## 7.5.2 写入最简配置文件

以上为手动 menuconfig 配置方法，也可直接写入最简配置文件：

```
BR2_arm=y
BR2_cortex_a7=y
BR2_TOOLCHAIN_EXTERNAL=y
BR2_TOOLCHAIN_EXTERNAL_LINARO_ARM=y
BR2_TARGET_GENERIC_HOSTNAME="kunos"
BR2_TARGET_GENERIC_ISSUE="Welcome to Allwinner KunoOS Platform"
BR2_TARGET_GENERIC_GETTY_PORT="ttyS3"
BR2_TARGET_GENERIC_GETTY_BAUDRATE_115200=y
BR2_PACKAGE_LIBLDNS=y
BR2_PACKAGE_DHCPD=y
BR2_PACKAGE_ANDROID_TOOLS=y
```

将以上内容保存为buildroot/buildroot-201902/configs/sun8iw20p1\_t113\_nand\_hf\_defconfig，也可实现硬浮点工具链配置。

保存配置文件后，进行./build.sh config操作，在最后一步选择“gnueabihf”，并按下文“验证硬浮点工具链”章节，确认完全生效。

## 7.5.3 验证硬浮点工具链

首先删除整个 out 目录，清理之前的工具链生成的编译文件。

然后进行./build.sh config操作，在最后一步选择“gnueabihf”，在生成的配置文件out/t113/evb1\_auto\_nand/longan/buildroot/.config中有如下内容，说明配置硬浮点工具链成功：

```
...省略部分内容
BR2_GCC_TARGET_FLOAT_ABI="hard"
...省略部分内容
BR2_ARM_EABIHF=y
...省略部分内容
```

然后使用./build.sh完全重新编译 SDK，在编译 buildroot 过程中，有类似如下 log，表明使用了硬浮点工具链“arm-linux-gnueabi-hf-gcc”进行编译：

```
*****/out/t113/evb1_auto_nand/longan/buildroot/host/bin/arm-linux-gnueabi-hf-gcc -
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -Os -std=c99 -
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -DHAVE_CONFIG_H -
DNDEBUG -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE
-DINET -DARP -DARPING -DIPV4LL -DINET6 -DDHCP6 -DAUTH -I.. -I../src -I./crypt -c auth.c
-o auth.o
```

使用如下命令查看生成的 bin 文件是否是硬浮点的：

```
./out/t113/evb1_auto_nand/longan/buildroot/host/bin/arm-linux-gnueabi-hf-readelf -h out/t113
/evb1_auto_nand/longan/buildroot/build/dhccpd-7.0.3/src/dhccpd
```

输出信息有如下内容，表明 bin 文件是硬浮点的：

```
Flags: 0x5000400, Version5 EABI, hard-float ABI
```

## 7.6 部分编译命令不支持

可能是因为没有环境变量，需要先回到 SDK 根目录执行如下命令：

```
source build/envsetup.sh
```

## 7.7 串口无打印、输入无反应等

两个串口都无打印，一般是由于使用默认的软件和硬件，默认固件是 UART3 作为调试串口，而默认的硬件需要连接 R844、R847 才能把 UART3 的 IO 口连接到 CON3 座子上。这种情况硬件连接 R844、R847 即可。

另一种情况是自行修改了串口配置，比如要使用 WiFi/BT，需要把调试串口改回 UART0，这种情况需要修改板级目录、uboot defconfig 等多个文件，如果修改不彻底，就会出现打印 log 不完整、串口输入无响应等情况，请按照随 SDK 发布的文档《T113\_Linux\_配置指南.pdf》进行正确的配置。

## 8 附录

### 8.1 在线帮助文档

makefile 帮助文档：<http://www.gnu.org/software/make/manual/make.html>

buildroot 帮助文档：<http://buildroot.uclibc.org/downloads/buildroot.html>




## 著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。