T5 longan linux SDK

开发环境配置手册

V1.3

Revision History

Version	Date	Author	Changes compared to previous issue
V0.1	2019-12-17	吴松睿	初建版本
V0.2	2020-4-2	袁国超	添加编译、QT、工具链、测试等章节
V0.3	2020-5-9	xufang	补充 rootfs 工具说明
V1.0	2020-5-10	袁国超	添加 auto sdk 等章节
V1.1	2020-7-4	袁国超	添加显示配置
V1.2	2021-12-3	KPA0501	增加 T507-H/T517-H 的编译说明
V1.3	2022-04-28	KPA0501	更新-H 系列编译说明

目录

1. 概述.....6
2. 目录结构.....7
2.1. buildroot.....7
2.2. kernel.....8
2.3. brandy.....8
2.4. platform.....8
2.5. tools.....9
2.6. test 系统 dragonboard.....9
2.7. device.....10
3. 开发环境配置.....12
3.1. Linux 服务器开发环境搭建.....12
3.1.1. 硬件配置.....12
3.1.2. 系统版本.....12
3.1.3. 网络环境.....13
3.1.4. 软件包.....13
3.2. Windows PC 环境搭建.....13
3.2.1. 开发工具安装.....13
3.2.2. 开发板驱动安装.....14
3.2.3. 烧录软件安装.....14
3.3. 开发板介绍.....15
3.3.1. 使用准备.....16
3.3.2. 开发板供电.....16
3.3.3. 串口连接.....16
3.3.4. USB 调试连接.....17
3.3.5. 摄像头硬件连接.....17
4. 编译代码和打包固件.....18
4.1. 编译基础.....18
4.1.1. 基本编译命令.....18
4.1.2. 编译选项.....18
4.1.3. 扩展编译命令.....19
4.2. 编译示例.....20
4.2.1. 配置环境.....20
4.2.2. T5 auto 编译详解.....21
4.2.3. T5 Qt 编译详解.....22
4.3. 工具链简介.....22
4.3.1. Kernel 工具链.....23
4.3.2. Buildroot 工具链.....23
4.3.3. Dragonboard 工具链.....23
5. 固件烧写.....24
5.1. USB 烧录.....24
5.1.1. 运行 PhoenixSuit.....24
5.1.2. 连接设备.....24

5.1.3. 选择 img 文件.....25
5.1.4. 开始烧录.....25
5.2. SD 卡烧录.....27
5.2.1. 制作升级卡.....27
5.2.2. 插入平台上电升级.....27
6. 系统调试.....28
6.1. 串口登录命令行.....28
6.2. 连接成功后，在串口终端回车即可。.....28
6.3. 使用 adb 调试.....28
6.3.1. adb 简介.....28
6.3.2. 运行 adb.....29
6.3.3. ADB 常用命令.....29
6.4. GDB 调试工具.....30
7. Dragonboard 测试系统的使用.....31
8. auto sdk 简介.....33
8.1. ION.....34
8.2. 显示.....34
8.2.1. 接口简介.....35
8.2.2. 应用示例.....35
8.2.3. 调试信息.....36
8.2.4. 应用双屏配置.....37
8.2.5. 内核双屏配置.....37
8.3. 摄像头.....40
8.3.1. dvr_factory 类.....40
8.3.2. sdktest 的使用.....40
8.3.3. csitest.....41
8.4. 播放器.....41
8.4.1. xplayer.....41
8.4.2. AUTPlayer.....42
8.5. 视频编码.....42
8.6. 视频解码.....43
8.7. fbinit.....43
8.8. GPU.....43
8.9. qt_demo.....43
8.10. rootfs.....43
8.10.1. cpu_monitor.....44
8.10.2. mtop 监控 ddr 状态.....44
8.10.3. gpio 状态设置与查询.....45
8.10.4. 音频输入测试 tinycap_ahub_t5.....45
8.10.5. 音频播放测试 tinyplay.....46
9. 常见问题.....47
9.1. 修改 nand Flash.....47
9.1.1. 分区修改.....47
9.1.2. 存储类型.....47

9.2. 适配 DDR3.....47
9.3. DE invalid address.....48
9.4. 开机 Logo 盖住了摄像头画面.....50
9.5. adb 概率性断开.....50
10. 附录.....51
10.1. 在线帮助文档.....51
11. Declaration.....52

1.概述

本文档用于介绍全志科技 T507 芯片的 longan linuxSDK。

longan 是 linuxSDK 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。

请重点关注编译章节和 auto sdk 章节，熟悉 T5 auto 编译、T5 Qt 编译的过程。

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。

本章依次介绍代码各目录的内容和功能。

```
├─── brandy
├─── buildroot
├─── kernel
├─── platform
├─── test
└─── tools
```

2.1. buildroot

使用的版本是 buildroot-201902

- 管理包之间的依赖关系
- 生成 ARM 交叉工具链
- 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

目录结构如下

```
├─── arch
├─── board
├─── boot
├─── CHANGES
├─── Config.in
├─── configs
├─── COPYING
├─── dl
├─── docs
├─── external-packages
├─── fs
├─── linux
├─── Makefile
├─── support
├─── package
├─── README
├─── scripts
```

```
├──── target
├──── support
├──── system
├──── utils
└──── toolchain
```

其中 configs 目录里存放预定义好的配置文件，比如我们的 sun50iw9p1_longan_defconfig，dl 目录里存放已经下载好的软件包，scripts 目录里存放 buildroot 编译的脚本，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。

2.2. kernel

linux 内核源码目录。当前使用的内核版本是 linux4.9.170。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。

2.3. brandy

brandy 目录下有 brandy2.0 版本，目前 T507 使用 brandy2.0 版本，其目录结构为:

```
├──── u-boot-2018
└──── tools
```

PS:默认的代码编译流程中并不会编译 uboot，用户需要修改 uboot 的时候需要自己编译。具体编译方法后续有介绍。

2.4. platform

平台私有软件包目录。


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

其中，framework/auto 内包含了 T5 linux 版本的 SDK 接口和示例。

```
platform/framework/auto/
├── rootfs
├── sdk_demo
└── sdk_lib
```

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（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。

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

2.6. test 系统 dragonboard

test 是一个测试系统，名叫 dragonboard。dragonboard 提供快速的板级测试。编译输出相见后续的章节。

2.7. device

```
├── config
│   ├── chips
│   │   ├── t507        T507 方案/T507-H 方案
│   │   └── t517        T517 方案/T517-H 方案
│   ├── common
│   │   ├── debug
│   │   ├── dtb
│   │   ├── hdcp
│   │   ├── imagecfg
│   │   ├── partition
│   │   ├── sign_config
│   │   ├── toc
│   │   ├── tools
│   │   └── version
│   └── rootfs_tar
```

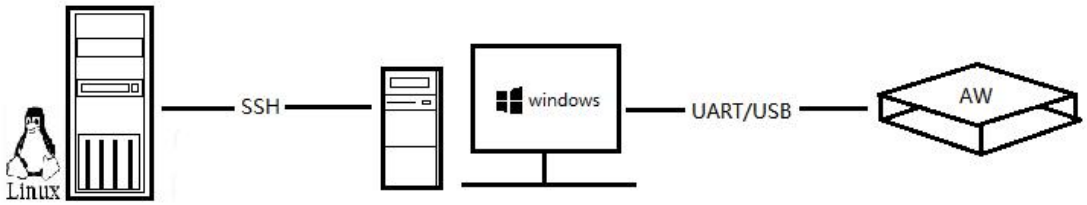
```
├── bin    打包时使用的启动文件
│   ├── bl31.bin
│   ├── boot0_nand_sun50iw9p1.bin    nand 用的 boot0 启动文件
│   ├── boot0_sdcard_sun50iw9p1.bin    emmc 用的 boot0 启动文件
│   ├── fes1_sun50iw9p1.bin  烧录工具用的初始化文件
│   ├── optee_sun50iw9p1.bin    optee
│   ├── sboot_sun50iw9p1.bin     安全启动的 bin，暂不支持
│   └── u-boot-sun50iw9p1.bin    uboot 的 bin
├── boot-resource
│   ├── boot-resource
│   │   ├── bat
│   │   └── bootlogo.bmp    启动画面，更新 bootlogo 可以替换此文件
│   └── boot-resource.ini
├── configs
│   ├── default  默认的平台性配置
│   │   ├── BoardConfig.mk
│   │   ├── boot_package.cfg    boot 引导的 boot 包内容配置
│   │   ├── diskfs.fex
│   │   ├── dragon_toc_android.cfg
│   │   ├── dragon_toc.cfg
│   │   ├── env_burn.cfg
│   │   ├── env.cfg      uboot 传给 linux 的启动参数
│   │   ├── env_dragon.cfg
│   │   ├── image.cfg
│   │   └── image_dragonboard.cfg
```

```
        ├── image_linux.cfg
        ├── image_longan.cfg      img 的配置项
        └── parameter.fex
    ├── demo2.0
        ├── board.dts  板级配置
        ├── dragonboard
            ├── BoardConfig.mk
            ├── sys_partition_dragonboard.fex
            └── test_config.fex
        ├── longan
            ├── BoardConfig.mk      内核，buildroot，工具链等配置
            └── sys_partition.fex      分区文件
        └── sys_config.fex      板级配置
    ├── evb
        ├── board.dts
        ├── dragonboard
            ├── BoardConfig.mk
            ├── sys_partition_dragonboard.fex
            └── test_config.fex
        ├── dtbo
        ├── longan
            ├── BoardConfig.mk
            └── sys_partition.fex
        └── sys_config.fex
    └── ver_v1_0    选 bsp 或 dragonboard 才支持 ver1 版本硬件 DDR16x2
        ├── android
            └── sys_partition.fex
        ├── board.dts
        └── sys_config.fex
└── tools
    ├── cardscript.fex
    ├── readme.txt
    └── usbtool_crash.fex
```

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。



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

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

除了 gcc，ncurse，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。

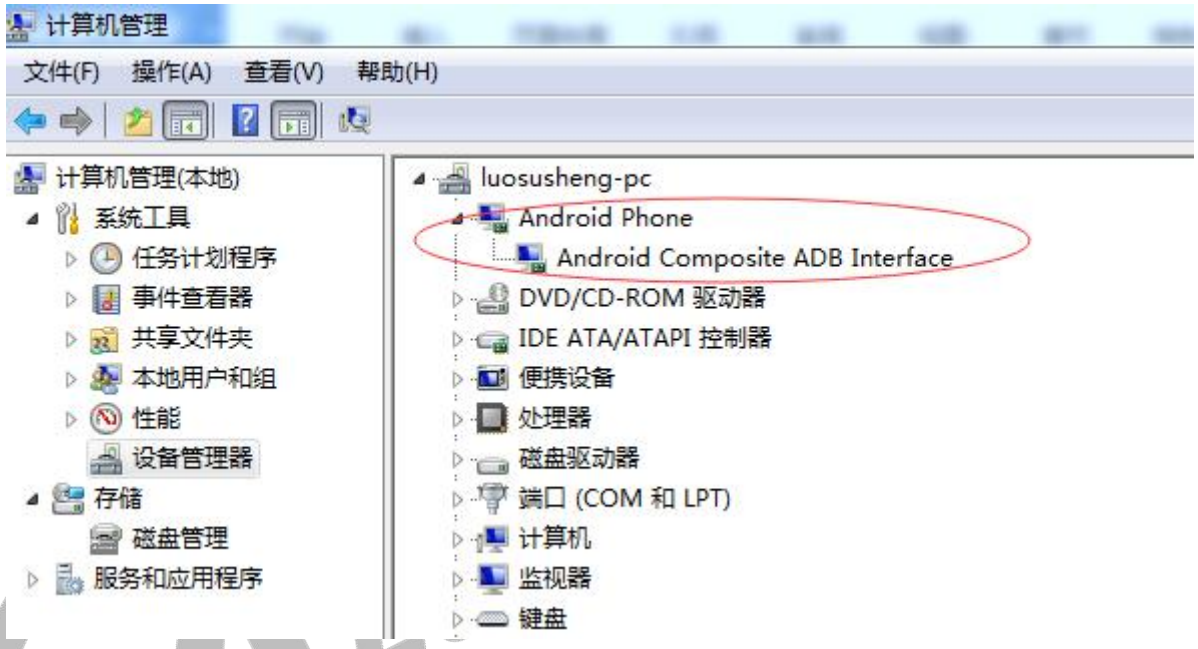
3.2.1. 开发工具安装

请自行选择 SourceInsight，Notepad++，Qt Creator 等 IDE 或其它编辑软件，

以及 Xshenll 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。

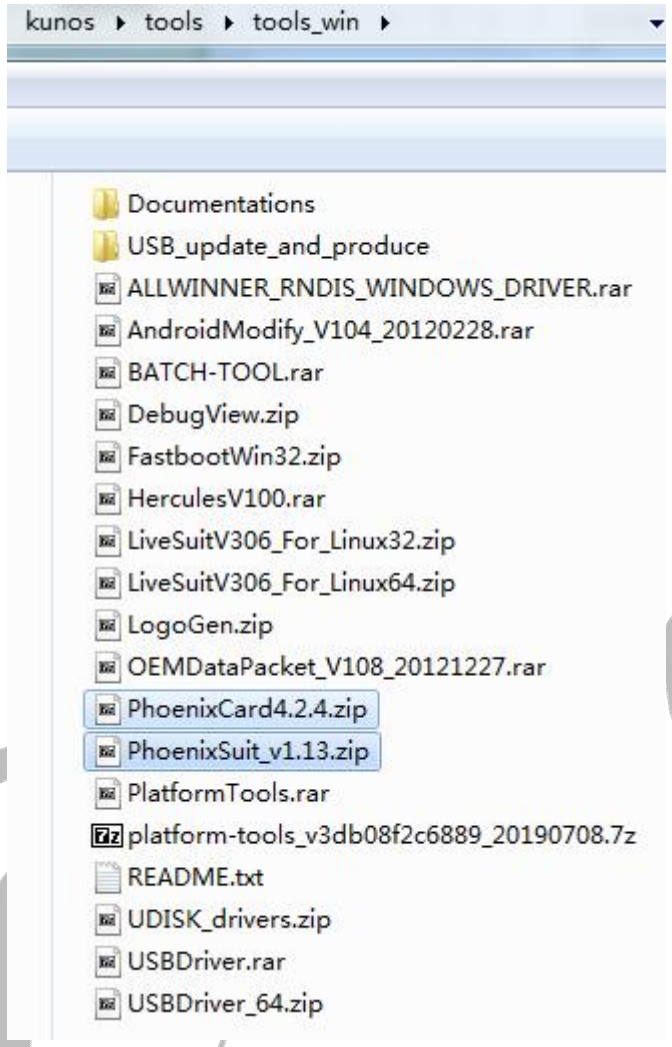


注意：有些电脑在设备上电并插上 USB 线之后，自动安装 USB 设备驱动程序会失败。推荐使用驱动人生等软件，自动检索安装驱动程序。

3.2.3. 烧录软件安装

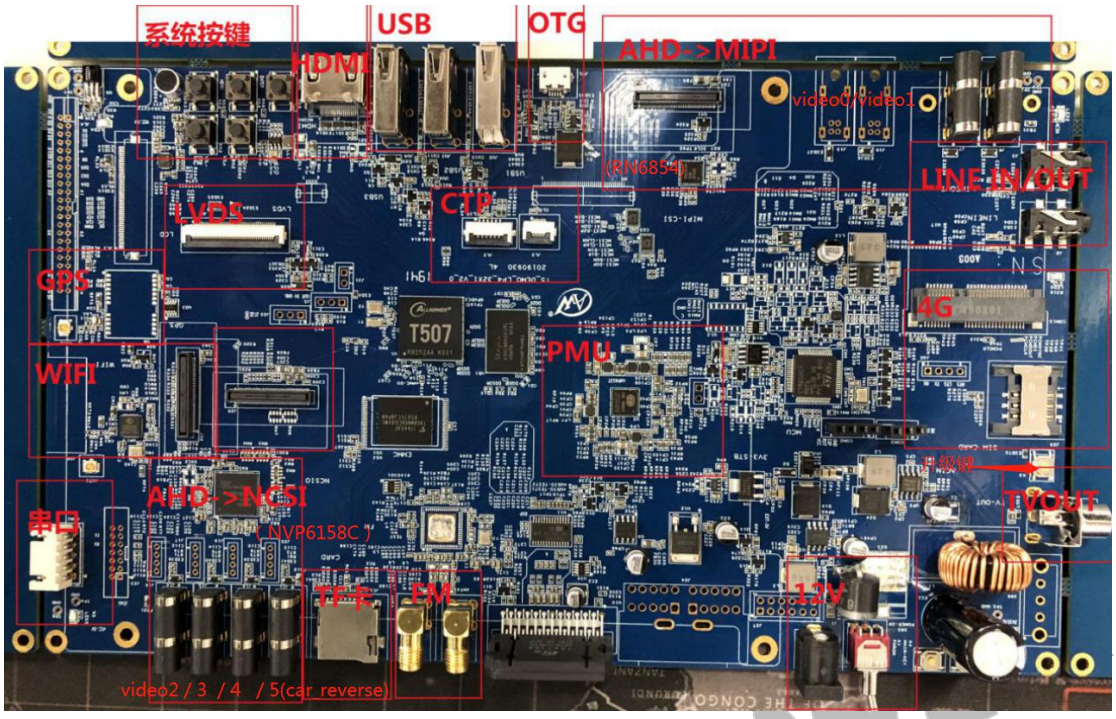
烧录软件 PhoenixSuit-v1.13 和 PhoenixCard4.2.4，分别为 USB 线刷和 SD 卡刷。这两个软件的安装包位于 tools\tools_win\。解压后会有中英文两个版本，选择一个版本即可。当 sdk 编译打包后，就可以通过 PhoenixSuit 烧录，详细步骤将在后文介绍。

(PS:请注意 T5 要使用新的 PhoenixSuit（v1.13）和 PhoenixCard（4.2.4），要不然可能会导致无法进行卡升级等问题。)



3.3. 开发板介绍

AW T5 公版如下



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

3.3.1. 使用准备

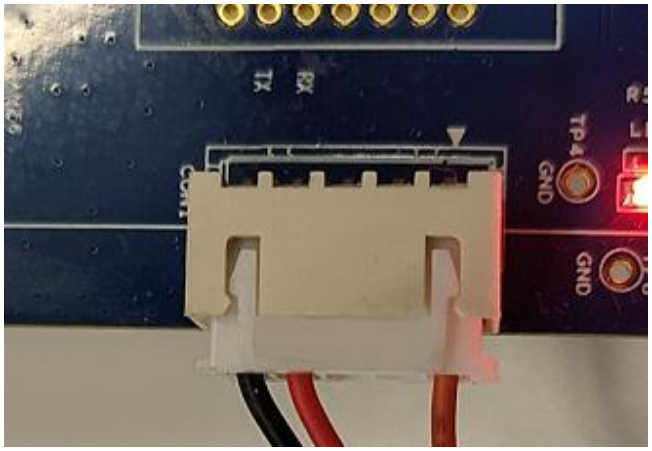
请检查串口硬件工具以及串口连接线、12V 直流电源、以及 miniUSB 线等是否就绪。

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。

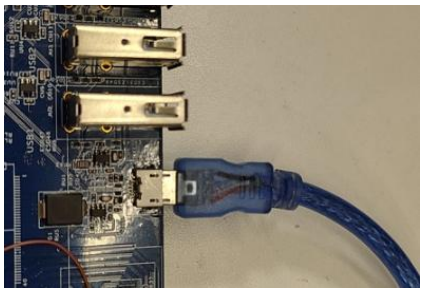
3.3.3. 串口连接

默认的调试串口用的是 uart0，电压为 3.3v，连接如下图：

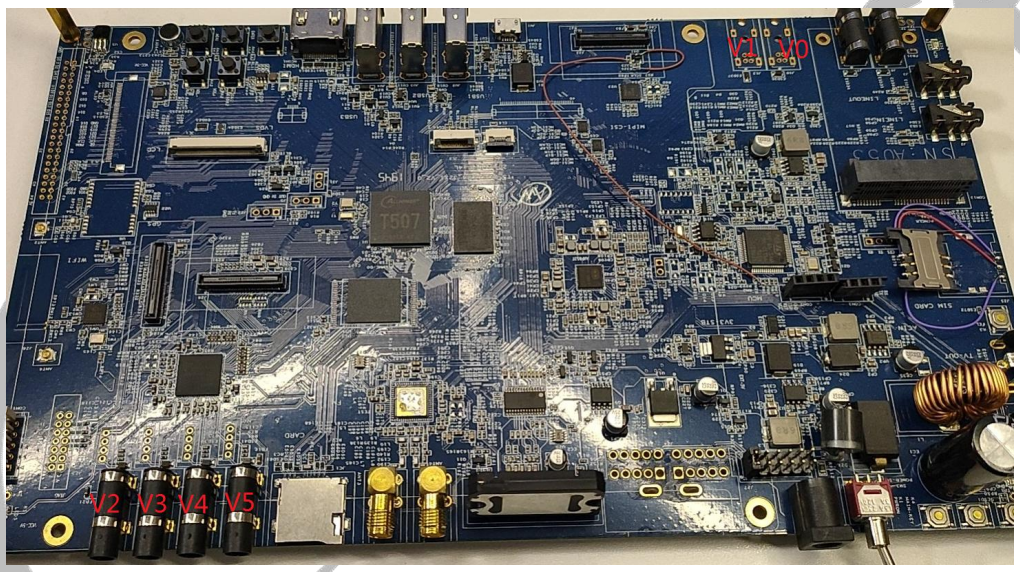


3.3.4. USB 调试连接

请使用 USB Micro 数据线，连接开发板和 windows PC 和 usb 端口。



3.3.5. 摄像头硬件连接



图中 V 表示 video,上面 V2 对应软件上的/dev/video2 节点，图中 0 和 1 两个摄像头默认没有接插座。

Video0/1 由 rn6854 捕获数据，并通过 MIPI-CSI 通道传入 T507

video2/3/4/5 由 nvp6158c 捕获数据，并通过 NCSI 通道传入 T507

所有 video 默认使用的电压都是 5v。

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。

```
步骤 1： source build/envsetup.sh //进行 sdk 环境配置（重要）
步骤 2： ./build.sh //编译整个 sdk
步骤 3： ./build.sh pack //打包固件
```

或者使用传统命令如下（不建议）：

```
步骤 1： ./build.sh config //进行 sdk 环境配置
步骤 2： ./build.sh //编译整个 sdk
步骤 3： ./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。

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
	./build.sh dragonboard	编译 dragonboard
	./build.sh sata	编译 sata
打包	./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

类别	命令	说明
	./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。

类别	扩展命令	等同于	说明
整体编译	source build/envsetup.sh	./build.sh config	编译配置，弹出编译选择，将扩展命令导出到环境变量中
	build	./build.sh	根据编译配置，编译 SDK
	build clean	./build.sh clean	清除过程文件和目标文件
	build distclean	./build.sh distclean	清除所有生成的文件
局部编译	build brandy	./build.sh brandy	编译 brandy
	build kernel	./build.sh kernel	编译 kernel
	build buildroot	./build.sh buildroot	编译 buildroot
	build qt	./build.sh qt	编译 qt
	build dragonboard	./build.sh dragonboard	编译 dragonboard
	build sata	./build.sh sata	编译 sata
打包	pack	./build.sh pack	打包命令，调试串口为 uart0
	pack -d	./build.sh pack_debug	打包命令，调试串口为 card0
	pack -s	./build.sh pack_secure	打包命令，生成 secure 固件，调试串口为 uart0
	pack -sd	./build.sh pack_debug_secure	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

类别	扩展命令	等同于	说明
	cdevice		跳转到 device/target/{prodcut}/{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：

```
./build.sh config
source build/envsetup.sh
build
build qt
build
pack
```

最终生成的 img 参考：out/t507_linux_demo2.0_uart0.img。

4.2.1. 配置环境

执行 build.sh config，在后续对话中选择配置 。Choice 选择时可以输入数字也可以输入数字对应的字符串。

```
$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
    0. android                          //Android 方案
    1. linux                            //非 Android，如 dragonboard
Choice [linux]:1
All available linux_dev:                //注意：Android 方案无此选项
    0. bsp                              //bsp 方案
    1. dragonboard                      //dragonboard 测试系统
    2. longan                           //longan sdk 方案
    3. tinyos
Choice [longan]: 2
All available ic:
    0. t507                             //T507 方案/T507-H 方案
    1. t517                             //T517 方案/T517-H 方案
Choice [t507]: 0
All available board:                    //板级方案
```

```
    0. demo2.0                          //默认 demo2.0
    1. evb                              //如果是 EVB 板，选择 evb

Choice [demo2.0]: 0
All available flash:        //flash 类型，只区分 nor 和非 nor 方案，Android 方案无此选项，默认非 nor
    0. default
    1. nor
Choice [default]: 0
```

4.2.2. T5 auto 编译详解

Auto 提供以下内容：

```
platform/framework/auto/
├── rootfs
├── sdk_demo
└── sdk_lib
```

rootfs: 提供 adb、自动挂载、多媒体等配置，每次编译时会拷贝其里面的内容到 out 目录对应位置（target 下），以便一起打包。在上电时，会挂载部分驱动，设置相关环境。

sdk_lib：提供一些常用库文件，如编解码，摄像头等相关资源。
sdk_demo：提供常用示例程序。其中 sdktest 是摄像头显示和录像程序。详情请查阅对应代码和提示。

```
├── audioenctest        音频编码测试 demo
├── autplayerTest        sdklib 中 autplayer 类（播放器类）测试 demo
├── bin                  此目录所有 demo 编译成功后都会拷贝一份到这里
├── csitest             直接用 v4l2 控制摄像头并显示的 demo
├── decoderTest          视频解码测试
├── encoderTest          视频编码测试(YUV--->h264）
├── fbinit              清除进入终端后的 logo
├── G2dDemo             硬件图像搬移模块，加速图像数据拷贝、合成、拆分
├── logwrapper          log
├── Makefile
├── makefile_cfg
├── mali_test           opengles（GPU）测试
├── memTest             ion 使用 demo,还可以配合 DE 进行色块显示
├── readme.txt
├── recordTest          录像测试,把 YUV--->MP4 （需要插入 sdcard）
├── sdktest             Dvrfactory 类测试，摄像头预览+录像（重要）
├── videoservice
└── xplayerdemo         命令行方式操作的播放器
```



```
├── yuvDisplayTest          DE 显示一张 yuv 图片
```

4.2.3. T5 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.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.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 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
```

4.3. 工具链简介

目前使用三个工具链。

```
build/toolchain/
├──── gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
├──── gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz
└──── gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
```

4.3.1. Kernel 工具链

Kernel 工具链位于：
build/toolchain/gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz
鉴于统一开发的需要，目前 64 位 IC 都采用该工具链来编译内核。

4.3.2. Buildroot 工具链

Buildroot 工具链位于：
build/toolchain/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
理论上，buildroot 应该和 kernel 使用同一个工具链。但在移植 Qt 过程中，发现 gcc-linaro-5.3.1 的 aarch64 版本存在 bug，无法正常编译 Qt。经过验证，需要更高版本的 gcc。因此，为了兼容上层对 Qt 的需求，不得不使用更高版本的 7.4.1 的 linaro gcc 来编译 buildroot 应用。

4.3.3. Dragonboard 工具链

Dragonboard 工具链位于：
build/toolchain/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz
同样的，在理论上 Dragonboard 应该和 kernel 使用同一个工具链。不过因为该系统只是提供板级测试，不必追求性能。我们希望它在 32 位和 64 位上都可以直接使用，避免重复移植编译。因此统一用完整的打包方式来移植，减少不必要的工作量。

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。工具路径 tools/tools_win/

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：

(PS:1.13 的烧录工具位于 tools\tools_win 目录)

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：

注意：如未检测到设备，可能驱动未正常安装。可以使用驱动人生等软件自动安装即可。

5.1.3. 选择 img 文件

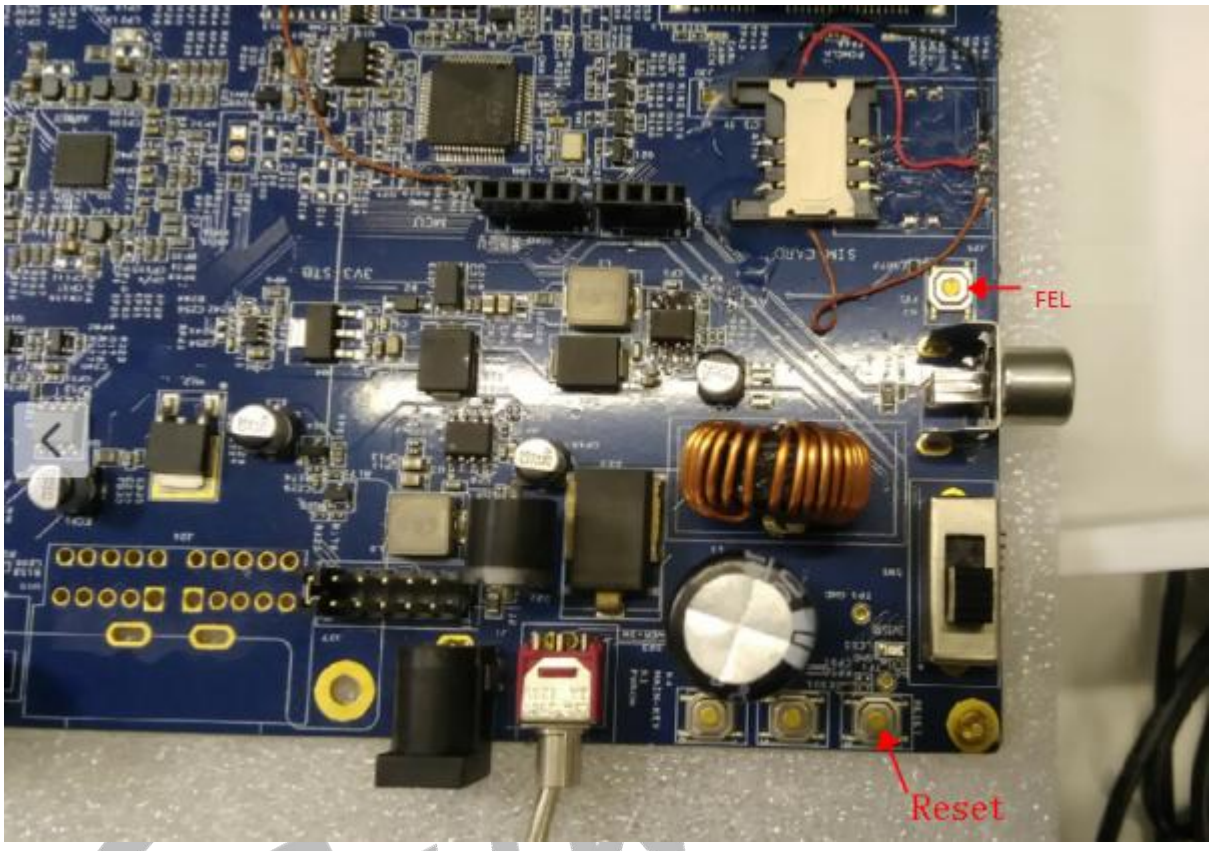
点击上方【一键刷机】图标，选择编译生成的 img 文件：

注意：选择完.img 文件后，若 PhoenixSuit 是 v1.13 以上的版本，可以点“立即升级”进行升级烧录。

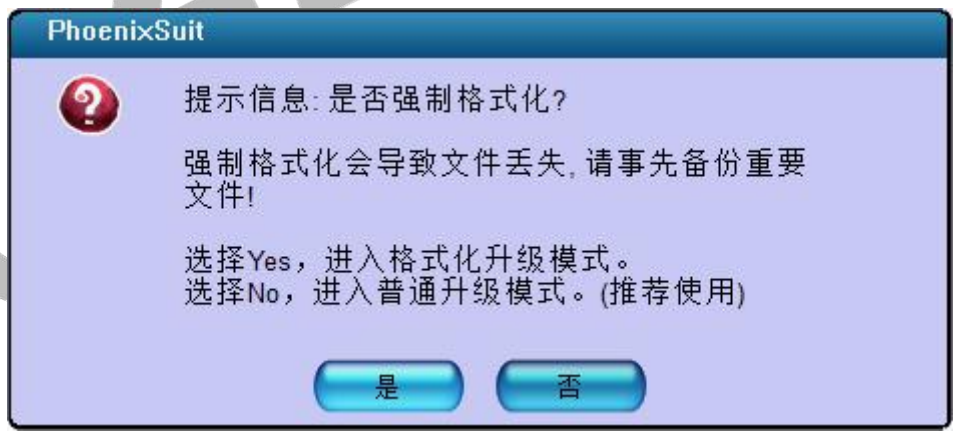
5.1.4. 开始烧录

关闭 T507 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 T507 供电。

或者直接按住 FEL 键，再按 reset。



正确操作后，PhoenixSuit 会提示是否强制格式化。选择是。



之后 PhoenixSuit 提示正在烧写固件，等待烧写成功。

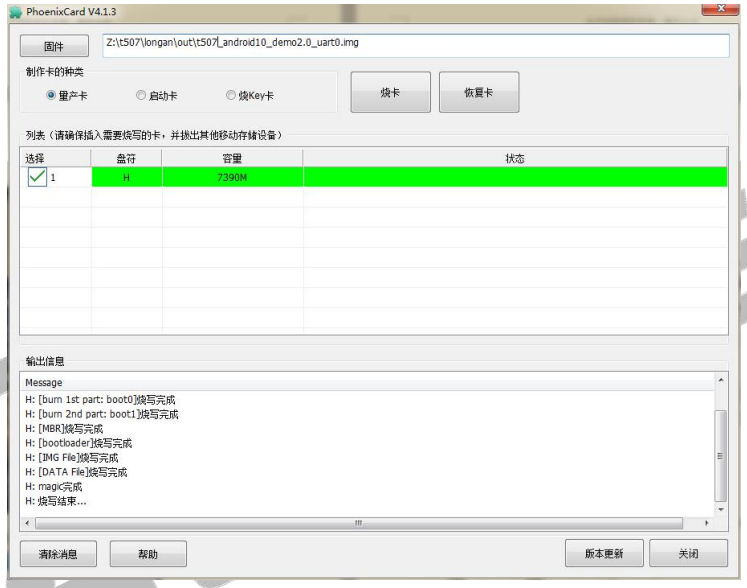
注意：在开始菜单，或者安装目录，可以找到《PhoenixSuit 用户指南.pdf》，里面介绍了烧录软件的整个流程。更具体的信息请参考该文档。

5.2. SD 卡烧录

此种方式常用于量产或售后软件升级。

5.2.1. 制作升级卡

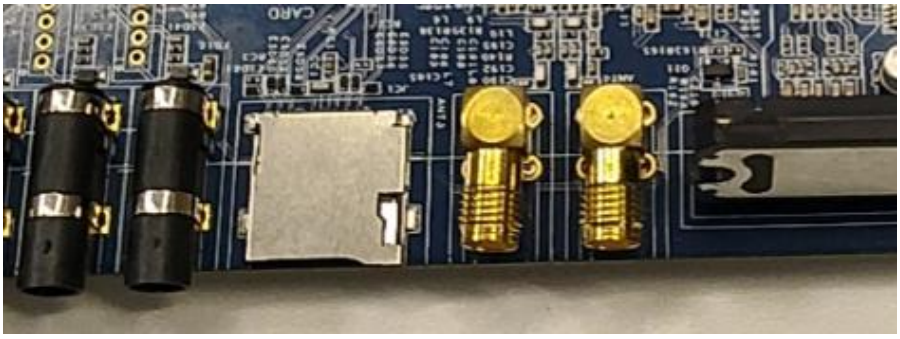
解压 PhoenixCard (工具位于 tools\tools_win 目录)， 然后打开 PhoenixCard.exe，制作 sd 升级卡的相关信息可以点“帮助”查看。参考下图：



（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，插到机器 SD 卡升级的槽上，如下图：



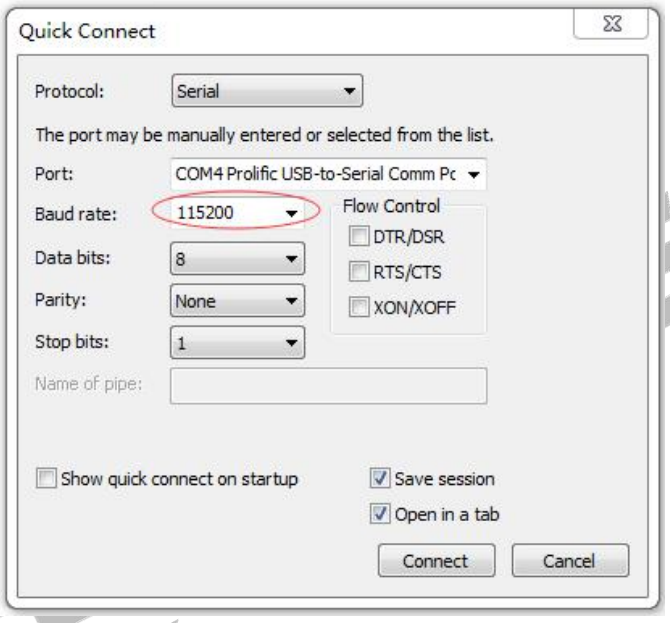
重新上电，机器就会自动升级了。可以看到屏幕上有进度条，调试串口有相应输出。整个过程大概 1~2 分钟，具体视固件大小而定。升级完毕后拔掉 sd 卡，然后重新上电。

6.系统调试

支持串口和 adb 方式来和 windows PC 通讯。

6.1. 串口登录命令行

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。



6.2. 连接成功后，在串口终端回车即可。

```
[      6.253599] insmod_device_driver
[      6.253599] device_chose finished!
add /dev/input/event3 to Qt Application.
#
```

6.3. 使用 adb 调试

6.3.1. 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：

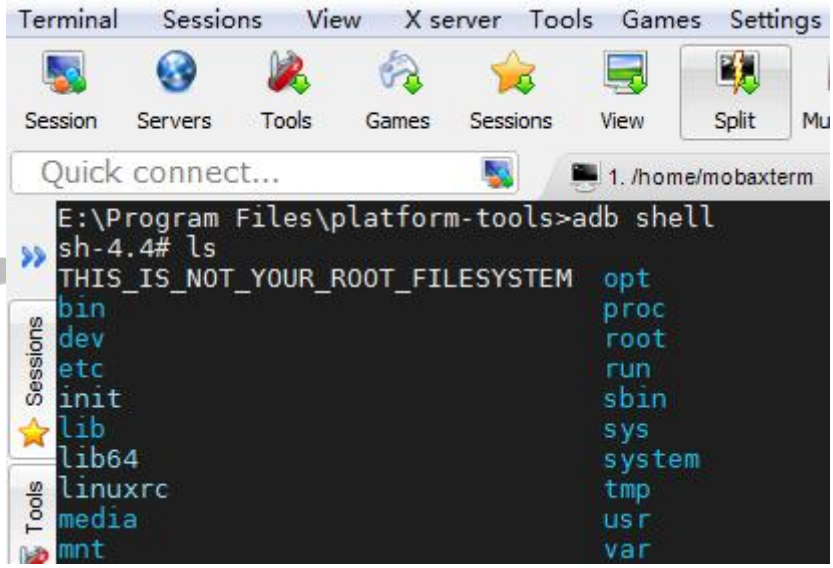
- 运行设备的 shell（命令行）
- 管理模拟器或设备的端口映射
- 在计算机和设备之间上传/下载文件

6.3.2. 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_start.sh ：

```
# /etc/adb_start.sh
[    122.160398] android_work: sent uevent USB_STATE=DISCONNECTED
device_chose finished!
[    122.160657]
[    122.160657] rmmod_device_driver
[    122.160657]
install_listener('tcp:5037','*smartsocket*')
[    122.160730]
[    122.160730] insmod_device_driver
[    122.160730]
[    122.167525] read descriptors
[    122.167541] read strings
# [    122.372886] android_work: sent uevent USB_STATE=CONNECTED
[    122.443130] configfs-gadget gadget: high-speed config #1: c
[    122.449972] android_work: sent uevent USB_STATE=CONFIGURED
...
```

如果出现 android_work: sent uevent USB_STATE=CONFIGURED，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，不用重新烧录了。



6.3.3. ADB 常用命令

6.3.3.1. PC 端查看当前连接的设备

```
adb devices
```


该命令返回的结果是电脑的 android 设备或者模拟器；

6.3.3.2. PC 端进入设备 shell 命令行模式

```
adb shell
```

6.3.3.3. 将电脑上的文件上传至设备

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

用 push 命令可以将电脑上的文件或者文件夹上传至设备。 local path 一般指电脑；remote path 一般指 adb 连接的设备。例如：将 F:\多媒体测试\cam720.yuv 文件上传至设备/tmp 目录：

```
adb push F:\多媒体测试\cam720.yuv /tmp
```

6.3.3.4. 下载设备里的文件到电脑

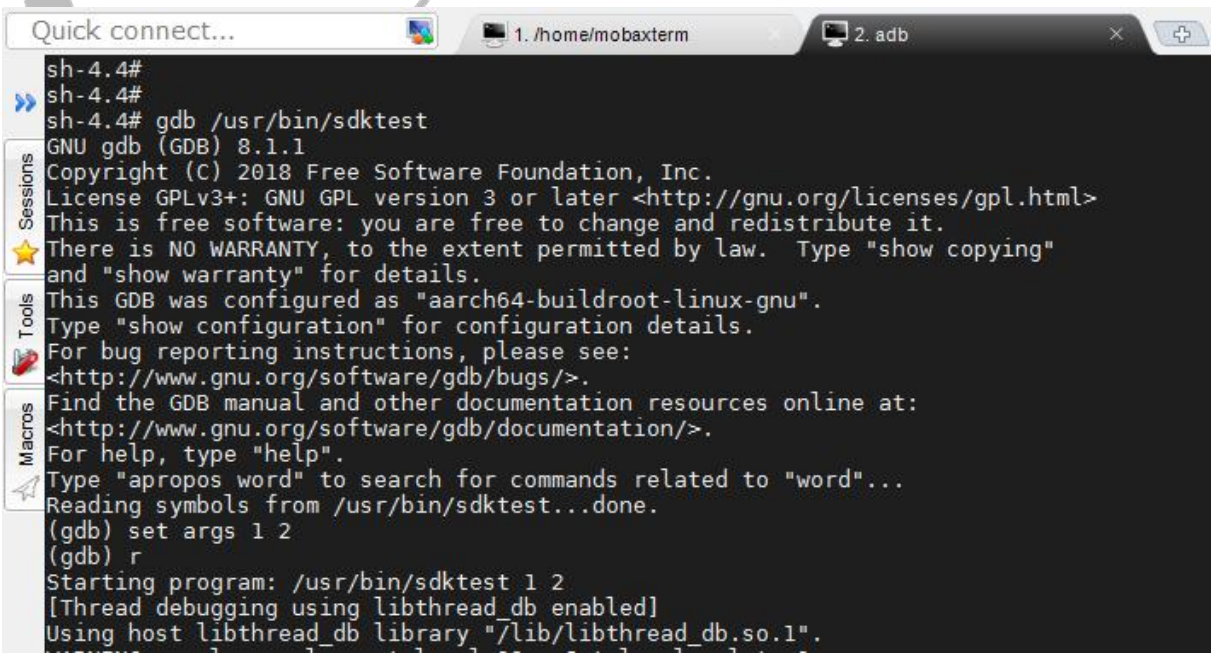
```
adb pull    <remote path>    <local path>
```

pull 命令可以把设备上的文件或者文件夹下载到本机电脑中。例如：将设备/tmp/cam720p.H264 文件下载至电脑 D:\

```
adb pull /tmp/cam720p.H264 D:\
```

6.4. GDB 调试工具

Sdk 中提供标准 gdb 工具可以在开发板上进行本地调试。
在终端可以直接运行 gdb 来调试程序：



7.Dragonboard 测试系统的使用

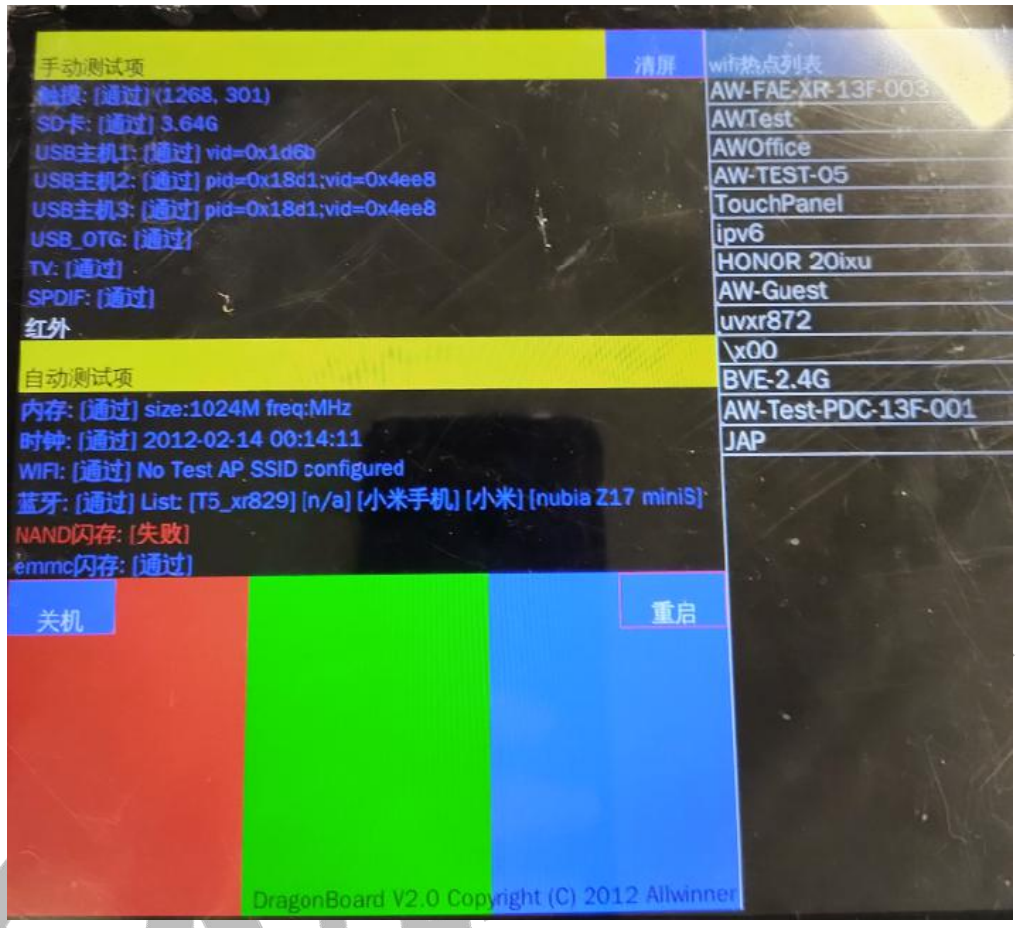
Dragonboard 是全志的测试系统。在编译时，选择 dragonboard 可以生成用于测试的 dragonboard 的 img，和 **longan 选项并列**：

```
lichee$ ./build.sh config
Welcome to mkscript setup progress
All available platform:
    0. android                          //Android 方案
    1. linux                            //非 Android，如 dragonboard
Choice [linux]:1
All available linux_dev:                //注意：Android 方案无此选项
    0. bsp                              //bsp 方案
    1. longan                           //longan sdk 方案
    2. Dragonboard                      //Dragonboard 测试系统
    3.tinyos
Choice [longan]: 2
All available ic:
    0. t507                             //T507/T517 方案
    1. t507_h                           //T507-H 方案（带 HDMI）
    2. t517_h                           //T517-H 方案（带 HDMI）
Choice [t507]: 0
All available board:                    //板级方案，这里选择 demo2.0
    0. demo
    1. demo2.0
Choice [demo2.0]: 1
All available flash:        //flash 类型，只区分 nor 和非 nor 方案，Android 方案无此选项，默认非 nor
    0. default
    1. nor
Choice [default]: 0
```

之后，编译打包，会生成 dragonboard 的 img：

```
 lichee$    ./build.sh
 lichee$    pack
 ... ...
Dragon execute image.cfg SUCCESS !
----------image is at----------
size:144M    /home/t5_rls/out/t507_dragonboard_demo2.0_uart0.img
pack finish
```

烧录这个 img，启动系统，可以进入 dragonboard 测试：



8.auto sdk 简介

auto sdk 为 T5 linux 提供的常用接口、示例和配置脚本等个性化工具。文件位于 platform/framework/auto。

```
.
├── build.sh                    //编译脚本
├── doc                         //文档目录
├── qt_demo                     //Qt 示例
├── readme.txt                  //编译说明
├── rootfs                      //rootfs 文件
├── sdk_demo                    //sdk 应用示例
└── sdk_lib                     //sdk 库
```

sdk_lib 提供了摄像头、显示、编解码的封装接口。

```
.
├── cedarx                      //编解码库
├── include                     //头文件
├── lib64                       //库文件路径
│   ├── libsdk_compose.so       //摄像头图像合成
│   ├── libsdk_decoder.so       //视频裸流解码
│   ├── libsdk_disp.so          //显示
│   ├── libsdk_dvr.so           //摄像头编码类
│   ├── libsdk_encoder.so       //视频裸流编码
│   ├── libsdk_g2d.so           //G2D 接口
│   ├── libsdk_player.so        //autplayer 播放器
│   └── libsdk_stream_player.so //视频裸流解码显示
├── Makefile                    //make 文件
├── makefile_cfg                //make 配置
├── push.bat                    //adb 更新 lib64 脚本
├── pushcdxlib.bat              //adb 更新 cedarx 脚本
├── sdk_audioenc                //音频编码
├── sdk_camera                  //摄像类
├── sdk_config                  //ini 解析
├── sdk_ctrl                    //系统控制
├── sdk_log                     //log 打印
├── sdk_memory                  //内存分配
├── sdk_misc                    //常用工具库
├── sdk_opengl                  //GPU 示例库
├── sdk_shm                     //共享内存
├── sdk_sound                   //音频库
├── sdk_storage                 //存储类
└── sdk_tinyalsa                //tinyalsa 音频库
```

以上单独列出的 lib64 中的库文件不开源。其余 sdk_lib(如 sdk_camera)在编译后，

生成的库文件(libsdk_camera.so)也会放入 lib64 中。

说明：

cedarx 是 Allwinner 系 SOC 平台的多媒体中间件，处理流媒体/解封装/解码/音视频同步等逻辑。

8.1. ION

sunxiMemInterface 提供便捷的 ION 接口，方便应用统一分配和管理。

头文件：
platform\framework\auto\sdk_lib\include\sunxiMemInterface.h
库文件：
libsdk_memory.so
源代码：
platform\framework\auto\sdk_lib\sdk_memory
示例：
platform\framework\auto\sdk_demo\memTest\MemTest.cpp 为对应的测试示例。
若想把申请的内存数据用 DE 显示模块显示出来的话，则可参考 MemTestDE.cpp

```
allocOpen       //打开设备，创建管理信息、引用计数等。
allocClose      //关闭设备。
allocAlloc      //分配内存。MEM_TYPE_CDX_NEW 一般用于 cedar 解码库。
allocFree       //释放内存。
allocVir2phy    //虚拟地址转物理地址。
allocPhy2vir    //物理地址转虚拟地址。
flushCache      //刷新。
```

8.2. 显示

HwDisplay 类提供显示接口。
头文件：
platform\framework\auto\sdk_lib\include\disp2\hwdisp2.h
库文件：
libsdk_disp.so
示例：
platform\framework\auto\sdk_demo\yuvDisplayTest\yuvDisplayTest.cpp

显示模组可以配置双屏 screen0 和 screen1。screen 0 有四个显示通道，每个显示通道有 4 个硬件图层。screen 1 有 2 个显示通道，每个显示通道有 4 个硬件图层。目前可以使用各通道的第一个图层，即 layer0。

8.2.1. 接口简介

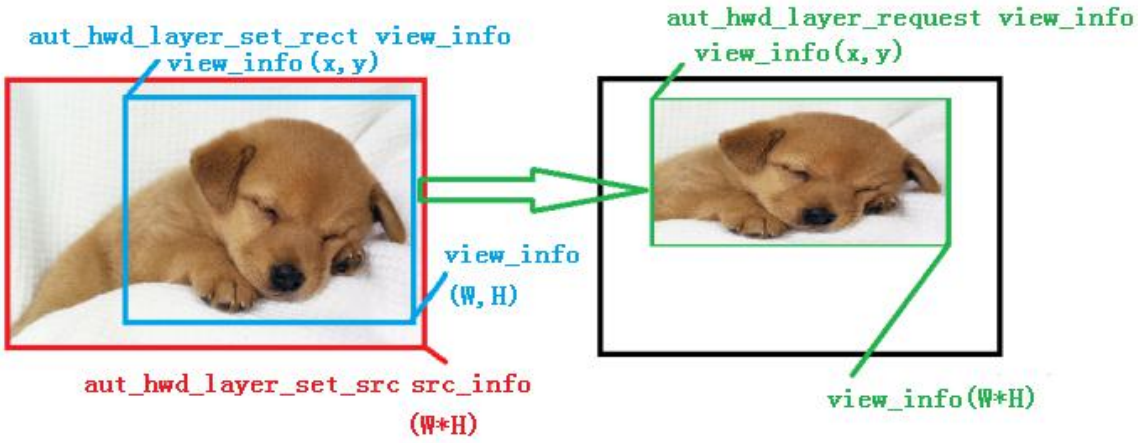
int aut_hwd_layer_request(struct view_info* surface,int screen_id,unsigned int channel,unsigned int layer_id);
申请显示图层,view_info 为显示位置和大小。

int aut_hwd_layer_set_src(unsigned int hlay, struct src_info *src,unsigned long addr0,unsigned int share_fd = -1);
设置图像源的大小，地址或者 dma_fd（二选一，如果 share_fd 为默认值，则使用物理地址）。

int aut_hwd_layer_set_rect(unsigned int hlay, struct view_info *src_rect);// impl todo
设置图像源显示范围，即从图像源中选择 src_rect 用来显示。

如图示，红色矩形代表 src 源数据，蓝色矩形为 corp。黑色矩形表示屏幕大小，绿色为显示窗口信息。

当然一般情况下，红蓝的大小是一致的，黑绿的大小也是一致的。当显示长宽比和 corp 不一致时，会有拉伸效果。



int aut_hwd_layer_set_zorder(unsigned int hlay,int zorder);
设置图层 zorder，zorder 值大的图层会覆盖在 zorder 值小的图层上方。

int hwd_other_screen(int screen, int type, int mode)
设置该 screen 输出到其它设备，比如 CVBS 或者 HDMI。

int aut_hwd_layer_open(unsigned int hlay);
开启图层。

8.2.2. 应用示例

platform\framework\auto\sdk_demo\yuvDisplayTest\yuvDisplayTest.cpp 是一个简单的示例，将一张 YUV 图像输出到屏幕显示。可以使用物理地址，也可以使用 dma_fd。

注意：

在实际应用中，建议使用两个以上的 buf，以防止硬件刷新时，上层更新 buf 导致花屏。

8.2.3. 调试信息

可以通过命令：cat /sys/class/disp/disp/attr/sys 来查看当前显示数据：

```
screen 0:
de_rate 696000000 hz, ref_fps:61
mgr0: 1280x800 fmt[rgb] cs[0x204] range[full] eotf[0x4] bits[8bits] err[0] force_sync[0] unblank direct_show[false]
dmabuf: cache[0] cache max[0] umap skip[0] overflow[0]
        lcd output      backlight( 50)  fps:61.4        1280x 800
        err:1   skip:28 irq:6051        vsync:6051      vsync_skip:0
  BUF   enable ch[0] lyr[0] z[1] prem[N] a[globl 255] fmt[ 77] fb[1280, 720; 640, 360; 640, 360] crop[   0,
0,1280, 720] frame[   0,   0,1280, 800] addr[fca00000,fcae1000,fcb19400] flags[0x       0] trd[0,0]
depth[ 0] transf[0]
  BUF   enable ch[1] lyr[0] z[5] prem[N] a[pixel   0] fmt[  0] fb[1280, 800;1280, 800;1280, 800] crop[   0,
0,1280, 800] frame[   0,   0,1280, 800] addr[ff800000,       0,       0] flags[0x       0] trd[0,0]
depth[ 0] transf[0]
```

```
screen 1:
de_rate 696000000 hz, ref_fps:60
mgr1:  720x480  fmt[yuv444]  cs[0x204]  range[full]  eotf[0x4]  bits[8bits]  err[0]  force_sync[0]  unblank
direct_show[false]
dmabuf: cache[0] cache max[0] umap skip[1] overflow[1]
        tv output mode(14)      fps:60.4        720x 480
        err:0   skip:0  irq:722 vsync:722       vsync_skip:0
  BUF   enable ch[0] lyr[0] z[1] prem[N] a[globl 255] fmt[ 77] fb[1280, 720; 640, 360; 640, 360] crop[   0,
0,1280, 720] frame[   0,   0, 720, 480] addr[fca00000,fcae1000,fcb19400] flags[0x       0] trd[0,0]
depth[ 0] transf[0]
```

以上分别为 screen 0 和 screen 1 开启的图层信息，通过以上信息可以看到 output 输出设备和配置信息,分别为 lcd output 和 tv output。

每一个 BUF 对应一个开启图层。没有出现的图层即为关闭状态。重点数据如下，相关数值对应 sunxi_display2.h 中 disp_layer_info2 定义：

```
ch[3] lyr[0] z[1]：通道、图层和 zorder 值。
a[pixel   0] ：alpha 类型和值。
fmt[  0] ：数据格式
fb[..] crop[..] frame[...] addr[...]  对应上述设置的 src_info、src_rect、和 view_info。
addr[...] 图像数据物理地址。
```

当显示异常时，我们可以通过上述信息快速排查相关参数是否正确。

说明：

默认地，视频(播放器和摄像等)会开启 ch0 ly0 的图层。因此 fb 节点是从 ch1 开

始的。

8.2.4. 应用双屏配置

显示模组可以配置双屏 screen0 和 screen1。在申请图层的时候，可以指定对应的 screen 和图层信息。

```
int aut_hwd_layer_request(struct view_info* surface,int screen_id,
unsigned int channel,unsigned int layer_id);
```

其中，screen_id,channel,layer_id 即为对应的 screen、显示通道、显示图层。

通过 hwd_other_screen 可以设置每个 screen 的显示设备：

```
int hwd_other_screen(int screen, int type, int mode);
```

sunxi_display2.h 定义了屏显的类型和模式：

```
enum disp_output_type
{
    DISP_OUTPUT_TYPE_NONE  = 0,
    DISP_OUTPUT_TYPE_LCD   = 1,
    DISP_OUTPUT_TYPE_TV    = 2,
    DISP_OUTPUT_TYPE_HDMI  = 4,
...
enum disp_tv_mode
{
    DISP_TV_MOD_480I                = 0,
    DISP_TV_MOD_1080P_60HZ          = 0xa,
    DISP_TV_MOD_NTSC                = 0xe,
...
```

以上设置也可以在系统启动时初始化 device\product\configs\demo2.0\board.dts，对应 screenx_output_type 和 screenx_output_mode：

```
    disp: disp@01000000 {
        disp_init_enable            = <1>;
        disp_mode                   = <0>;
        screen0_output_type         = <1>;
        screen0_output_mode         = <4>;
        dev0_output_type            = <1>;
        dev0_output_mode            = <4>;
```

screen0 为系统启动之后的显示设置，dev0 为 boot 阶段的显示设置。

8.2.5. 内核双屏配置

board.dts 也可以支持 kernel 启动时的双屏设定。需要两步设置。

第一步，开启该功能需要打开内核配置选项 DISP2_SUNXI_MUTI_SCREEN。

-> Device Drivers -> Graphics support -> Frame buffer Devices -> Video support for sunxi

```
[ ] Framebuffer Console Support(sunxi)
<*> DISP Driver Support(sunxi-disp2)
< >   HDMI Driver Support(sunxi-disp2)  ----
[*] HDMI2.0 HDCP  --->
[*] HDMI2.0 CEC  --->
[ ] new frequency spread spectrum(hershey) of sunxi
<*> HDMI2.0 Driver Support(sunxi-disp2)
<*> TV Driver Support(sunxi-disp2)
< > VDPO Driver Support(sunxi-disp2)
< > GM7121 TV module Support(sunxi-disp2)
< > EDP Driver Support(sunxi-disp2)
[ ] boot colorbar Support for disp driver(sunxi-disp2)
[*] debugfs support for disp driver(sunxi-disp2)
[*] composer support for disp driver(sunxi-disp2)
[ ] ESD detect support for LCD panel
[*] Support for two screen display
```

注意：

该功能仅在较新的代码版本上支持。如果找不到该选项，则代表该代码版本不支持。

第二步，在 board.dts 中通过 disp_mode 指定显示模式。

- disp_mode0= <0>;

为 screen0 单屏显示。

- disp_mode0= <3>;

为双屏异显，fb1 单独绑定到 screen1 的 ch 1 ly 0。fb1 的长宽 fb1_width 和 fb1_height 可以自行指定。其余 fb 绑定到 screen0 的对应 ch。

- disp_mode0= <5>;

为双屏同显，fb0 同时绑定到 screen0 的 ch 1 ly 0 和 screen1 的 ch 1 ly 0。其余 fb 绑定到 screen0 的其它通道。

screenx_output_type 和 screenx_output_mode 用来指定两个屏幕的输出类型。其值参见上一节的描述。例如，需要设置为 LCD 和 HDMI 同时输出可以使用下述值：

```
disp: disp@01000000 {
    disp_init_enable            = <1>;
    disp_mode                   = <5>;

    screen0_output_type         = <1>;
    screen0_output_mode         = <4>;

    screen1_output_type         = <3>;
    screen1_output_mode         = <10>;
```

对应代码为 drivers\video\fbdev\sunxi\disp2\disp\dev_fb.c 的 fb_init 函数，其它个性化设置因方案不同差异也较大，不做统一支持。用户可以阅读该代码，自行修改每个 FB 的参数，和需要绑定的显示图层。

config_fb_by_screen：根据对应显示设备配置 FB 的大小等参数，并分配内存。
bind_fb_to_screen：将配置好的 FB 绑定到对应的显示设备。

```
//bind fb to screen.
switch (disp_mode) {
    case DISP_INIT_MODE_SCREEN0: {
        screen_id = 0;//bind all fb to screen0
        config_fb_by_screen(i, screen_id);
        ret = bind_fb_to_screen(i, screen_id);
        break;
    }
    case DISP_INIT_MODE_SCREEN1: {
        screen_id = 1;//bind all fb to screen1
        config_fb_by_screen(i, screen_id);
        ret = bind_fb_to_screen(i, screen_id);
        break;
    }
    case DISP_INIT_MODE_TWO_DIFF_SCREEN: {
        if (i == 1) {
            screen_id = 1;          //bind fb1 to screen1
        } else{
            screen_id = 0;          //bind other fbs to screen0
        }
        config_fb_by_screen(i, screen_id);
        bind_fb_to_screen(i, screen_id);          //bind all fbs to screen0
        break;
    }
    case FB_MODE_DUAL_DIFF_SCREEN_SAME_CONTENTS: {
        if (i == 0) {//bind fb0 to all screen.
            config_fb_by_screen(i, screen_id);
            for (screen_id = 0; screen_id < num_screens; screen_id++) {
                ret = bind_fb_to_screen(i, screen_id);//bind fb0 to all screen.
            }
        } else { //bind other fbs to screen0
            screen_id = 0;
            config_fb_by_screen(i, screen_id);
            bind_fb_to_screen(i, screen_id);
        }
        break;
    }
    default: {
        screen_id = 0;//bind all fb to screen0
        config_fb_by_screen(i, screen_id);
        ret = bind_fb_to_screen(i, screen_id);
```

```
                    break;
                }
            }
```

8.3. 摄像头

8.3.1. dvr_factory 类

sdk_camera 中 dvr_factory 类提供了摄像头开闭、录像、预览等功能。位于 platform\framework\auto\sdk_lib\include\DvrFactory.h 。 部分源代码位于 platform\framework\auto\sdk_lib\sdk_camera。库文件 libsdk_camera.so。相关接口如下。

```
dvr_factory* pdvr=new dvr_factory(0);//参数 0 表示打开/dev/video0 节点摄像头
pdvr->recordInit();         //初始化录像参数
pdvr->setDuration(60);    //设置录像时间 1min
pdvr->startRecord();        //开始录像
pdvr->startPriview();       //启动摄像头预览
pdvr->stopPriview();        //停止摄像头预览
pdvr->stopRecord();          //停止录像
```

platform\framework\auto\sdk_demo\sdktest\sdktest.cpp 是对应 dvr_factory 使用的示例。

8.3.2. sdktest 的使用

直接在终端输入 sdktest 可以得到使用方法提示。

sdktest 1 0 表示测试 1 个摄像头，摄像头节点为 0，即/dev/video0,运行成功的话应该会出来一张纯蓝图片显示在 LCD 上，这个纯蓝图片是 rn6854 输出的纯色 YUV。

sdktest 1 1 测试 1 个摄像头，其节点为 1。也是一张纯色 YUV 图片

sdktest 1 3 测试 1 个摄像头，其节点为 3（/dev/video3）。在无摄像头接入的情况下，这个输出的是 color bar。

sdktest 2 1 3 测试两个摄像头，其节点分别为 1,3。第一个摄像头显示 7s 后黑屏 3 秒，切第二个摄像头画面 7 秒，然后黑屏 3 秒，切第一个摄像头画面。。。

sdktest 1 360 测试 4 摄像头（/dev/video2~5）同时田字格显示。

如果用户提前有插入 sd 卡的话，则会做对应的录像动作，录像文件路径默认是/mnt/sdcard/mmcblk1p1/下。

默认不接摄像头的话，板子上的 sensor 也是有 color bar 输出的，插入摄像头后，color bar 数据变成摄像头数据。

关于摄像头节点
/dev/video0　驱动对应 rn6854(MIPI-CSI 通道输入 t507)　默认无摄像头情况下数据为纯蓝
/dev/video1　驱动对应 rn6854(MIPI-CSI 通道输入 t507)　默认无摄像头情况下数据为纯蓝

/dev/video2　驱动对应 nvp6158c(NCSI 通道输入 t507)　默认无摄像头下数据为纯绿色
/dev/video3　驱动对应 nvp6158c(NCSI 通道输入 t507)　默认无摄像头下数据为竖彩条
/dev/video4　驱动对应 nvp6158c(NCSI 通道输入 t507)　默认无摄像头下数据为彩格子。
/dev/video5　驱动对应 nvp6158c(NCSI 通道输入 t507)　默认无摄像头下数据为横彩条。这个节点也是倒车摄像头使用的节点。在测试倒车的时候，需要确保应用没有使用这个节点，然后波动 tvout 旁边的拨码开关，即可进入倒车摄像头界面。

8.3.3. csitest

如果不想用 dvr_factory 类，只想用 v4l2 直接控制摄像头怎么办？或者用户想通过一个更简单的摄像头应用来调试 sensor 怎么办？针对上面场景，我们做了一个 csitest 用例，介绍如何使用 v4l2 接口来控制摄像头，把摄像头数据以文件的形式保存下来。
代码路径位于：
platform\framework\auto\sdk_demo\csitest，具体使用方法在此目录的 readme 中有详细介绍。
如果把代码中的#define display_frame 1 打开，则可以把摄像头数据直接显示在屏幕上。
如果在调试 sensor 初期，不太建议直接显示，建议还是用保存文件的形式来调，避免显示设置问题带来的麻烦。
相反的，如果摄像头直接显示花屏，也请看一下对应保存的文件是否有问题，再做进一步分析。

8.4. 播放器

播放器是通过 libsdk_player.so 以及对应的 cedar 库实现。

8.4.1. xplayer

xplayer 是 cedarx 提供的播放器接口。

头文件：

platform\framework\auto\sdk_lib\cedarx\include\xplayer\include\xplayer.h，需要配合音频、显示等接口实现播放。
库文件：
platform\framework\auto\sdk_lib\cedarx\
示例文件：
platform\framework\auto\sdk_demo\xplayerdemo\xplayerdemo.cpp

其中 libsdk_player.so 封装了音频、显示等接口实现。

8.4.2. AUTPlayer

AUTPlayer 是基于 xplayer 封装的播放类，屏蔽了音频和显示等细节，便于直接使用。

头文件：
platform\framework\auto\sdk_lib\include\AutPlayer.h
库文件：
libsdk_player.so 与 cedarx 库
示例文件：
platform\framework\auto\sdk_demo\autplayerTest\autplayerTest.cpp

```
setUserCb()          //请勿在 callback 直接操作 AUTPlayer 接口。
setUrl()             //设置播放文件路径
play()               //播放
pause()              //暂停
getMediaInfo()       //获取媒体信息，ID3 等
switchSubtitle()     //切换字幕
switchAudio()        //切换音轨
```

8.5. 视频编码

AWVideoEncoder 将指定的 YUV 图像数据，编码为 H264 或者 Jpeg 格式。额外可以进行画面旋转和缩放。注意，T5 不支持 h265 硬编码。

头文件：
platform\framework\auto\sdk_lib\include\AWVideoEncoder.h
库文件：
libsdk_encoder.so
示例：
platform\framework\auto\sdk_demo\encoderTest\encoderTest.cpp

详细描述参见 platform\framework\auto\sdk_demo\encoderTest 的说明文件。

8.6. 视频解码

AWVideoDecoder 将指定的 H264、H265 或者 Jpeg 视频流，解码为 YUV 数据。可以额外实现缩小和旋转。

头文件：
platform\framework\auto\sdk_lib\include\AWVideoDecoder.h
库文件：
libsdk_decoder.so
示例：
platform\framework\auto\sdk_demo\decoderTest\decoderTest.cpp

详细描述参见 platform\framework\auto\sdk_demo\decoderTest 的说明文件。

8.7. fbinit

清除 fb 上的数据。

源代码：
A:\t5\platform\framework\auto\sdk_demo\fbinit\fbinit.c

开发板终端输入：
fbinit 0

8.8. GPU

GPU 库位于 platform\core\graphics\gpu_um_pub\
示例代码位于：platform\core\graphics\samples

sdktest 使用 GPU：
auto sdk 中，有摄像头使用渲染示例：platform\framework\auto\sdk_lib\sdk_opengl
使用时，在 platform\framework\auto\sdk_lib\makefile_cfg 中，打开
DEFINES +=-DDMAFD_TEST 定义，重新编译运行即可。

8.9. qt_demo

qt_demo 是简单的 Qt 示例，主要包含摄像头 CameraUI 和播放器 MediaUI 两个应用。分别基于 dvr_factory 和 AUTPlayer 实现。

8.10. rootfs

```
rootfs/
├── etc
```



```
│   ├── adb_start.sh          //启动 adb
│   ├── asound.conf           //音频配置
│   ├── cedarc.conf           //cedarc 配置插件、打印等级等信息
│   ├── cedarx.conf           //cedarx 配置部分解码参数和打印信息
│   ├── dvrconfig.ini         //摄像头参数设置
│   ├── fstab                 //
│   ├── init.d                //
│   │   └── rcS               //开机自启动脚本
│   ├── inittab               //
│   ├── mdev/...              //自动挂载脚本
│   ├── mdev.conf             //
│   └── qtenv.sh              //Qt 环境设置
└── usr                       //
    └── bin                   //
        ├── cpu_monitor       //cpu 监测
        ├── gpio              //gpio 控制
        ├── logcat            //logcat 程序
        └── mtop              //DDR 监测
```

在编译时，打包工具会将这些工具打包到目标板的 rootfs 中，系统启动之后会开始加载和运行。

8.10.1. cpu_monitor

cpu 的监控工具，监控当前各个 cpu 的运行情况。

```
---------------------- CPU[x]<Freq:MHZ>-<Usage:%>-------------------------
    CPU0          CPU1          CPU2          CPU3       Temp   GPU  GTemp   DDR  Times
|1512   2%| |1512   6%| |1512   6%| |1512   4%|   62   306  62270     0   112
|1512   5%| |1512   4%| |1512   8%| |1512   4%|   62   306  62189     0   113
|1512   4%| |1512   2%| |1512  10%| |1512   4%|   62   306  62189     0   114
|1512   4%| |1512   6%| |1512   4%| |1512   4%|   62   306  61622     0   115
|1512   0%| |1512   8%| |1512   2%| |1512  10%|   62   306  61622     0   116
|1512   2%| |1512   6%| |1512   0%| |1512   8%|   62   306  62189     0   117
|1512   2%| |1512   0%| |1512  12%| |1512   4%|   62   306  62189     0   118
|1512   7%| |1512  10%| |1512   6%| |1512   4%|   62   306  62189     0   119
|1512   2%| |1512   4%| |1512   7%| |1512   4%|   62   306  62189     0   120
|1512   7%| |1512   8%| |1512   6%| |1512   4%|   62   306  61865     0   121
```

8.10.2. mtop 监控 ddr 状态

ddr 监控工具，监控当前 ddr 的使用情况。使用方法可以带参-h 得到提示

```
mtop -h

Usage: mtop [-n iter] [-d delay] [-m] [-o FILE] [-h]
```

```
        -n NUM      Updates to show before exiting.
        -d NUM      Seconds to wait between update.
        -m unit: KB
        -o FILE     Output to a file.
        -v Display mtop version.
        -h Display this help screen.
```

mtop 实际运行情况如下：

```
#mtop

iter: -1
dealy: 1.0
unit: KB
output:

total: 840938, num: 1, Max: 840938, Average: 840938
   totddr    cpuddr    gpuddr    de_ddr    ve_ddr ve1_ddr csi0_ddr csi1_ddr
   840938    109743         0    731195         0       0       0       0
   100.00     13.05      0.00     86.95      0.00    0.00    0.00    0.00
```

不带参的情况下，统计时间是 1s，统计的有各个模块的 ddr 使用情况以及对应占比。

8.10.3. gpio 状态设置与查询

使用 gpio 命令可以进行 gpio 状态查询。直接输入 gpio，可以得到相应的使用提示。比如想查看 PC8 脚的当前状态，则可以这么实现：

```
#gpio -g PC8
get PC8 status:
pin[PC8] funciton: 3
pin[PC8] data: 0
pin[PC8] dlevel: 2
pin[PC8] pull: 1
```

直接把 PC8 设置为输出高电平，则可以这么实现：

```
#gpio -s PC8 1 1 0 0
set PC8 function=1      #=1 表示配置为输出
set PC8 data=1          #数据是高电平，所以 data=1
set PC8 dlevel=0      #驱动等级
set PC8 pull=0     #不做内部上拉
```

8.10.4. 音频输入测试 tinycap_ahub_t5

不带参执行 tinycap_ahub_t5 可以得到示例用法

```
# tinycap_ahub_t5
Usage: tinycap_ahub_t5 file.wav [-aD ahub card] [-ad ahub device] [-D card] [-d device][-c channels] [-r rate] [-b bits] [-p period_size] [-n n_periods] [-t seconds]
```

Usage: for example: tinycap_ahub_t5 /tmp/data.wav -aD 1 -ad 0 -D 3 -d 0 -r 48000

执行 tinycap_ahub_t5 /tmp/data.wav -aD 1 -ad 0 -D 3 -d 0
则录制得到的音频会被存到/tmp/data.wav 中。

8.10.5. 音频播放测试 tinyplay

tinyplay /tmp/data.wav
硬件连接口如下图：

9.常见问题

9.1. 修改 nand Flash

9.1.1. 分区修改

由于公版 SDK 是基于 emmc 开发的，rootfs 分区较大，支持 nand flash 需要对 rootfs 根据实际选用的 nand flash 大小进行裁剪，并修改分区配置文件。

以 longan 为例，修改以下配置文件：

device/config/chips/t507/configs/demo2.0/longan/sys_partition.fex

1、减少各分区的 size，单位为扇区，一个扇区 512 字节。

2、分区大小最好保证为 16M 字节的整数倍。

分区大小计算方法：

128M nandflash

Rootfs_size=128-(7+4+16+16+16)=67M

7 表示 boot-resource 大小

4 表示 env 大小

16 表示 boot 大小

16 表示 MBR 大小

16 表示系统预留大小，为总大小的 8 分之 1，即 128/8=16

9.1.2. 存储类型

修改 device\config\chips\t507\configs\demo2.0\sys_config.fex 文件以下内容：

```
storage_type            = 1         ; -1 表示  emmc,1 表示 nand
```

9.2. 适配 DDR3

如果需要适配 DDR3 的参数，请参考下面 DDR(16x4 2G)参数的设定。将参数修改为对应的实际值，其他参数和 DDR 模板以及支持列表请联系 FAE。

device\config\chips\t507\configs\evb\sys_config.fex

```
[dram_para]
dram_clk            = 672
dram_type           = 3
dram_dx_odt         = 0x08080808
dram_dx_dri         = 0x0e0e0e0e
```

```
dram_ca_dri      = 0x0e0e
dram_odt_en      = 1
dram_para1       = 0x30FA
dram_para2       = 0x0000
dram_mr0         = 0x840
dram_mr1         = 0x4
dram_mr2         = 0x8
dram_mr3         = 0x0
dram_mr4         = 0x0
dram_mr5         = 0x0
dram_mr6         = 0x0
dram_mr11        = 0x0
dram_mr12        = 0x0
dram_mr13        = 0x0
dram_mr14        = 0x0
dram_mr16        = 0x0
dram_mr17        = 0x0
dram_mr22        = 0x0
dram_tpr0        = 0x0
dram_tpr1        = 0x0
dram_tpr2        = 0x0
dram_tpr3        = 0x0
dram_tpr6        = 0x33808080
dram_tpr10       = 0x00f93336
dram_tpr11       = 0x0
dram_tpr12       = 0x0
dram_tpr13       = 0x40
```

9.3. DE invalid address

部分情况下，出现 DE 报错，提示地址无效：

```
[ 1389.467854] L2 PageTable Invalid
[ 1389.471496] 0xff500000 is not mapped!
[ 1389.475611] DE invalid address: 0xff500000, data:0x0, id:0x1
```

该问题是因为应用退出，送显的内存已经释放，但是显示图层没有关闭。因此 DE 一直在使用该无效的地址，从而报错。重新关闭对应图层即可。参考 "platform\framework\auto\sdk_demo\csitest\csi_test_mplane_usrptr.c"中的 disp_disable()

```
static int disp_disable(void)
{
#if display_frame
    int ret;
```

```
unsigned long arg[6];
struct disp_layer_config disp;

/* release memory && clear layer */
arg[0] = 0;
arg[1] = 0;
arg[2] = 0;
arg[3] = 0;
ioctl(dev.layer_info.dispfh, DISP_LAYER_DISABLE, (void *)arg);

/*close channel 0*/
memset(&disp, 0, sizeof(disp_layer_config));
disp.channel = 0;
disp.layer_id = 0;
disp.enable = 0;
arg[0] = dev.layer_info.screen_id;
arg[1] = (unsigned long)&disp;
arg[2] = 1;
arg[3] = 0;
ret = ioctl(dev.layer_info.dispfh, DISP_LAYER_SET_CONFIG, (void *)arg);
if (ret != 0)
    printf("disp_disable:disp_set_addr fail to set layer info\n");
```

这个是关闭 screen=0,channel=0,layer=0 的，图层信息如下加粗的部分：

```
# cat /sys/class/disp/disp/attr/sys
screen 0:
de_rate 696000000 hz, ref_fps:61
mgr0: 1280x800 fmt[rgb] cs[0x204] range[full] eotf[0x4] bits[8bits] err[0] force_sync[0] unblank direct_show[false]
dmabuf: cache[0] cache max[0] umap skip[0] overflow[0]
        lcd output          backlight( 50)    fps:61.4              1280x 800
        err:1     skip:17 irq:25338           vsync:25338         vsync_skip:0
    BUF       enable ch[1] lyr[0] z[5] prem[N] a[pixel      0] fmt[    0] fb[1280, 800;1280, 800;1280, 800] crop[      0,
0,1280, 800] frame[      0,      0,1280, 800] addr[ff800000,             0,             0] flags[0x            0] trd[0,0]
depth[ 0] transf[0]
    BUF       enable ch[2] lyr[0] z[3] prem[N] a[pixel      0] fmt[    0] fb[1280, 800;1280, 800;1280, 800] crop[      0,
0,1280, 800] frame[      0,      0,1280, 800] addr[ff000000,             0,             0] flags[0x            0] trd[0,0]
depth[ 0] transf[0]
    BUF       enable ch[3] lyr[0] z[1] prem[N] a[pixel      0] fmt[    0] fb[1280, 800;1280, 800;1280, 800] crop[      0,
0,1280, 800] frame[      0,      0,1280, 800] addr[fe800000,             0,             0] flags[0x            0] trd[0,0]
depth[ 0] transf[0]
```

如果想关闭上面的第二个图层，则需要改成

disp.channel = 2;

disp.layer_id = 0;

即可。

9.4. 开机 Logo 盖住了摄像头画面

logo 位于 fb0。由于 fb0 的 zorder 大于摄像头图层值，如果 fb 存在数据，则会覆盖在摄像头画面上。执行一下 fbinit 命令，把 fb0 即可。

9.5. adb 概率性断开

手动在串口终端执行一下/etc/adb_start.sh

```
# /etc/adb_start.sh
[     43.804868] android_work: sent uevent USB_STATE=DISCONNECTED
[     43.811420] android_work: did not send uevent (0 0                    (null))
device_chose finished!
[     43.813057]
[     43.813057] rmmod_device_driver
[     43.813057]
install_listener('tcp:5037','*smartsocket*')
[     43.813121]
[     43.813121] insmod_device_driver
[     43.813121]
# [     43.819492] read descriptors
[     43.819505] read strings
[     44.599238] android_work: sent uevent USB_STATE=CONNECTED
[     44.730465] configfs-gadget gadget: high-speed config #1: c
[     44.737146] android_work: sent uevent USB_STATE=CONFIGURED
```

看到最后一句加粗字样，就说明 adb 从新连上了，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。

10. 附录

10.1. 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

11. Declaration

This is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind， including fitness for any particular application.