**Rokid Kamino18 Universal**

**Install Guide**

**STRICTLY PROPRIETARY and CONFIDENTIAL**

The material in this document is the intellectual property of **Rokid**. It is provided solely for information. You may not reproduce this document in whole or in part by any means. While every care has been taken in the preparation of this document, **Rokid** accepts no liability for any consequences of its use. Our products are under continual improvement and we reserve the right to change their specification without notice. This document contains information on a product under development. **Rokid** reserves the right to change or this product without notice.

芋头科技（杭州）有限公司

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 撰写/修改人 | 日期 | 修改记录 |
| V1.0 | 朱斌 | 2018/6/9 | Initial version |
| V2.0 | 朱斌 | 2018/7/13 | Modify for DEV3.0 board |
| V2.1 | 朱斌 | 2018/7/28 | Amend for vsp compiling tool chain |
| V2.2 | 张鹏 | 2018/8/2 | Add new lib for openwrt compile |
| V2.3 | 朱斌 | 2018/8/9 | Modify for kernel uboot compile |
| V2.4 | 朱斌 | 2018/9/8 | Modify for image flash tool |
| V2.5 | 朱斌 | 2018/10/25 | To match YodaOS |

**目录**

[1 概述 1](#_Toc528253480)

[1.1 目的 1](#_Toc528253481)

[1.2 适用范围 1](#_Toc528253482)

[2 Kamino18平台简介 2](#_Toc528253483)

[2.1 芯片框图 2](#_Toc528253484)

[2.2 智能语音方案设计简介 3](#_Toc528253485)

[2.3 硬件开发板说明 4](#_Toc528253486)

[3 软件架构与编译方法 7](#_Toc528253487)

[3.1 代码目录 7](#_Toc528253488)

[3.2 编译环境配置 7](#_Toc528253489)

[3.2.1 Openwrt编译环境搭建 8](#_Toc528253490)

[3.2.2 自带编译工具链环境搭建 8](#_Toc528253491)

[3.2.3 MCU编译环境搭建 8](#_Toc528253492)

[3.3 代码编译 9](#_Toc528253493)

[3.3.1 Openwrt编译 9](#_Toc528253494)

[3.3.2 Kernel编译 10](#_Toc528253495)

[3.3.3 Uboot编译 10](#_Toc528253496)

[3.3.4 VSP编译 10](#_Toc528253497)

[4 PINMUX及GPIO设置 11](#_Toc528253498)

[4.1 MCU域PINMUX设置 11](#_Toc528253499)

[4.2 ARM域PINMUX设置 12](#_Toc528253500)

[4.3 GPIO设置 14](#_Toc528253501)

[4.3.1 GPIO 规划 14](#_Toc528253502)

[4.3.2 内核层GPIO使用方法 15](#_Toc528253503)

[4.3.3 MCU侧GPIO使用方法 16](#_Toc528253504)

[4.3.4 应用层GPIO使用方法 16](#_Toc528253505)

[5 Flash分区及文件系统管理 18](#_Toc528253506)

[5.1 分区设置 18](#_Toc528253507)

[5.2 文件系统配置 20](#_Toc528253508)

[6 镜像烧写 21](#_Toc528253509)

[6.1 镜像位置 21](#_Toc528253510)

[6.2 Linux烧写工具使用 22](#_Toc528253511)

[6.3 Windows烧写工具使用 23](#_Toc528253512)

[6.4 工厂烧写工具使用 24](#_Toc528253513)

[7 SN与Seed烧写 26](#_Toc528253514)

[8 配网验证 27](#_Toc528253515)

[8.1 BT配网 27](#_Toc528253516)

[9 常见问题 28](#_Toc528253517)

# 概述

## 目的

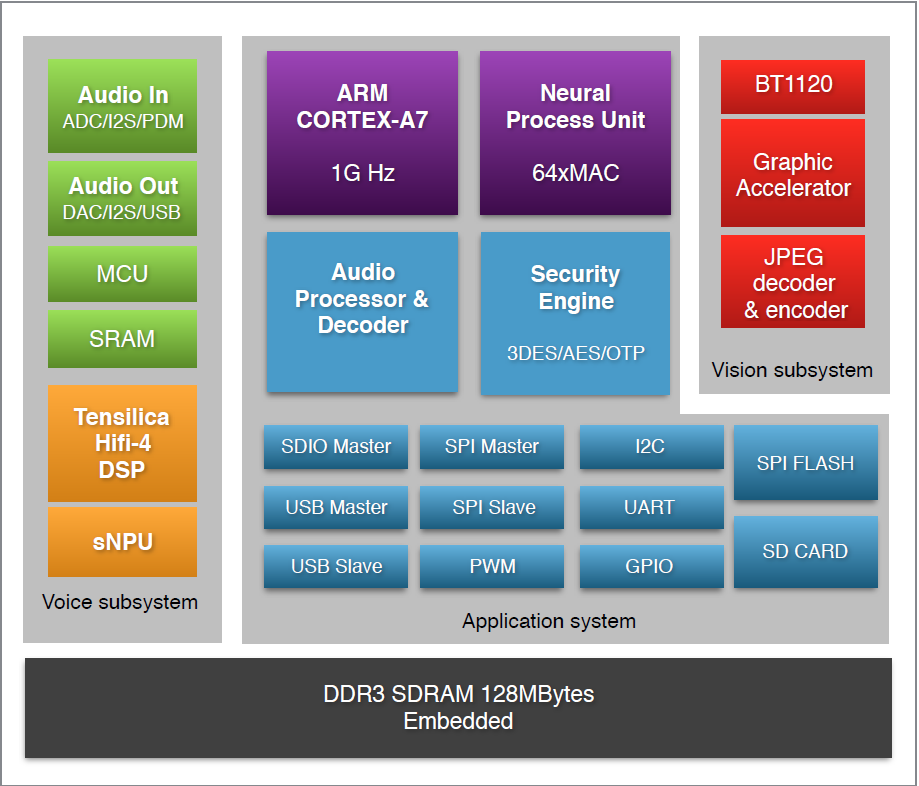
本文档的目的在于帮助使用Rokid Kamino18 AI平台的厂商, 能够快速掌握平台代码的配置与编译，镜像的烧写等方法。

## 适用范围

适用于快速导入Rokid Kamino18 AI平台设计生产智能语音产品。

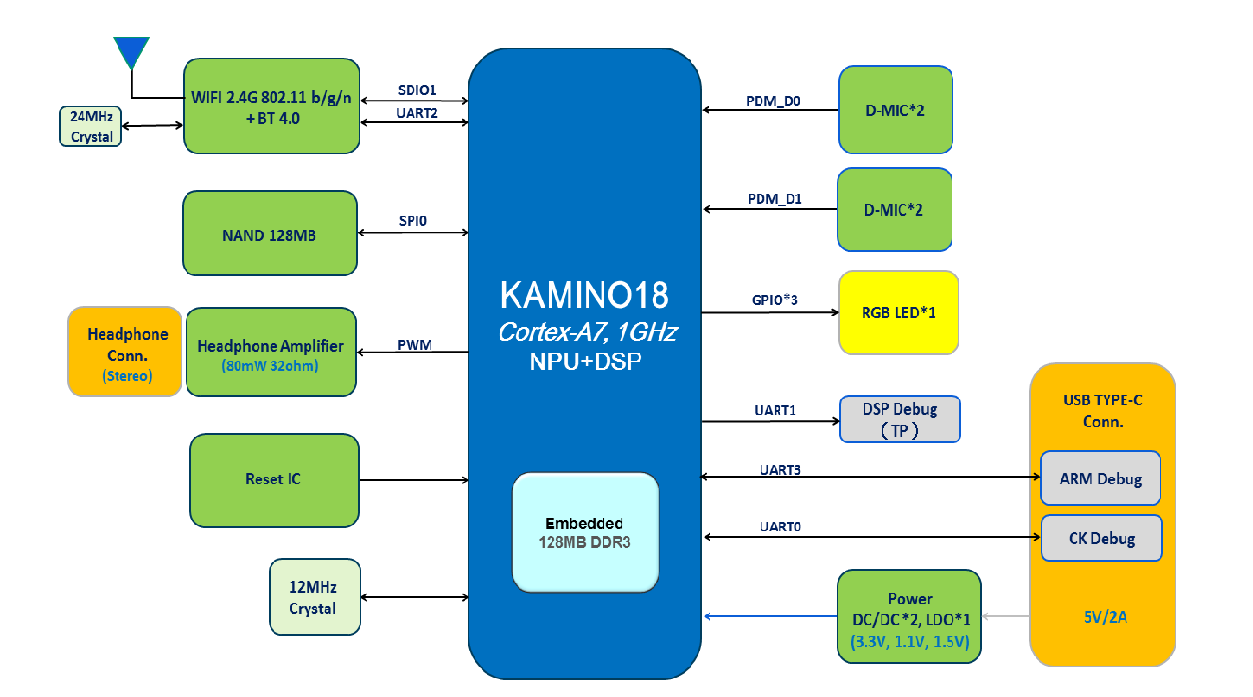
# Kamino18平台简介

## 芯片框图



* 支持2/4/6/8路麦克风阵列
* 集成8通道ADC/I2S/PDM音频输入接口，支持模拟麦和数字麦
* 集成DAC/I2S音频输出接口，支持内部audio codec输出或外接audio codec输出
* 采用单MCU和单Cortex-A7的异步处理器架构，便于实现低功耗离线语音激活功能
* 内置DSP，支持降噪、AEC、Beam-forming、去混响等算法
* 内置NPU和SNPU, 主NPU由Cortex-A7控制，SNPU由MCU控制
* 内置BT1120接口，支持YUV格式的camera数据输入
* 内置SPI/I2C/USB/UART/PWM/SDIO等connectivity接口，支持集成WIFI/BT/LED等外围器件
* 内存配置：128MB SPI NAND Flash+内置128MB DDR

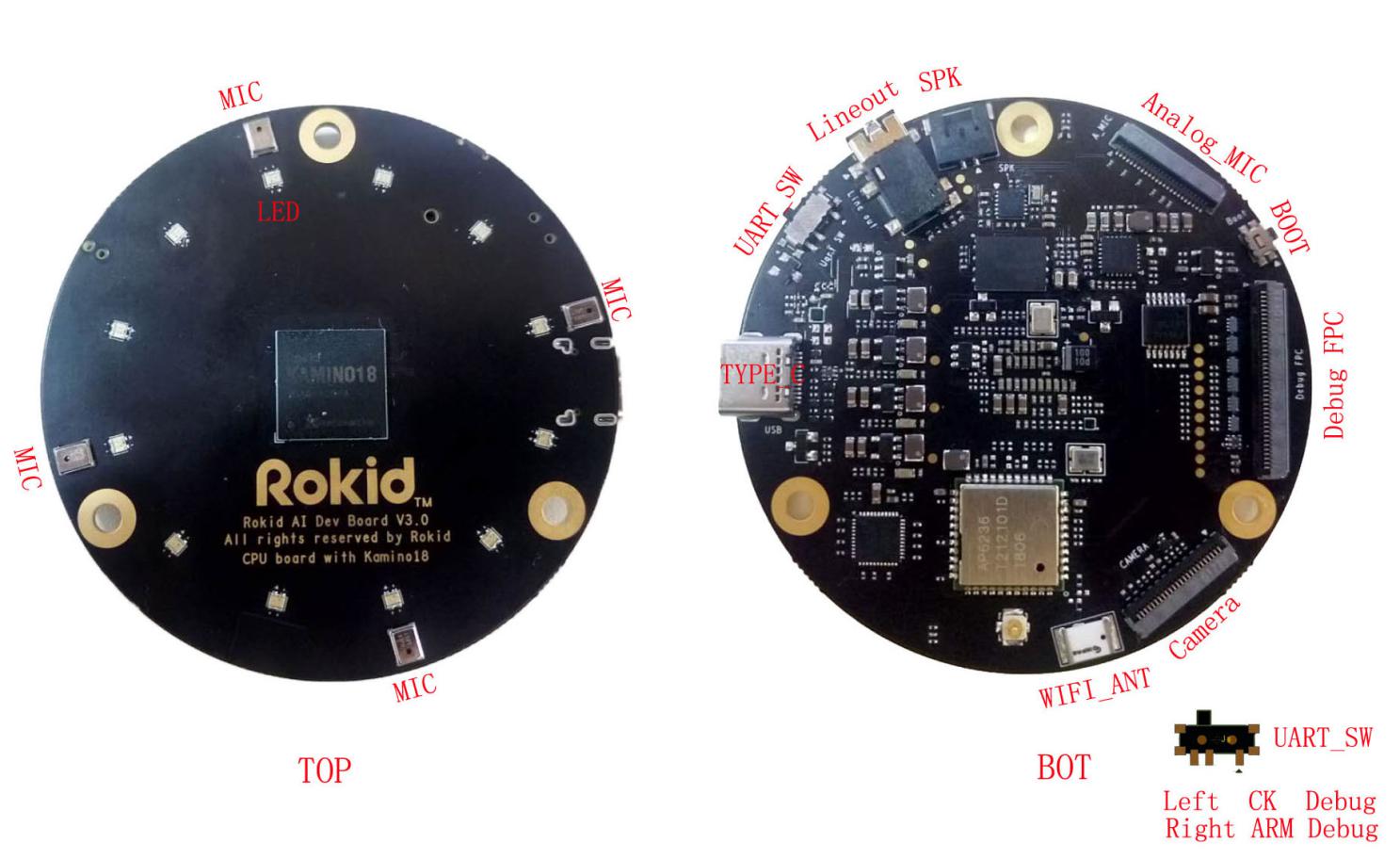
## 智能语音方案设计简介



* 电源：锂电池（便携模式）, 5V/2A,5V/1A AC/DC电源适配器（普通模式）或PC USB接口供电
* 存储器：SPI NAND Flash单颗1GBit~2GBit
* 音频：Audio Right/Left，模拟音频输出
* MIC阵列：支持最高8路模拟MIC输入，支持单端和差分模式，每路自带BIAS电源
* Connectivity：支持SDIO接入WIFI模块，UART接入BT模块
* USB：一路USB2.0 HOST接口，一路USB2.0 Slave接口
* 操作系统：YodaOS（基于Linux内核）
* 云平台：默认对接Rokid语音云平台，也可对接第三方语音云平台
* 低功耗设计：支持Voice subsystem(MCU+DSP)上运行的低功耗离线语音激活模型

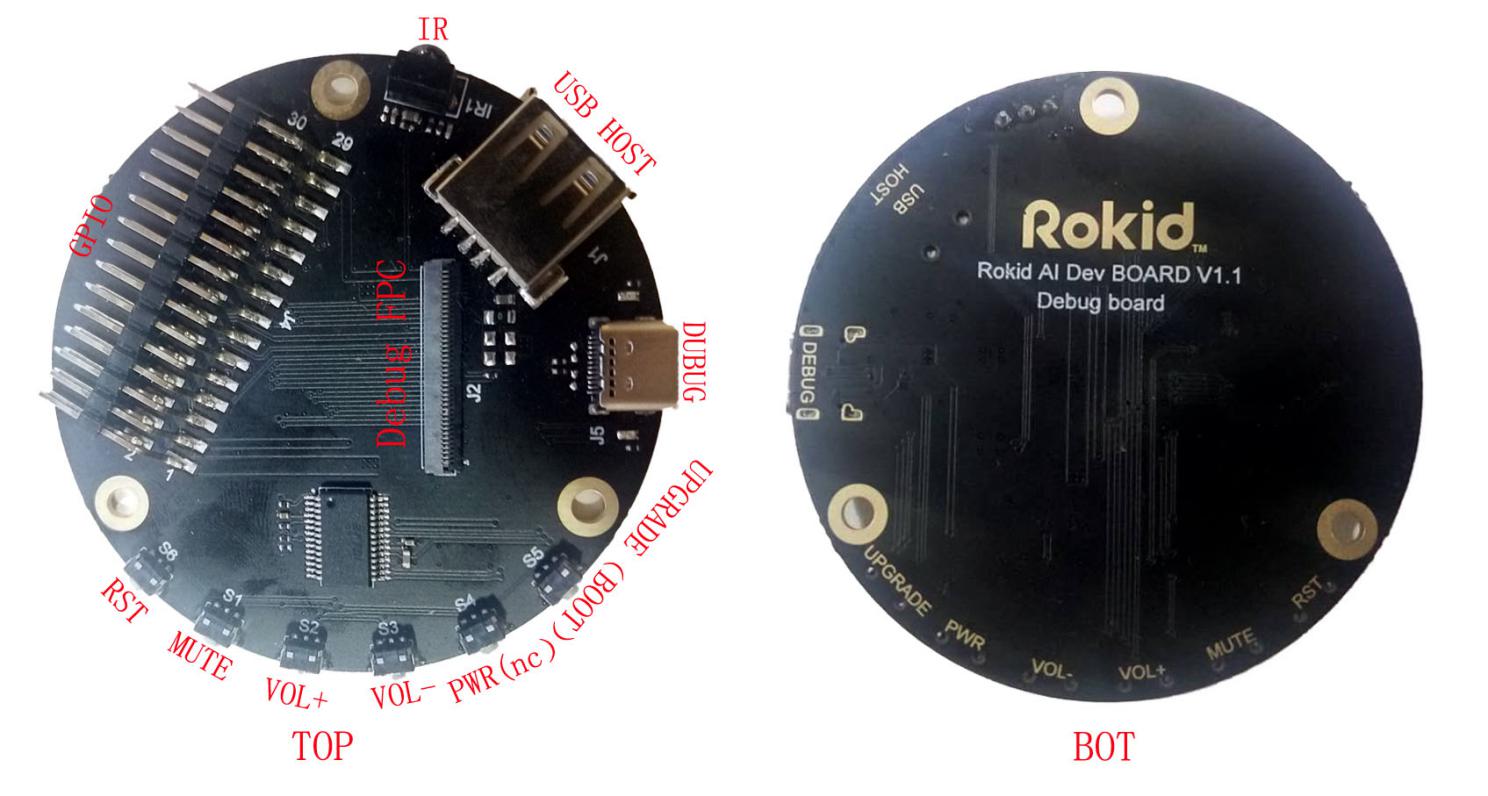
## 硬件开发板说明

Kamino18 Dev平台是面向智能语音应用而设计的硬件参考方案，分为主板和调试子板两部分。如下图所示：



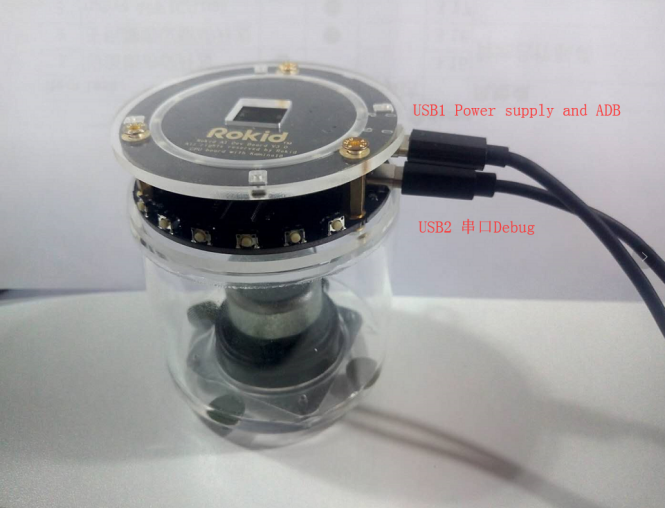
**主板示意图**

1. Lineout 立体声耳机输出口
2. SPK Speaker 3.2w 4欧姆
3. Analog\_MIC 支持6路模拟MIC
4. BOOT 升级按键
5. Debug FPC 连接Debug板
6. Camera 2M Camera(OV2640)
7. WIFI ANT 板载陶瓷天线，可外接天线需要手工选贴电阻。
8. TYPE\_C 内含ADB debug,CK,ARM,调试串口（TTL电平）
9. UART\_SW SW 连接器，SW拔到左侧选择CK调试口,SW拔到右侧选择ARM调试口.
10. LED 12PCS RGB LED
11. MIC 4PCS PDM MIC

****

**调试子板示意图**

1. IR 红外接收口
2. USB HOST 可以外面USB设备
3. DEBUG USB转串口调试口,USB线不提供电源。
4. UPGRADE(BOOT) 按住按键写Image
5. PWR(NC) 此按键为预留
6. VOL- 音量减
7. VOL+ 音量加
8. MUTE 麦克静音
9. RST 系统复位
10. GPIO 外接GPIO 型号定义参考原理图



**USB线连接说明图**

1. USB1 power supply and ADB,
2. USB2 串口Debug,一端接Debug一端接PC,需要在PC端按装FT232驱动。UART SW

Left for CK Debug ,Right for ARM Debug.USB2不能给开发板提供电源。

**说明：**

* 主板上有4个PDM 接口的MEMS MIC, 12个RGB 3色LED；
* 主板上的TYPEC口， 连接信号包含 USB信号，CK UART信号，ARM UART信号，BOOT\_KEY信号。按下BOOT\_KEY，可用于烧写镜像。也可以通过标准TPYEC线连接电脑用于ADB调试和供电，建议最好用适配器给开发板供电，因AUDIO SPK功率比较大；如果音量调节比较小可以用USB供电。
* 调试子板上另一个TYPEC口，只提供USB信号不供电。通过标准TYPEC线连接PC；

# 软件架构与编译方法

## 代码目录

Kamino18 Dev平台采用基于Linux内核的YodaOS操作系统，其代码目录如下图所示：

├── 3rd

├── applications

├── build.sh -> /home/zhubin/kamino18/products/rokid/common/build.sh

├── check\_by\_jenkins

├── envsetup.sh -> /home/zhubin/kamino18/products/rokid/common/envsetup.sh

├── frameworks

├── hardware

├── kernel

├── openwrt

├── products

├── toolchains

├── tools

├── uboot

└── vendor

* 3rd: 第三方库，第三方sdk等
* applications: 本地应用
* build.sh: 统一编译脚本
* envsetup.sh: 设置toolchains的环境变量脚本
* frameworks: 应用框架，native 服务、命令、库，和对第三方的抽象等
* hardware: HAL (hardware abstract layer)，包括主芯片和外设的硬件抽象
* kernel: Kamino18 DEV平台Linux内核代码
* openwrt: YodaOS编译环境代码
* toolchains:自带工具链目录
* tools: YodaOS开发调试测试工具目录
* uboot: Kamino18 DEV平台uboot代码
* vendor: 第三方平台供应商驱动或sdk代码

## 编译环境配置

因为Kamino18平台采用MCU/ARM的异步架构，开发者需要安装arm和MCU交叉编译工具链。

### Openwrt编译环境搭建

整个Kamino18平台的YodaOS是基于openwrt进行编译的，通过openwrt我们可以得到文件系统，交叉编译工具链等。请在进行openwrt编译前，确保如下工具包已经在ubuntu环境上安装完成。

* sudo apt-get install build-essential subversion libncurses5-dev zlib1g-dev gawk gcc-multilib flex git-core gettext libssl-dev unzip texinfo
* sudo apt-get install device-tree-compiler
* sudo apt-get install gcc-4.7-arm-linux-gnueabihf
* sudo apt-get install libusb-1.0-0-dev

### 自带编译工具链环境搭建

Kamino18平台代码中包含自带工具链，如下图：

├── toolchains

│   ├── arm

│   ├── csky

│   └── scripts

自带工具链目录包含arm（64位ubuntu-16.04编译服务器上使用）和csky(用于编译MCU)两个交叉编译工具链。为了使用这两个工具链，需要设置环境变量, 在代码包根目录运行命令如下：

* source env.sh
* export ARCH=arm
* export CROSS\_COMPILE=arm-openwrt-linux-gnueabi-

### MCU编译环境搭建

Kamino18的VSP子系统由MCU+DSP组成，其中MCU是控制部分，DSP处理算法部分。MCU编译工具链（toolchains/csky/）的依赖环境如下：

* sudo dpkg --add-architecture i386
* sudo apt-get update
* sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 libx11-6:i386 gtk2-engines:i386 lib32ncurses5 lib32z1 libxtst6:i386 libgtk2.0-0:i386 lib32ncurses5 libcanberra-gtk3-0:i386

## 代码编译

### Openwrt编译

作为Kamino18 YodaOS的整体编译环境，使用openwrt可以编译出系统正常运行所需的主要image如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **镜像名字** | **镜像运行位置** | **镜像说明** | **镜像生成位置** |
| mcu.bin | MCU | The image which run on MCU | $(K18DIR)/openwrt/bin/leo-k18-universal-glibc/mcu.bin |
| u-boot-spl.bin | ARM | Secondary program loader, which is charge of DDR initialization and download u-boot.img into DDR to run | $(K18DIR)/openwrt/bin/leo-k18-universal-glibc/uboot-leo-k18-universal/openwrt-leo-k18-universal-u-boot-spl.bin |
| u-boot.img | ARM | BootLoader download dtb, kernel image and rootfs image into DDR and jump to kernel offset to run | $(K18DIR)/openwrt/bin/leo-k18-universal -glibc/uboot-leo-k18-universal /openwrt- leo-k18-universal - u-boot.img |
| dtb | ARM | Device tree blob for kernel image | $(K18DIR)/openwrt/bin/leo-k18-universal -glibc/openwrt-leo-k18-universal.dtb |
| zImage | ARM | Linux kernel image | $(K18DIR)/openwrt/bin/leo-k18-universal -glibc/openwrt- leo-k18-universal-zImage |
| ubi.img | ARM | Rootfs | $(K18DIR)/openwrt/bin/leo-k18-universal -glibc/openwrt-leo-k18-universal-ubi.img |

有两种整体编译Kamino18 YodaOS镜像的方法如下：

1. 系统根目录下运行

* ./build.sh –p leo\_k18\_universal\_node -n openwrt-leo-k18-universal -f leo-k18-universal -j12 -r

2. 为了获取更多的编译信息，可以进入openwrt进行编译，命令如下：

* cd openwrt
* cp configs/leo\_k18\_universal\_node\_defconfig .config
* make defconfig
* make -j1 V=99

编译结果存放在$(K18DIR)/openwrt/bin目录下。

修改openwrt package配置：

* make menuconfig

修改kernel配置：

* make kernel\_menuconfig

openwrt整个编译后，如需编译单个package用如下命令（详细说明参考$(K18DIR)/openwrt/README.md）：

* make package/some\_package/clean
* make package/some\_package/compile V=99

### Kernel编译

通过自带工具链，开发者可以独立编译kernel代码，命令如下：

* cd openwrt
* make kernel\_menuconfig
* make target/linux/clean
* make target/linux/compile
* make target/linux/install

kernel编译生成的dtb和zImage文件在$(K18DIR)/kernel/arch/arm/boot/目录中。

### Uboot编译

通过自带工具链，开发者可以独立编译uboot代码，命令如下：

* cd openwrt
* make package/uboot-leo/clean
* make package/uboot-leo/ compile 或 make package/uboot-leo/install

uboot编译生成的u-boot-spl.bin在$(K18DIR)/uboot/spl/目录中, u-boot.img在$(K18DIR)/uboot/目录中。

### VSP编译

VSP目录中包含MCU和VSP驱动的源文件（DSP为非开源算法代码，编译方法本文略去），其编译方法如下：

* cd $(K18DIR)/hardware/nationalchip/gx8010/vsp
* make k18\_universal\_defconfig
* make

编译生成的镜像如下表所示：

|  |  |  |
| --- | --- | --- |
| **镜像名字** | **镜像说明** | **镜像生成位置** |
| mcu\_nand.bin | The mcu image which is flashed into spi nand flash | $(K18DIR)/hardware/nationalchip/gx8010/vsp/output/mcu\_nand.bin |
| mcu\_nor.bin | The mcu image which is flashed into spi nor flash | $(K18DIR)/hardware/nationalchip/gx8010/vsp/output/mcu\_nor.bin |
| vsp.ko | The vsp driver module which is installed into kernel | $(K18DIR)/ hardware/nationalchip/gx8010/vsp/output/vsp.ko |

# PINMUX及GPIO设置

Kamino18的PINMUX分为MCU和ARM两个域。其中ARM域中的PINMUX在系统深度休眠时状态会复位，系统从深度休眠唤醒时PINMUX会在u-boot-spl.bin中重置（具体在4.1中详述）。

## MCU域PINMUX设置

MCU域中有34个pin（id）可以配置，每个pin对应的function如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | **function0** | **function1** | **function2** | **function3** | **function4** |
| 1 | POWERDOWN | PD1PORT01 |  |  |  |
| 2 | UART0RX | PD1PORT02 |  |  |  |
| 3 | UART0TX | PD1PORT03 |  |  |  |
| 4 | OTPAVDDEN | PD1PORT04 |  |  |  |
| 5 | SDBGTDI | DDBGTDI | SNDBGTDI | PD1PORT05 |  |
| 6 | SDBGTDO | DDBGTDO | SNDBGTDO | PD1PORT06 |  |
| 7 | SDBGTMS | DDBGTMS | SNDBGTMS | PCM1INBCLK | PD1PORT07 |
| 8 | SDBGTCK | DDBGTCK | SNDBGTCK | PCM1INLRCK | PD1PORT08 |
| 9 | SDBGTRST | DDBGTRST | SNBGTRST | PCM1INDAT0 | PD1PORT09 |
| 11 | PCM1INBCLK | PD1PORT11 |  |  |  |
| 12 | PCM1INLRCK | PD1PORT12 |  |  |  |
| 13 | PCM1INDAT0 | PD1PORT13 |  |  |  |
| 14 | PCMOUTMCLK | DUARTTX | SNUARTTX | PD1PORT14 |  |
| 15 | PCMOUTDAT0 | SPDIF | PD1PORT15 |  |  |
| 16 | PCMOUTLRCK | PD1PORT16 |  |  |  |
| 17 | PCMOUTBCLK | PD1PORT17 |  |  |  |
| 18 | UART1RX | PD1PORT18 |  |  |  |
| 19 | UART1TX | PD1PORT19 |  |  |  |
| 20 | DDBGTDI | SNDBGTDI | PD1PORT20 |  |  |
| 21 | DDBGTDO | SNDBGTDO | PD1PORT21 |  |  |
| 22 | DDBGTMS | SNDBGTMS | PD1PORT22 |  |  |
| 23 | DDBGTCK | SNDBGTCK | PD1PORT23 |  |  |
| 24 | DDBGTRST | SNDBGTRST | PD1PORT24 |  |  |
| 25 | DUARTTX | SNUARTTX | PD1PORT25 |  |  |
| 26 | SDA0 | PD1PORT26 |  |  |  |
| 27 | SCL0 | PD1PORT27 |  |  |  |
| 28 | SDA1 | PD1PORT28 |  |  |  |
| 29 | SCL1 | PD1PORT29 |  |  |  |
| 30 | PCM0INDAT1 | PDMDAT3 | PD1PORT30 |  |  |
| 31 | PCM0INDAT0 | PDMDAT2 | PD1PORT31 |  |  |
| 32 | PCM0INMCLK | PDMDAT1 | PD1PORT32 |  |  |
| 33 | PCM0INLRCK | PDMDAT0 | PCM0OUTLRCK | PD1PORT33 |  |
| 34 | PCM0INBCLK | PDMCLK | PCM0OUTBCLK | PD1PORT34 |  |
| 35 | IR | PD1PORT35 |  |  |  |

MCU域的PINMUX设置统一在如下的数组中：

$(K18DIR)/vsp/mcu/board/nationalchip/leo\_k18\_universal/misc\_board.c

static const PADMUX\_PAD\_CONFIG \_pad\_configs[] = {

/\* id| func // function0 | function1 | function2 | function3 | function4 \*/

{ 1, 0}, // POWERDOWN | PD1PORT01 |

{ 2, 0}, // UART0RX | PD1PORT02 |

{ 3, 0}, // UART0TX | PD1PORT03 |

…

{33, 1}, // PCM0INLRCK | PDMDAT0 | PCM0OUTLRCK | PD1PORT33

{34, 1}, // PCM0INBCLK | PDMCLK | PCM0OUTBCLK | PD1PORT34

{35, 1}, // IR | PD1PORT35

};

配置表中的第一列表示芯片的port口，第二列表示port口的功能，根据硬件给出的管脚复用表，对应修改每个port口的功能即可。

## ARM域PINMUX设置

ARM域中有52个pin（id）可以配置，每个pin对应的function如下表所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **id** | **function0** | **function1** | **function2** | **function3** | **function4** | **function5** |
| 0 | BTDAT00 | PD2PORT00 |  |  |  |  |
| 1 | BTDAT01 | PD2PORT01 |  |  |  |  |
| 2 | BTDAT02 | PD2PORT02 |  |  |  |  |
| 3 | BTDAT03 | PD2PORT03 |  |  |  |  |
| 4 | BTCLKIN | PD2PORT04 |  |  |  |  |
| 5 | BTRESET | PD2PORT05 |  |  |  |  |
| 6 | BTCLKVSYNC | PD2PORT06 |  |  |  |  |
| 7 | BTCLKHREF | PD2PORT07 |  |  |  |  |
| 8 | BTCLKOUT | PD2PORT08 |  |  |  |  |
| 9 | BTDAT04 | SDA2 | SPI1SCK | PD2PORT09 |  |  |
| 10 | BTDAT05 | SCL2 | SPI1MOSI | PD2PORT10 |  |  |
| 11 | BTDAT06 | UART2RX | SPI1CSn | PD2PORT11 |  |  |
| 12 | BTDAT07 | UART2TX | SPI1MISO | NUARTTX | AUARTTX | PD2PORT12 |
| 13 | BTDAT08 | PD2PORT13 |  |  |  |  |
| 14 | BTDAT09 | PD2PORT14 |  |  |  |  |
| 15 | BTDAT10 | PD2PORT15 |  |  |  |  |
| 16 | BTDAT11 | PD2PORT16 |  |  |  |  |
| 17 | BTDAT12 | PD2PORT17 |  |  |  |  |
| 18 | BTDAT13 | PD2PORT18 |  |  |  |  |
| 19 | BTDAT14 | PD2PORT19 |  |  |  |  |
| 20 | BTDAT15 | PD2PORT20 |  |  |  |  |
| 21 | UART3RX | SD1CDn | SDA3 | PD2PORT21 |  |  |
| 22 | UART3TX | SD1DAT1 | SCL3 | PD2PORT22 |  |  |
| 23 | DBGTDI | SD1DAT0 | SPI2SCK | PD2PORT23 |  |  |
| 24 | DBGTDO | SD1CLK | SPI2MOSI | PD2PORT24 |  |  |
| 25 | DBGTMS | SD1CMD | SPI2CSn | PD2PORT25 |  |  |
| 26 | DBGTCK | SD1DAT3 | SPI2MISO | PD2PORT26 |  |  |
| 27 | DBGTRST | SD1DAT2 | PD2PORT27 |  |  |  |
| 28 | UART2RX | SDA2 | PD2PORT28 |  |  |  |
| 29 | UART2TX | SCL2 | PD2PORT29 |  |  |  |
| 30 | SPI1SCK | NDBGTDI | ADBGTDI | PD2PORT30 |  |  |
| 31 | SPI1MOSI | NDBGTDO | ADBGTDO | PD2PORT31 |  |  |
| 32 | SPI1CSn | NDBGTMS | ADBGTMS | PD2PORT32 |  |  |
| 33 | SPI1MISO | NDBGTCK | ADBGTCK | PD2PORT33 |  |  |
| 34 | SPI2SCK | NDBGTRST | ADBGTRST | PD2PORT34 |  |  |
| 35 | SPI2MOSI | PD2PORT35 |  |  |  |  |
| 36 | SPI2CSn | PD2PORT36 |  |  |  |  |
| 37 | SPI2MISO | PD2PORT37 |  |  |  |  |
| 38 | SD0CDn | PD2PORT38 |  |  |  |  |
| 39 | SD0DAT1 | PD2PORT39 |  |  |  |  |
| 40 | SD0DAT0 | PD2PORT40 |  |  |  |  |
| 41 | SD0CLK | PD2PORT41 |  |  |  |  |
| 42 | SD0CMD | PD2PORT42 |  |  |  |  |
| 43 | SD0DAT3 | PD2PORT43 |  |  |  |  |
| 44 | SD0DAT2 | PD2PORT44 |  |  |  |  |
| 45 | SD1CDn | PD2PORT45 |  |  |  |  |
| 46 | SD1DAT1 | PD2PORT46 |  |  |  |  |
| 47 | SD1DAT0 | PD2PORT47 |  |  |  |  |
| 48 | SD1CLK | PD2PORT48 |  |  |  |  |
| 49 | SD1CMD | PD2PORT49 |  |  |  |  |
| 50 | SD1DAT3 | PD2PORT50 |  |  |  |  |
| 51 | SD1DAT2 | PD2PORT51 |  |  |  |  |

ARM域的PINMUX设置统一在如下的数组中：

$(K18DIR)/uboot/board/nationalchip/leo\_k18\_universal/pinmux.c

static const struct pin\_config pin\_table[] = {

/\* id |func function0 | function1 | function2 | function3 | function4 | function5 \*/

{ 0, 1}, // BTDAT00 | PD2PORT00

{ 1, 1}, // BTDAT01 | PD2PORT01

{ 2, 1}, // BTDAT02 | PD2PORT02

{ 3, 1}, // BTDAT03 | PD2PORT03

…

{49, 0}, // SD1CMD | PD2PORT49

{50, 0}, // SD1DAT3 | PD2PORT50

{51, 0}, // SD1DAT2 | PD2PORT51

};

配置表中的第一列表示芯片的port口，第二列表示port口的功能，根据硬件给出的管脚复用表，对应修改每个port口的功能即可。

## GPIO设置

### GPIO 规划

PD1开头的port对应MCU域的port，PD2开头的port对应ARM域的port, 其对应的gpio组如下：

* PD2PORT0 - PD2PORT31 对应 gpa0 组
* PD2PORT32 - PD2PORT51 对应 gpa1 组
* PD1PORT0 - PD1PORT31 对应ck\_gpa0 组
* PD1PORT32 - PD1PORT35 对应ck\_gpa1 组

目前不提供ck的gpio中断在A7的支持，所有的GPIO都可以配置为中断模式，但要注意管脚复用配置正确，为GPIO模式。GPIO number的数字就是PDXPORTXX的数字 % 32。

### 内核层GPIO使用方法

控制ARM域GPIO设备树节点例子如下：

test: test{

compatible = "nationalchip,LEO\_A7-test";

test1-gpio = <&gpa0 1 GPIO\_ACTIVE\_HIGH>;

};

控制MCU域GPIO设备树节点例子如下：

test: test{

compatible = "nationalchip,LEO\_A7-test";

test1-gpio = <&ck\_gpa0 10 GPIO\_ACTIVE\_HIGH>;

};

Note：

这里GPIO\_ACTIVE\_HIGH代表gpio输出极性控制为正output1实际输出高电平，output0实际输出低电平），GPIO\_ACTIVE\_LOW代表gpio输出极性控制为负（output1实际输出低电平，output0实际输出高电平）

驱动代码调用gpio接口如下：

* desc = devm\_gpiod\_get(&pdev->dev, "test1", GPIOD\_OUT\_LOW); //申请test1-gpio这个属性对应的GPIO口描述符，并设置为输出模式，输出低电平
* gpiod\_direction\_input(desc); //设置desc描述符对应的GPIO口为输入模式
* gpiod\_to\_irq(desc) //获取此gpio对应的中断号，中断号要注意检查，大于0为有效的
* gpiod\_direction\_output(desc,1); //设置desc描述符对应的GPIO口为输出模式，输出高电平
* gpiod\_set\_value(desc,0); //设置desc描述符对应的GPIO口输出值为0
* gpio\_level = gpiod\_get\_value(desc); //得到desc描述符对应的GPIO口的电平
* devm\_gpiod\_put(&test\_pdev->dev, desc); //释放desc描述符对应的GPIO口，这样别人才能申请这个GPIO口

在kernel里只提供ARM gpio的pwm功能，MCU的gpio不支持。pwm0对应gpa0组,pwm1对应gpa1组, port换算关系为port 的数字就是PDXPORTXX的数字 % 32，驱动代码调用pwm接口如下：

* struct pwm\_device \*devm\_pwm\_get(struct device \*dev, const char \*con\_id) //获取pwm句柄
* int pwm\_config(struct pwm\_device \*pwm, int duty\_ns, int period\_ns) //配置pwm
* int pwm\_enable(struct pwm\_device \*pwm) //开启pwm
* void pwm\_disable(struct pwm\_device \*pwm) //关闭pwm

设备树节点例子：

控制PD2PORT26脚为

test:test{

compatible = "nationalchip,LEO\_A7-test";

pwms = <&pwm0 26 5000000>;

};

### MCU侧GPIO使用方法

MCU端同样提供了gpio的支持方法，如下：

* GPIO\_DIRECTION GpioGetDirection(unsigned int port);
* int GpioSetDirection(unsigned int port, GPIO\_DIRECTION direction);
* GPIO\_LEVEL GpioGetLevel(unsigned int port);
* int GpioSetLevel(unsigned int port, GPIO\_LEVEL level);
* int GpioEnableTrigger(unsigned int port, GPIO\_TRIGGER\_EDGE edge, GPIO\_CALLBACK callback, void \*pdata);
* int GpioDisableTrigger(unsigned int port);
* int GpioEnablePWM(unsigned int port, unsigned int freq, unsigned int duty);
* int GpioDisablePWM(unsigned int port);

例子如下：

* GpioSetDirection(10, GPIO\_DIRECTION\_OUTPUT) //设置ck的PD1PORT10为输出
* GpioSetLevel(33, GPIO\_LEVEL\_HIGH) //设置ck的PD1PORT33为高电平

### 应用层GPIO使用方法

应用层可以通过/sys/class/gpio/中的sysfs节点来控制gpio。

echo pin number进/sys/class/gpio/export 生成gpioxx目录，其换算关系如下：

* ARM域的port echo进export的数字就是PD2PORTXX的数字
* MCU域的port echo进export的数字是PD1PORTXX的数字加上64

例子如下：

控制PD2PORT42脚

echo 42 > /sys/class/gpio/export

cd /sys/class/gpio/gpio42

cat direction //查看GPIO的输入输出属性

echo out > direction //设置管脚为输出

echo 1 > value //设置管脚为高电平

控制PD1PORT10脚

echo 74 > /sys/class/gpio/export

cd /sys/class/gpio/gpio74

cat direction //查看GPIO的输入输出属性

echo out > direction //设置管脚为输出

echo 1 > value //设置管脚为高电平

pwmchip0对应gpa0组，pwmchip32对应gpa1组，应用层控制pwm输出例子如下:

控制PD2PORT26脚

echo 26 > /sys/class/pwm/pwmchip0/export

cd /sys/class/pwm/pwmchip0/pwm26

echo 0 > enable //关闭此port的pwm

echo 100000 > period //周期设置为100 us

echo 50000 > duty\_cycle //高电平持续时间设置为50 us

echo 1 > enable

控制PD2PORT33脚

echo 1 > /sys/class/pwm/pwmchip32/export

cd /sys/class/pwm/pwmchip32/pwm1

echo 0 > enable //关闭此port的pwm

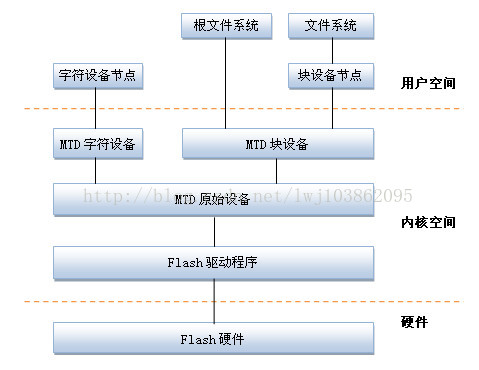
echo 100000 > period //周期设置为100 us

echo 50000 > duty\_cycle //高电平持续时间设置为50 us

echo 1 > enable

# Flash分区及文件系统管理

Kamino18在Linux内核中，引入MTD（Memory Technology Device）层为NOR FLASH和NAND FLASH设备提供统一接口。MTD将文件系统与底层FLASH存储器进行了隔离。



## 分区设置

K18 Flash的分区位置与大小在uboot中设置，如下：

$(K18DIR)/uboot/include/configs/leo\_k18\_universal.h

#define KERNEL\_SIZE\_NAND 0x500000

#define CONFIG\_KERNEL\_OFFS 0x500000

#define CONFIG\_KERNEL\_SIZE KERNEL\_SIZE\_NAND

#define CONFIG\_FDT\_ADDR 0x10000100

#define CONFIG\_FDT\_OFFS 0x280000

#define CONFIG\_FDT\_SIZE 0x80000

#define CONFIG\_OTA\_OFFS 0xA00000

#define CONFIG\_OTA\_SIZE 0x800000

#define CONFIG\_FDT\_IMAGE "leo\_k18\_universal.dtb"

#define CONFIG\_ROOTFS\_OFFS 0x1200000

#define CONFIG\_ROOTFS\_SIZE 0x3200000

#define CONFIG\_ROOTFS\_POS 9

#define CONFIG\_MEM\_SIZE 127

#define CONFIG\_PSTORE\_MEM\_ADDRESS 0x17f00000

#define CONFIG\_PSTORE\_MEM\_SIZE 0x100000

/\* \* mtd partitioning for serial NOR/NAND flash

\*

\* device <spinand\_mcu>, # parts = 10

\* #: name size offset mask\_flags

\* 0: mcu 0x00100000 0x00000000 0

\* 1: uboot-stage1 0x00040000 0x00100000 0

\* 2: uboot\_env 0x00040000 0x00140000 0

\* 3: uboot-stage2 0x00080000 0x00180000 0

\* 4: boot\_music 0x00080000 0x00200000 0

\* 5: kernel\_dtb 0x00080000 0x00280000 0

\* 6: misc 0x00200000 0x00300000 0

\* 7: kernel 0x00500000 0x00500000 0

\* 8: ota 0x00800000 0x00A00000 0

\* 9: rootfs 0x03C00000 0x01200000 0

\* 10: data 0x03200000 0x04E00000 0 \* \*/

#define MTDIDS\_NOR "spi32766.1"

#define MTDIDS\_NAND "spinand\_mcu"

#define MTDPARTS\_DEFAULT "mtdparts=${mtdids\_default}:" \

"1m(mcu)," \

"256k(uboot\_stage1)," \

"256k(uboot\_env)," \

"512k(uboot\_stage2)," \

"512k(boot\_music)," \

"${fdt\_size}@${fdt\_offs}(kernel\_dtb)," \

"2m(misc)," \

"${kernel\_size}@${kernel\_offs}(kernel)," \

"${ota\_size}@${ota\_offs}(ota\_kernel)," \

"${rootfs\_size}@${rootfs\_offs}(rootfs)," \

"60m(data)"

Note:

当修改各分区位置时，也需要修改$(K18DIR)/openwrt/target/linux/leo/k18-universal/download.sh中各分区的烧写位置，如下：

# partition start address

MCU\_OFFSET=0x0

SPL\_OFFSET=0x100000

UBOOT\_OFFSET=0x180000

BOOTMUSIC\_OFFSET=0x280000

KERNEL\_DTB\_OFFSET=0x300000

MISC\_OFFSET=0x380000

KERNEL\_OFFSET=0x480000

OTA\_OFFSET=0x980000

ROOTFS\_OFFSET=0x1180000

DATA\_OFFSET=0x3A00000

FLASH\_SIZE=0x8000000

## 文件系统配置

Rootfs文件系统设置成ubifs readonly，相应mount命令在uboot中设置，如下：

$(K18DIR)/uboot/include/configs/leo\_common.h

/\* Extra Environment \*/

#define CONFIG\_EXTRA\_ENV\_SETTINGS \

…

"mtdargs=ubi.mtd="\_\_stringify(CONFIG\_ROOTFS\_POS) ",2048 root=ubi0:rootfs rootwait " \

"rootfstype=ubifs ro loglevel=6 mem="\_\_stringify(CONFIG\_MEM\_SIZE)"M\0" \

…

为了调试系统方便，有时我们需要把rootfs分区变为可读写分区，有两种方法。

* 方法1，在uboot命令行中执行如下命令：

LEO#editenv mtdargs

LEO#ubi.mtd=9,2048 root=ubi0:rootfs rootwait rootfstype=ubifs rw loglevel=6

LEO#save

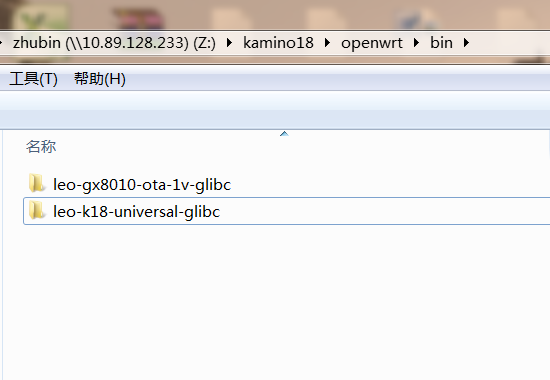
* 方法2，在adb shell命令行中执行如下命令：

mount -o remount rw /

# 镜像烧写

## 镜像位置

当系统镜像用build.sh完整编译完成后在$(K18DIR)/openwrt/bin/目录中会生成两个目录，如下图所示：



leo-k18-universal-glibc/full\_images目录中包含系统烧写所需全部镜像，以及Linux或Windows电脑上用的烧写工具。leo-gx8010-ota-1v-glibc目录中包含recovery镜像，最终recovery镜像也会被包含在leo-k18-universal-glibc/full\_images目录中。

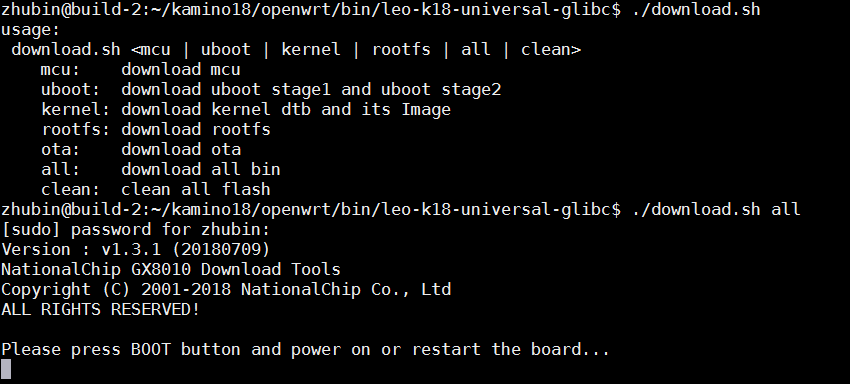
leo-k18-universal-glibc/full\_images目录中各文件的含义如下表：

|  |  |
| --- | --- |
| **文件名** | **文件用途** |
| bootmusic.wav | 启动音乐文件 |
| bootx | Linux电脑烧写工具可执行文件 |
| bootx.exe | Windows电脑烧写工具可执行文件 |
| download.bat | Windows电脑刷机脚本 |
| download.sh | Linux电脑刷机脚本 |
| mcu.bin | MCU分区镜像 |
| openwrt-leo-k18-universal.dtb | Kernel device tree blob分区镜像 |
| openwrt-leo-k18-universal-app.img | Data分区app镜像 |
| openwrt-leo-k18-universal-squashfs.img | 无用 |
| openwrt-leo-k18-universal-ubi.img | Rootfs分区ubi镜像 |
| openwrt-leo-k18-universal-u-boot.img | Uboot分区镜像 |
| openwrt-leo-k18-universal-u-boot-spl.bin | Spl分区镜像 |
| openwrt-leo-k18-universal-zImage | Kernel分区镜像 |
| openwrt-leo-ota-fit-uImage-initramfs.itb | Recovery分区镜像 |
| rokid-openwrt-leo-k18-universal-7.26.0-20181024-121845-ota.img | Ota全包升级镜像 |
| md5sum.txt | Ota全包升级镜像MD5值 |
| rokid-openwrt-leo-k18-universal-7.26.0-20181024-121845-upgrade.img | 不用 |

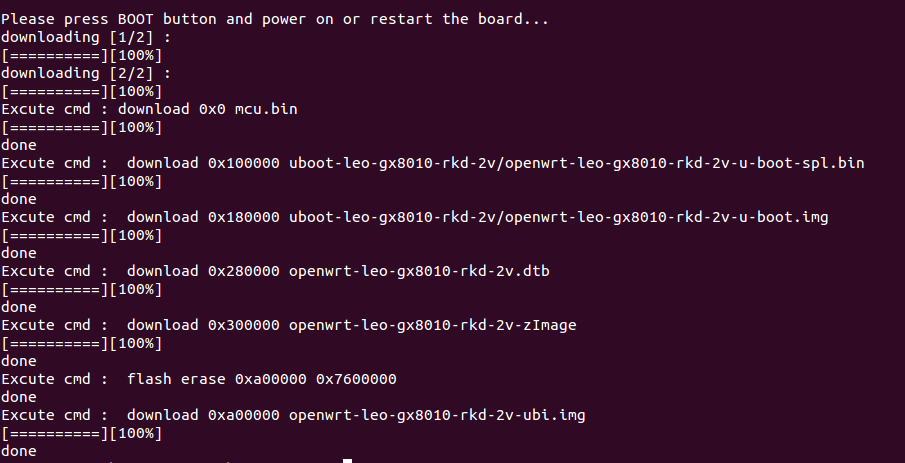
## Linux烧写工具使用

Linux电脑上具体烧写流程如下：

* cd $(K18DIR)/openwrt/bin/ leo-k18-universal-glibc /full\_images
* ./download.sh all



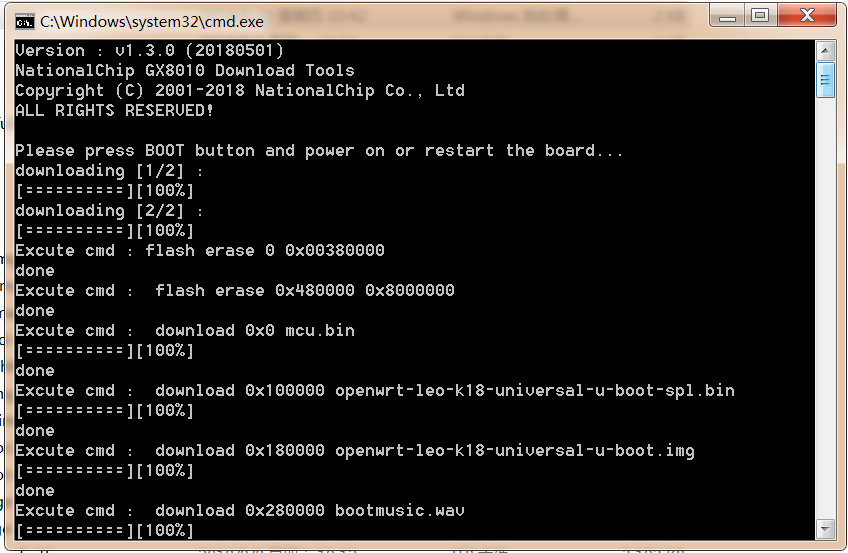
根据提示按住调试板上的BOOT\_KEY，然后连接USB线到PC上，下载完成如下：



## Windows烧写工具使用

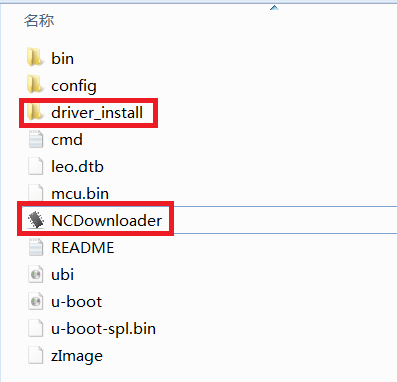
使用windows烧写工具前，对于一台新电脑需要安装usb驱动（驱动安装文件和说明在leo-k18-universal-glibc/bootx\_win/driver\_install目录下）,Windows电脑上具体烧写流程如下：

* 进入$(K18DIR)/openwrt/bin/ leo-k18-universal-glibc /full\_images目录
* 双击download.bat
* 根据提示按住调试板上的BOOT\_KEY，然后连接USB线到PC上.



## 工厂烧写工具使用

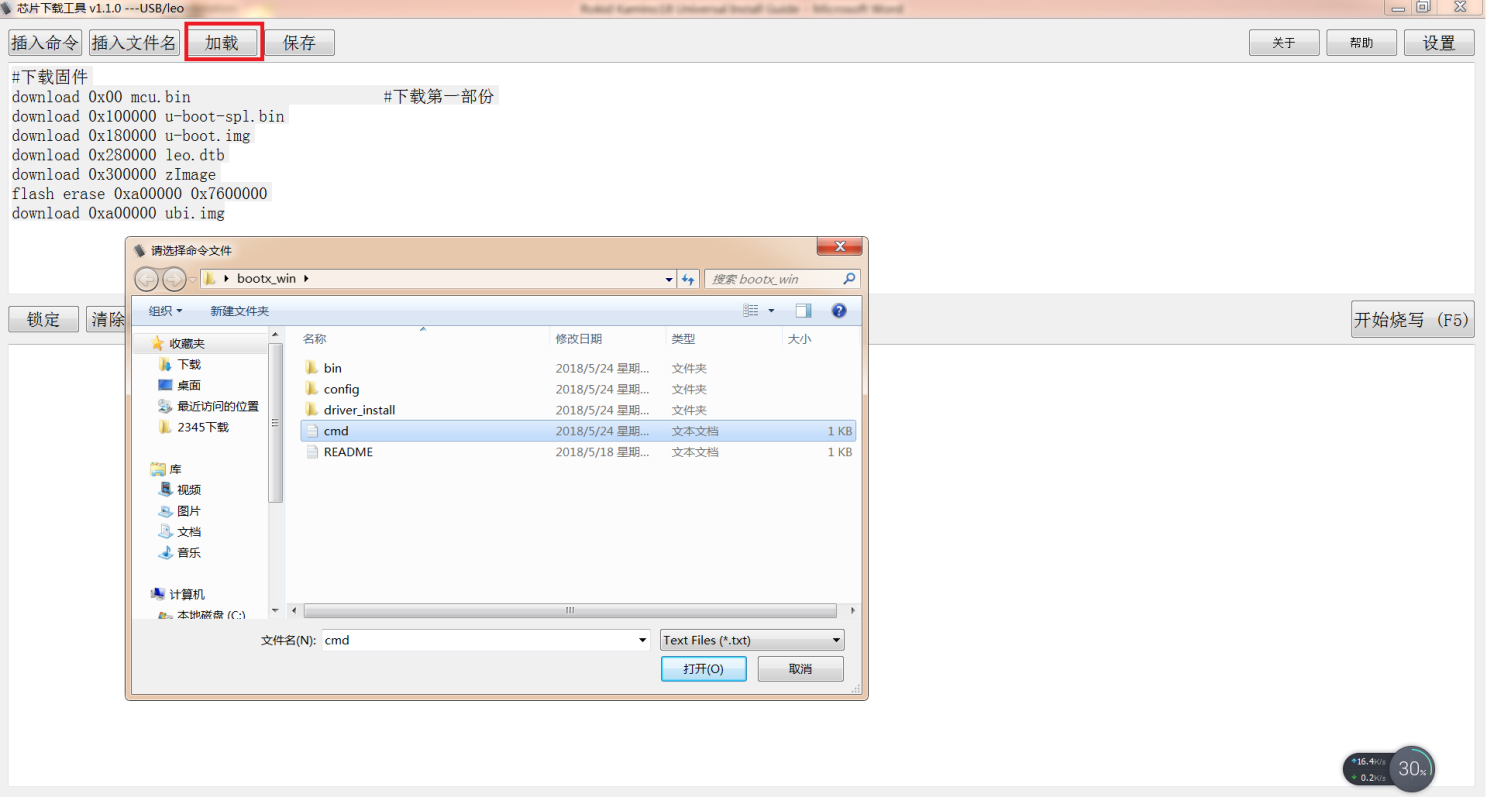
工厂烧写工具是一个带UI界面的windows烧写工具，对于一台新电脑需要安装usb驱动（驱动安装文件和说明在leo-k18-universal-glibc/bootx\_win/driver\_install目录下），同时把需要烧写的镜像复制到bootx\_win目录下，如下图所示：

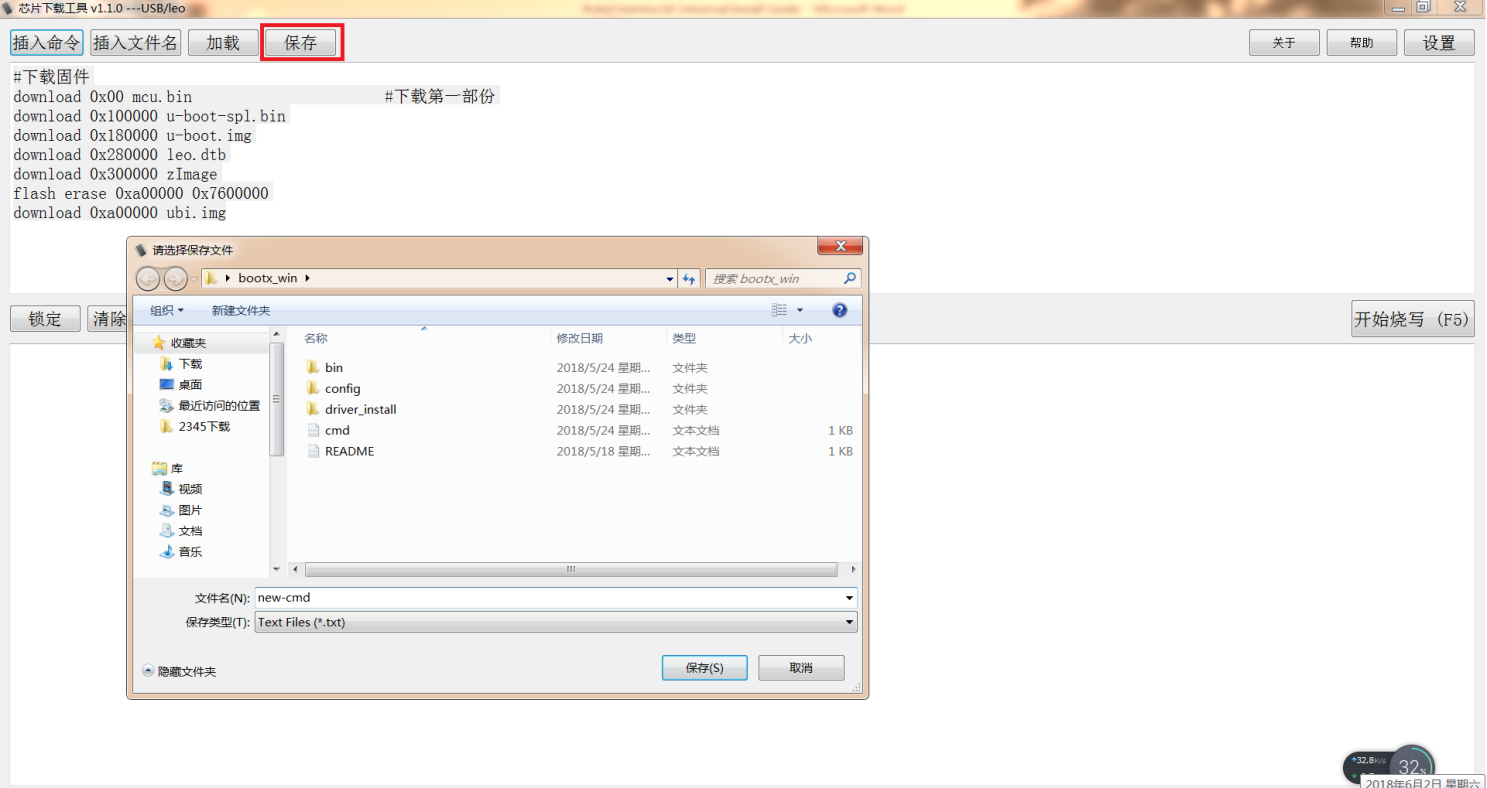


该工具最多支持同时烧写8台Kamino18的设备端，运行烧写工具NCDownloader,显示界面如下：



位于上方的输入框是烧写命令列表栏，下方的输出框是烧写log输出栏。我们可以自己输入烧写命令，也可以从命令文件载入烧写命令，另外可以把烧写命令列表栏中的命令保存到命令文件中。如下两图所示：





同Linux电脑烧写过程类似，按住调试板上的BOOT\_KEY，然后连接USB线到PC上，然后点击烧写工具的“开始烧写”按钮，完成烧写过程如下：



Note：

烧写命令列表栏中的download offset需要同uboot中设定的各分区的offset一致。

# SN与Seed烧写

为了能够正常配网并使用ROKID网络服务，需要正常烧写经ROKID授权的SN和Seed号，其步骤如下：

* factory 0 0   ---进入ftm模式，系统会重启，待系统重启后再执行下面步骤；
* factory 9 1 xxx   ---写入seed，xxx是具体seed number
* factory 9 3 xxx  ---写入sn，xxx是具体sn number
* factory 9 5 xxx  ---写入devicetypeid，xxx是具体devicetypeid number
* factory 9 7 xxx  ---写入factory\_date，xxx是具体factory\_date number
* factory 0 3 ---重启并进入正常模式；

上面步骤中，第2~5步，无所谓顺序，也可以只执行某几条。

# 配网验证

烧写完成后，为Kamino18 Dev3.0连接电源，用户会听到“系统启动中”的开机提示，稍后会听到配网提示。对开发者来说，需要下载Rokid手机APP进行配网，方式如下：

## BT配网

在手机上安装Rokid APP，通过蓝牙配置WIFI热点的SSID和Password到设备端后，系统配网成功可以正常进行语音交互。



**蓝牙连接**



**输入WIFI路由器SSID和密码**



**设备配网成功**

# 常见问题