Rockchip RK2118 RT-Thread 快速入门

文档标识：RK-JC-YF-589

发布版本：V1.0.1

日期：2024-03-15

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文主要描述了 RK2118 的基本使用方法，旨在帮助开发者快速了解并使用 RK2118 SDK 开发包。

各芯片系统支持状态

芯片名称	内核版本
RK2118	RT-Thread v4.1.x

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	版本	作者	修改说明
2024-02-26	V1.0.0	Roger Hu	初始版本
2024-03-15	V1.0.1	Roger Hu	更详细的说明==（请简单罗列新增了哪些东西）==

目录

开发环境搭建

开发环境选择

本 SDK 推荐的编译环境是64位的 Ubuntu20.04 或 Ubuntu18.04。

编译工具链选择

编译工具选用的是 RT-Thread 官方推荐的 SCons + GCC，SCons 是一套由 Python 语言编写的开源构建系统， GCC 交叉编译器由 ARM 官方提供，可直接使用以下命令安装所需的所有工具：

```
sudo add-apt-repository ppa:team-gcc-arm-embedded/ppa
sudo apt-get update
sudo apt-get install gcc-arm-embedded scons clang-format astyle libncurses5-dev
build-essential python-configparser
```

从ARM 官网下载编译器，通过环境变量指定 toolchain 的路径即可，具体如下：

```
wget https://developer.arm.com/-/media/Files/downloads/gnu/13.2.rel1/binrel/arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi.tar.xz?rev=e434b9ea4afc4ed7998329566b764309&hash=688C370BF08399033CA9DE3C1CC8CF8E31D8C441
tar -xvf arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi.tar.xz

export RTT_EXEC_PATH=/path/to/toolchain/arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi/bin
```

此外，我们也会在SDK初始包中附上编译器：arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi.tar.xz

目录结构

以下是SDK主要目录对应的说明：

```
├── applications            # Rockchip应用demo源码
├── AUTHORS
├── bsp                     # 所有芯片相关代码
│   ├── rockchip
│   │   ├── common
│   │   │   ├── drivers     # Rockchip OS适配层通用驱动
│   │   │   ├── hal         # Rockchip HAL(硬件抽象层)实现
│   │   │   └── tests       # Rockchip 驱动测试代码
│   │   ├── rk2118          # RK2118 主目录
│   │   │   ├── board       # 板级配置
│   │   │   ├── build       # 编译主目录，存放中间文件
│   │   │   ├── drivers     # RK2118 私有驱动目录
│   │   │   ├── Image       # 存放固件
│   │   └── tools           # Rockchip 通用工具
├── ChangeLog.md
├── components              # 系统各个组件，包括文件系统，shell和框架层等驱动
│   ├── hifi4
│   │   ├── dsp             # dsp代码
│   │   ├── rtt             # 运行在mcu上dsp相关代码
│   │   ├── shared          # mcu/dsp 公共代码
│   │   └── tools           # dsp固件生成工具
├── documentation           # RT-Thread官方文档
├── examples                # RT-Thread例子程序和测试代码
├── include                 # RT-Thread官方头文件目录
├── Kconfig
├── libcpu
├── LICENSE
├── README.md
├── README_zh.md
├── RKDocs                 # Rockchip 文档
├── src                    # RT-Thread内核源码
├── third_party            # Rockchip增加的第三方代码的目录
└── tools                  # RT-Thread官方工具目录，包括menuconfig和编译脚本
```

工程编译配置

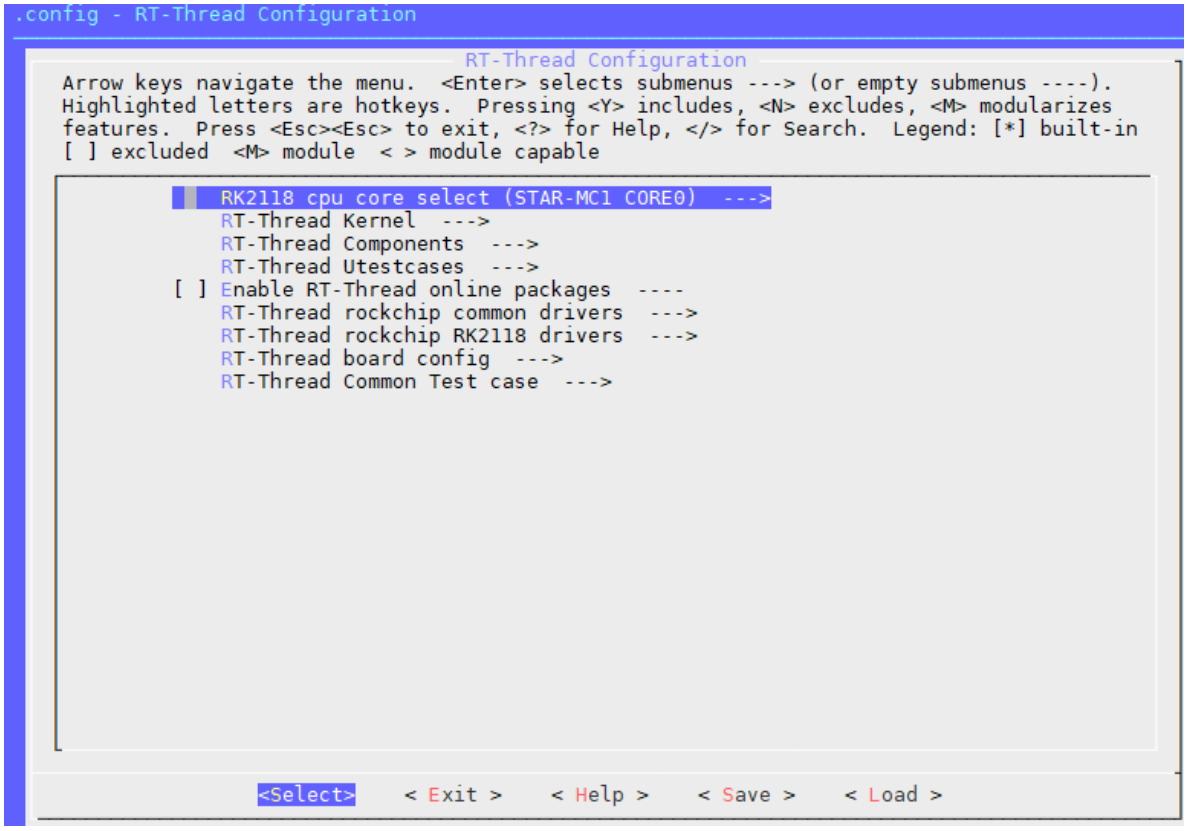
RT-Thread 用 SCons 来实现编译控制，SCons 是一套由 Python 语言编写的开源构建系统，类似于 GNU Make。它采用不同于通常 Makefile 文件的方式，而使用 SConstruct 和 SConscript 文件来替代。这些文件也是 Python 脚本，能够使用标准的 Python 语法来编写，所以在 SConstruct、SConscript 文件中可以调用 Python 标准库进行各类复杂的处理。

工程配置

进入到对应的工程目录： bsp/rockchip/rk2118 ，选择一个board开发板配置如adsp_demo，并运行工程配置工具 menuconfig：

```
cd bsp/rockchip/rk2118
cp board/adsp_demo/defconfig .config   #(位于各个board目录下的defconfig文件是默认配置)
scons --menuconfig
```

会弹出如下界面，这个过程会从 .config 载入当前的默认配置，退出保存配置时会覆盖 .config，同时自动生成一个 rtconfig.h 文件，这2个文件包含了我们选中的各种配置，最终参与编译的只有这个 rtconfig.h。



menuconfig 工具的常见操作如下：

- 上下箭头：移动
- 回车：进入子菜单
- ESC 键：返回上级菜单或退出
- 英文问号：调出帮助菜单（退出帮助菜单，请按回车键）。
- 空格、Y 键或 N 键：使能/禁用 [*] 配置选项
- / 键：寻找配置项目

  每个板级目录下都有一个默认的配置文件 defconfig，里面包含了这个板子的常规配置，可以基于这个配置去修改。

保存配置

在每一个板级配置目录下都有一个默认配置 defconfig，如果没有执行 scons menuconfig，会用默认的 rtconfig.h 参与编译。要修改板子的 defconfig，可以先用它的 defconfig 文件覆盖 .config，通过 menuconfig 修改后再使用新的 .config 更新 defconfig 文件，下面是具体例子：

```
cp board/xxx/defconfig .config              ; 拷贝要修改的板子的默认配置
scons --menuconfig                          ; 修改配置项并保存
cp .config board/xxx/defconfig              ; 保存配置为板子的默认配置
```

工程编译

编译命令

编译命令如下：

```
cd bsp/rockchip/rk2118
scons -j8
```

以上命令将使用当前目录的 rtconfig.h 作为编译配置，最后会在当前目录下生成如下文件：

```
ls -l rtthread*
-rwxrwxr-x 1 cmc cmc 599616 Feb 15 19:45 rtthread.elf    #elf可执行文件，可用于jtag
调试
-rw-rw-r-- 1 cmc cmc 489470 Feb 15 19:45 rtthread.map    #符号表
-rwxrwxr-x 1 cmc cmc  56760 Feb 15 19:45 rtthread.bin    #RT-Thread系统固件
```

打包固件命令如下，需要指定一个合适固件打包配置文件，如board/adsp_demo/setting.ini，否则将使用默认的配置文件board/common/setting.ini：

```
./mkimage.sh board/adsp_demo/setting.ini
```

将会在Image目录下生成完整固件文件：

```
ls -l Image/Firmware.img
-rw-r--r-- 1 rk rk 1638400 Feb 25 07:40 Image/Firmware.img #这就是完整固件
```

其中 Firmware.img 是我们下载到机器上的二进制固件，它由 Loader（rk2118_loader.bin）、 Trust Firmware (tfm_s.bin)、RT-Thread 系统固件（rtthread.bin）、DSP固件 (dsp0.bin, dsp1.bin, dsp2.bin) 和根文件系统root.img打包而成。

```
ll bsp/rockchip/rk2118/rtthread.bin
-rwxrwxr-x 1 hwg hwg 157696 Mar 15 14:36 bsp/rockchip/rk2118/rtthread.bin //
rtthread固件
ll bsp/rockchip/rk2118/Image/
-rw-rw-r--  1 hwg hwg 3735552 Mar 15 14:36 Firmware.img // 打包后的整个固件
-rw-rw-r--  1 hwg hwg  158212 Mar 15 14:36 rtthread.img // 打包rtthread.bin成img
ll bsp/rockchip/rk2118/rkbin/
-rw-rw-r--  1 hwg hwg 39303 Mar 12 10:44 rk2118_db_loader.bin // 下载固件使用的
loader
-rw-rw-r--  1 hwg hwg 10382 Mar  5 17:11 rk2118_ddr.bin // loader中负责ddr的探测和
初始化
```

```
-rw-rw-r--  1 hwg hwg 12592 Mar  5 17:11 rk2118_loader.bin // loader固件
-rw-rw-r--  1 hwg hwg 17152 Mar  5 17:11 rk2118_upgrade.bin // loader中usb plug功
能
-rw-rw-r--  1 hwg hwg 10752 Mar 12 10:44 tfm_s.bin // Trust Firmware
ll components/hifi4/rtt/dsp_fw/
-rw-rw-r-- 1 hwg hwg 14336 Mar  5 09:10 dsp0.bin // dsp0固件
-rw-rw-r-- 1 hwg hwg 14336 Mar  5 09:10 dsp1.bin // dsp1固件
-rw-rw-r-- 1 hwg hwg 14336 Mar  5 09:10 dsp2.bin // dsp2固件
```

SCons 构建系统默认是通过 MD5 来判断文件是否需要重新编译，如果代码文件内容没变，而只是时间戳变了（例如通过 touch 更新时间戳），是不会重新编译这个文件及其依赖的。另外，如果仅修改无关内容，例如代码注释，则只会编译，而不会链接，因为 obj 文件内容未发生变化。因此，在开发过程中如果碰到各种修改后实际并未生效的问题，建议在编译前做一次清理，命令如下：

```
scons -c
```

如果做完上面的清理以后，还有异常，可以强制删除所有中间文件，命令如下：

```
rm -rf build
```

其他 SCons 命令，可以看帮助或文档：

```
scons -h
```

固件打包

固件打包是为了把系统需要的各种固件打包在一起，如分区表、loader、rtthread OS、DSP固件和根文件系统，RK2118 的固件打包脚本是：bsp/rockchip/rk2118/mkimage.sh 。具体参考上面的./mkimage.sh 命令。

固件烧录

注意

- 由于打印的uart和烧写的uart是同一个，所以需要确保烧写的时候，要把串口打印的客户端先断开
- uart烧写的时候不能插usb

Windows版升级工具

工具位于：bsp/rockchip/tools/SocToolKit_v1.9_20240130_01_win.zip

烧写前先确保设备已经切到maskrom模式，目前Windows也支持uart和usb两种烧写方式，所以第一步先选择烧写方式，选择uart的话要指定串口号和波特率，目前最高支持1.5M波特率烧写。同时支持全固件烧写和分区烧写两种方式，其中全固件烧写方式如下：

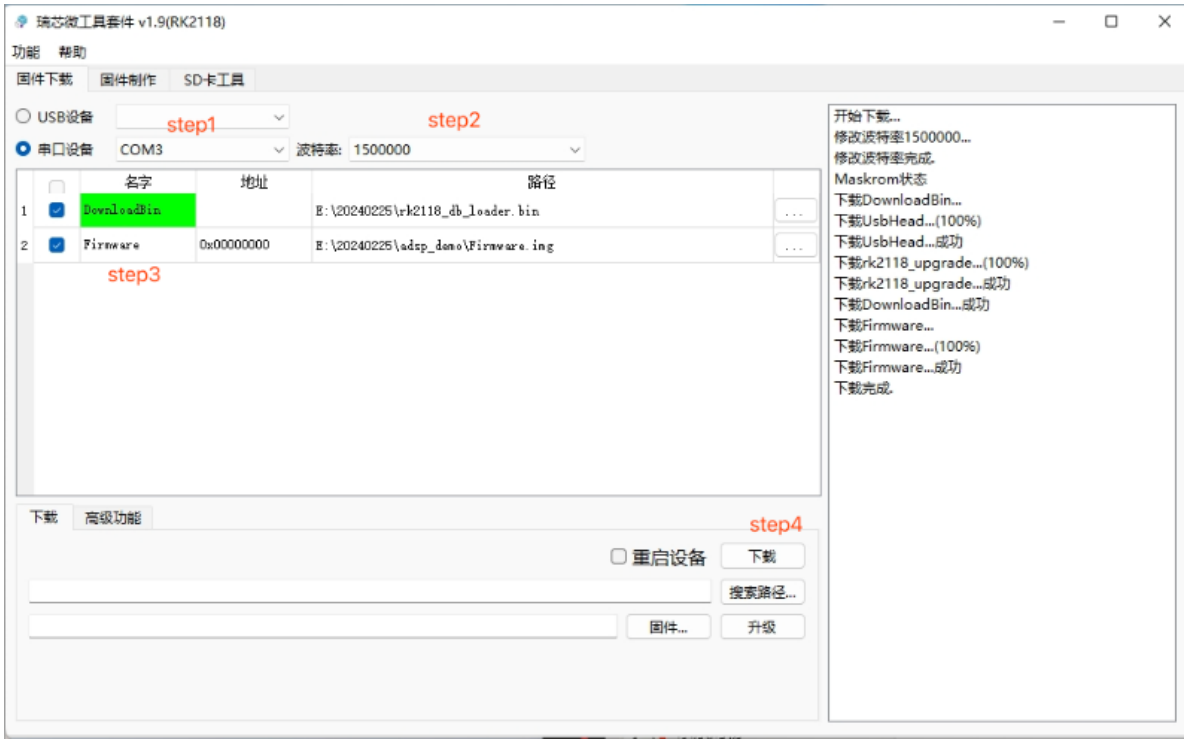
step1：选择正确的串口

step2：才能波特率1500000

step3：选择loader：bsp/rockchip/rk2118/rkbin/rk2118_db_loader.bin

选择Firmware.img：bsp/rockchip/rk2118/Image/Firmware.img，地址为0x0

step4：开始下载

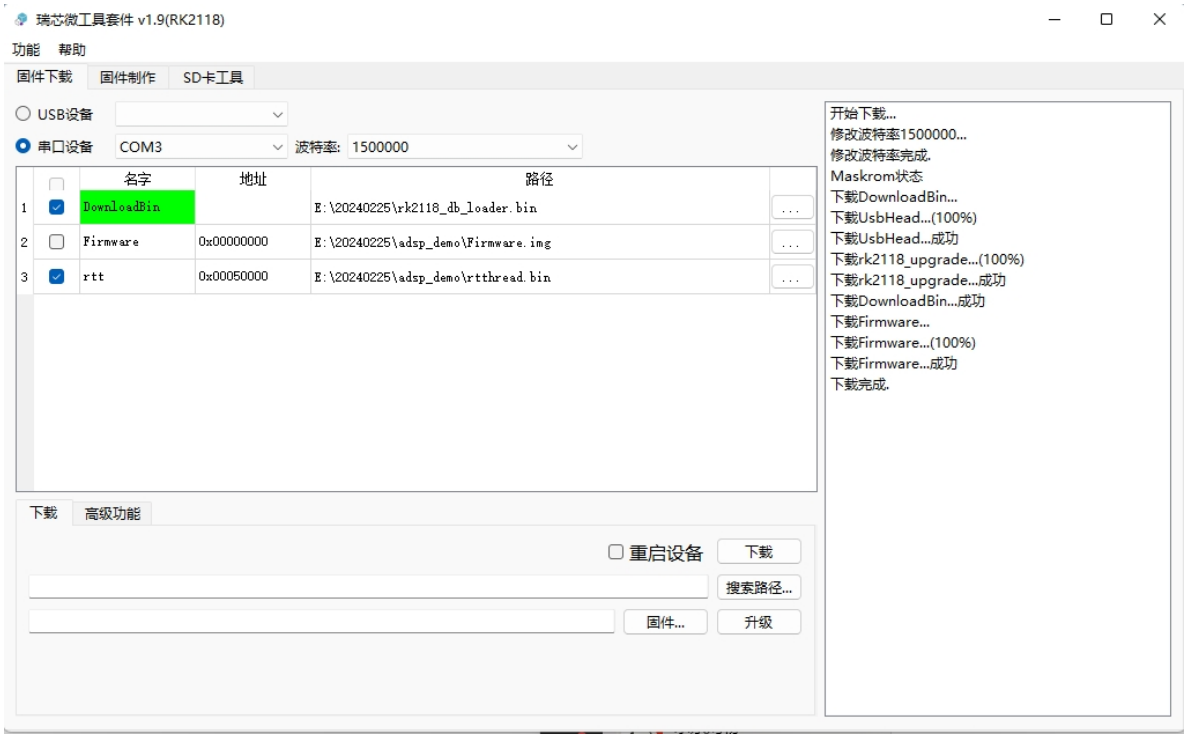


如果想烧写指定分区，则需要指定分区的地址，烧写方式如下：

选择loader：bsp/rockchip/rk2118/rkbin/rk2118_db_loader.bin

选择要升级的分区，如rtthread：bsp/rockchip/rk2118/rtthread.bin，地址从对应的setting.ini中查找，例如：

```
[UserPart3]
Name=OSA
Type=0x200
PartOffset=0x280  // rtthread.bin所在的位置，
                  // sector为单位，需要*512转换为字节，如0x280 X 512 = 0x5000
PartSize=0xa00
Flag=
File=../../rtthread.bin
```



Linux版烧录工具及命令

UART烧写

linux下默认ttyUSB设备是没有写权限的，所以烧之前可以通过sudo chmod 666 /dev/ttyUSBx（x是你的串口号）修改权限。

如果不想每次都去改串口权限，可以通过如下方式添加默认写权限：

```
sudo vi /etc/udev/rules.d/70-rk.rules
# 在创建中添加如下内容，保持退出即可
KERNEL=="ttyUSB[0-9]*", MODE="0666"
```

烧写方式如下：

```
sudo chmod 666 /dev/ttyUSB0
# 先切到maskrom下烧db loader
../tools/upgrade_tool/upgrade_tool db /dev/ttyUSB0 ./Image/rk2118_db_loader.bin
# 完整固件烧写
../tools/upgrade_tool/upgrade_tool wl /dev/ttyUSB0 0 ./Image/Firmware.img
# 在完整固件烧完之后，开发过程可以只烧录自己有更新的固件，例如下面命令就是单独烧cpu0的rtt固件，第三个参
数是固件位置，算法从上面的setting.ini里UserPart3分区，其PartOffset=0x280，所以这里就填0x280
../tools/upgrade_tool/upgrade_tool wl /dev/ttyUSB0 0x280 ./rtthread.bin
```

USB烧写

烧写方式如下：

```
# 先切到maskrom下烧db loader
../tools/upgrade_tool/upgrade_tool db ./Image/rk2118_db_loader.bin
# 完整固件烧写
../tools/upgrade_tool/upgrade_tool wl 0 ./Image/Firmware.img
# 在完整固件烧完之后，开发过程可以只烧录自己有更新的固件，例如下面命令就是单独烧cpu0的rtt固件，其中位置
信息需要保持和setting.ini中该分区的其PartOffset相等即可
../tools/upgrade_tool/upgrade_tool wl 0x280 ./rtthread.bin
```

运行调试

系统启动

系统启动方式有以下几种：

1. 固件升级后，自动重新启动；
2. 插入电源供电直接启动；
3. 有电池供电的设备，按Reset键启动；

系统调试

RK2118 支持串口调试。不同的硬件设备，其串口配置也会有所不同。

串口通信配置信息如下：

波特率：1500000

数据位：8

停止位：1

奇偶校验：none

流控：none

成功进入调试的截图：

```
Booting TF-M 20240303
                    clk_init: PLL_GPLL = 800000000
clk_init: PLL_VPLL0 = 1179648000
clk_init: PLL_VPLL1 = 903168000
clk_init: PLL_GPLL_DIV = 800000000
clk_init: PLL_VPLL0_DIV = 294912000
clk_init: PLL_VPLL1_DIV = 150528000
clk_init: CLK_DSP0_SRC = 400000000
clk_init: CLK_DSP0 = 700000000
clk_init: CLK_DSP1 = 200000000
clk_init: CLK_DSP2 = 200000000
clk_init: ACLK_NPU = 400000000
clk_init: HCLK_NPU = 133333333
clk_init: CLK_STARSE0 = 400000000
clk_init: CLK_STARSE1 = 400000000
clk_init: ACLK_BUS = 266666666
clk_init: HCLK_BUS = 133333333
clk_init: PCLK_BUS = 133333333
clk_init: ACLK_HSPERI = 133333333
clk_init: ACLK_PERIB = 133333333
clk_init: HCLK_PERIB = 133333333
clk_init: CLK_INT_VOICE0 = 49152000
clk_init: CLK_INT_VOICE1 = 45158400
clk_init: CLK_INT_VOICE2 = 98304000
clk_init: CLK_FRAC_UART0 = 64000000
clk_init: CLK_FRAC_UART1 = 48000000
clk_init: CLK_FRAC_VOICE0 = 24576000
clk_init: CLK_FRAC_VOICE1 = 22579200
clk_init: CLK_FRAC_COMMON0 = 12288000
clk_init: CLK_FRAC_COMMON1 = 11289600
clk_init: CLK_FRAC_COMMON2 = 8192000
clk_init: PCLK_PMU = 100000000
clk_init: CLK_32K_FRAC = 32768
clk_init: CLK_MAC_OUT = 50000000
clk_init: MCLK_PDM = 100000000
clk_init: CLKOUT_PDM = 3072000
clk_init: MCLK_SPDIFTX = 6144000
clk_init: MCLK_OUT_SAI0 = 12288000
clk_init: MCLK_OUT_SAI1 = 12288000
clk_init: MCLK_OUT_SAI2 = 12288000
clk_init: MCLK_OUT_SAI3 = 12288000
clk_init: MCLK_OUT_SAI4 = 12288000
clk_init: MCLK_OUT_SAI5 = 12288000
clk_init: MCLK_OUT_SAI6 = 12288000
clk_init: MCLK_OUT_SAI7 = 12288000
clk_init: SCLK_SAI0 = 12288000
clk_init: SCLK_SAI1 = 12288000
clk_init: SCLK_SAI2 = 12288000
clk_init: SCLK_SAI3 = 12288000
clk_init: SCLK_SAI4 = 12288000
clk_init: SCLK_SAI5 = 12288000
clk_init: SCLK_SAI6 = 12288000
clk_init: SCLK_SAI7 = 122880?

 \ | /
- RT -     Thread Operating System
 / | \     4.1.1 build Mar 15 2024 17:10:53
 2006 - 2022 Copyright by RT-Thread team
[I/I2C] I2C bus [i2c2] registered
[DSP0 INFO] Hello RK2118 dsp0, build Mar 15 2024 16:58:49
[DSP1 INFO] Hello RK2118 dsp1, build Mar 15 2024 16:59:09
[DSP2 INFO] Hello RK2118 dsp2, build Mar 15 2024 16:59:30
delay 1s test start
msh >delay 1s test end
```