国家电网
STATE GRID

北京智芯微电子科技有限公司
BEIJING SMARTCHIP MICROELECTRONICS TECHNOLOGY CO., LTD.

智芯
SMART CHIP

SCA200x 开发环境用户指南

文档版本　01

发布日期　2022-03-01

商标声明

是北京智芯微电子科技有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受智芯公司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的购买或使用范围之内。除非合同另有约定，本公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

北京智芯微电子科技有限公司

地址：北京市昌平区中科云谷园智芯园区　　邮编：102200

修订历史

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

版本	修订日期	修订描述
V1.0	2022-03-01	第一版

目录

1、 概述

本文描述了使用 SDK 搭建用户开发环境的方法。包括使用 BurningTool 进行 EVB 板烧写，uboot 的编译移植，kernel 的编译移植，文件系统的制作、烧写等。

1.1 注意 1

系统 kernel、uboot、busybox 基础环境等使用 64bit 工具链编译。
SDK 开发包中提供了 lib64 和 lib32 两套基础运行库，用户应用程序默认是 32bit 环境，使用 32bit 工具链编译程序可以直接在 EVB 板上进行测试。

编译 64bit 应用程序在 EVB 板上运行时需要在编译中使用如下选项来主动指定到 lib64：

```
LDFLAGS += -Wl,-rpath,/lib64:/usr/lib64。
```

1.2 注意 2

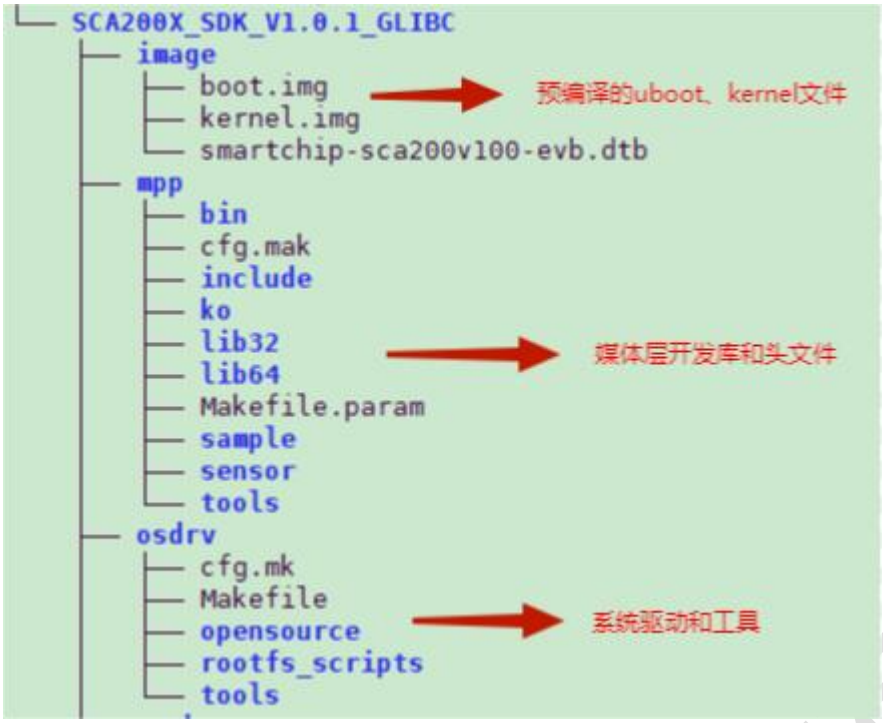
安装好工具链后，可以直接在 SDK 发布包的 osdrv 目录下执行 make 操作，自动进行 uboot、kernel、busybox、文件系统的编译。

本文档相对应的 SDK 产品版本为 SCA200VX_SDK_Vx.0.x_GLIBC

2、 SDK 目录说明

将 SDK 压缩包 SCA200x_SDK_V1.0.1_GLIBC.tgz 解压后，目录主要内容如下图所示：

2.1 sdk 根目录



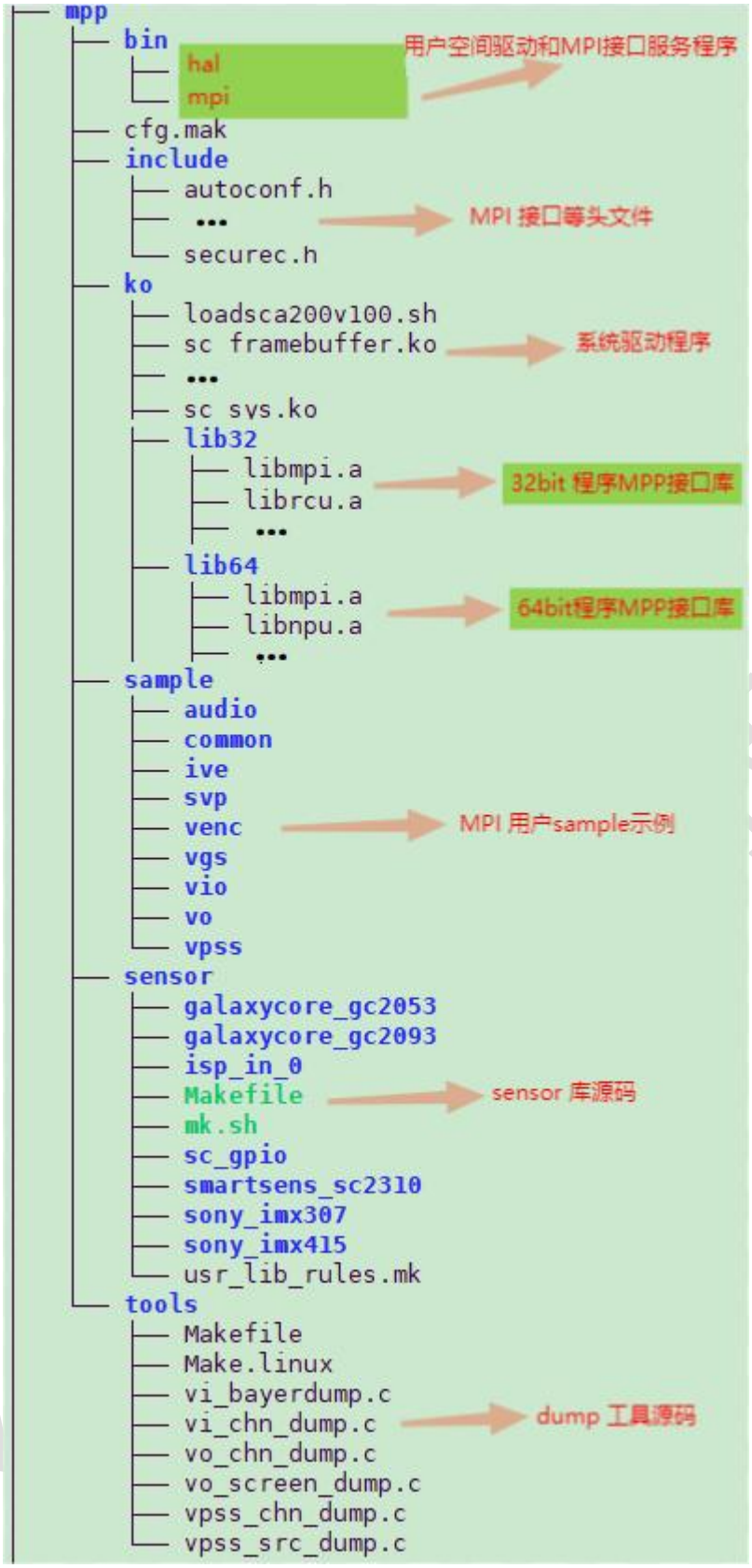
2.2 osdrv 子目录

编译整个 sdk:

- cd osdrv; make

2.3 mpp 子目录

mpp 目录中包含了用户需要添加到文件系统中的用户空间驱动程序(bin 目录下)、kernel 驱动和脚本(ko 目录)、基础 Lib 库(lib64 或者 lib32，根据用户的应用程序决定)等，在 ko 目录中有 loadsca200v100.sh 脚本，会自动运行 bin 下面的 hal 和 mpi，并加载 KO 文件等，用户可以根据自己文件系统组织目录适当修改该脚本。



3、 开发环境搭建

3.1 linux 开发服务器安装

推荐使用 ubuntu 20.04 x86_64. 安装如下软件包:

- sudo apt-get install gcc make
- sudo apt-get install bison flex u-boot-tools zlib1g

建议安装如下包:

- apt-get install ssh openssh-server samba smbclient nfs-kernel-server nfs-common portmap

3.2 交叉编译工具链安装

随 SDK 发布包一起含有两套编译工具链，分别是 32bit 和 64bit，其中 kernel、uboot、busybox 必须用 64bit 来编译，用户应用程序可以选择使用 32bit 或者 64bit 工具链来编译。

- 64bit 工具链：gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
- 32bit 工具链：gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf.tar.xz

安装时可以使用目录中的 install.sh 脚本安装：
命令如下：

- sudo bash install.sh

3.3 文件系统移植

3.3.1 文件准备

EVB 板目前采用的是 EMMC，使用 ext4 文件系统。用户可以根据自身需求制作对应存储器件的文件系统。在 SDK 发布包的 osdrv\rootfs_scripts 目录下有基础文件系统，已经将/lib32，/lib64 基础环境准备好。用户自制文件系统时涉及到以下内容需要拷贝：

- 拷贝 ko 驱动：SCA200V100_SDK_V1.0.1_GLIBC/mpp/ko/*.ko ==> rootfs/komod/
- 拷贝 load 脚本：SCA200V100_SDK_V1.0.1_GLIBC/mpp/ko/loadsca200v100.sh ==> rootfs/komod/
- 拷贝用户空间驱动：SCA200V100_SDK_V1.0.1_GLIBC/mpp/bin/hal ==> rootfs/usr/bin/hal
- 拷贝 MPI 接口服务程序 SCA200V100_SDK_V1.0.1_GLIBC/mpp/bin/mpi ==> rootfs/usr/bin/mpi
- tunning 文件 SCA200V100_SDK_V1.0.1_GLIBC/mpp/sensor/tuning/*.bin rootfs/local/

3.3.2 文件系统制作

工具：make_ext4fs

工具在 SDK 发布包的位置：osdrv/tools/pc/ext4_utils/

使用工具可以制作 3 种类型的镜像，分别是：
raw image：原始格式文件，主要用于烧写器烧写。
gzip image：将 raw image 使用 gzip 压缩后的格式，用户可以用来做升级包(大小较小，利于网络升级，但是需要接收端支持解压后烧写)。
sparse image：稀疏文件格式(可自行参考)，大小介于 raw 和 gzip 之间，mmc 写入时可以使用 mmc swrite 命令进行烧写。

- ./make_ext4fs -l 16M test.raw.img test/ //16MB
- ./make_ext4fs -l 16M -z test.gzip.img test/ //17KB
- ./make_ext4fs -l 16M -s test.sparse.img test/ //4.4MB

工具用法：
make_ext4fs [-l <len>] [-z | -s] <filename> [<directory>]

-z: gzip
-s: sparse

4、 编译

4.1 uboot 配置和编译

生成目标 boot.img

4.1.1 默认配置

- make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- sca200v100_bd1_rel_emmc_defconfig

注：如果是 EVB2 使用如下：
make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- sca200v100_bd2_rel_emmc_defconfig

注意：EVB1 和 EVB2 是两种不同的开发版，具体请参考硬件手册。

4.1.2 修改配置

- make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- menuconfig

4.1.3 编译 boot.img

- make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu-

4.1.4 mmc 分区

使用 环境变量 bootargs, 传参 blkdevparts 给 kernel, 进行 mmc 分区。

配置文件 configs/sca200v100_bd2_rel_emmc_defconfig 中的配置项 CONFIG_BOOTARGS 生成 uboot 环境变量 bootargs.

示例:

```
grep CONFIG_BOOTARGS configs/sca200v100_bd2_dev_emmc_defconfig
CONFIG_BOOTARGS="console=ttyS0,115200,earlyprintk loglevel=8,quiet root=/dev/mmcblk0p5 rootwait ro rootfstype=ext4
blkdevparts=mmcblk0:512K(reserve),512K(uboot_env),8M(kernel),512K(kernel_dtb),512M(rootfs),-(other)"
```

4.1.5 内存分配

使用 环境变量 bootargs, 传参 mem/TOTAL_MEM/OS_MEM 给 kernel, 进行 mmz 和 os mem 分配。

示例:

```
setenv bootargs console=ttyS0,115200,earlyprintk loglevel=8,quiet root=/dev/mmcblk0p5 rootwait ro rootfstype=ext4
blkdevparts=mmcblk0:512K(reserve),512K(uboot_env),8M(kernel),512K(kernel_dtb),512M(rootfs),-(other) mem=512m TOTAL_MEM=1024 OS_MEM=512
```

注意:TOTAL_MEM 和 OS_MEM 用于 linux ko 计算 mmz 的大小,可以参考 loadsca200v100.sh 脚本设置 sc_osal.ko 部分。该部分功能由 uboot 代码中 common/autoboot.c 文件的 regenerate_bootargs() 函数实现。

4.2 linux 内核编译

生成目标: kernel.img 和 smartchip-sca200v100-evb.dtb

4.2.1 默认配置

- make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- sca200v100_emmc_defconfig

4.2.2 编译 kenel.img

-j16 多线程编译, 可根据用户服务器核心数自己调整

- make -j16 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- kernel-img

4.2.3 编译 dtb 文件

- make -j16 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- sc/smartchip-sca200v100-evb.dtb

注: 如果为 evb2 使用如下命令:

```
make -j16 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- sc/smartchip-sca200v100-bd2.dtb
```

注意: sca200v100_emmc_defconfig 配置中使能了 CONFIG_COMPAT 以便兼容客户 32bit 应用程序。sdk 提供的 kmodule 也都默认支持了 compat_ioctl。在 32 位应用程序下,客户自行开发的驱动也需要支持 compat_ioctl。

4.3 busybox 编译

4.3.1 编译

- make CROSS_COMPILE=aarch64-linux-gnu- smartchip_defconfig
- make -j16 CROSS_COMPILE=aarch64-linux-gnu- EXTRA_LDFLAGS="-Wl,-rpath,/lib64:/usr/lib64" busybox

注意：busybox 需要增加 EXTRA_LDFLAGS 设置 rpath 指向 64bit 环境。

4.3.2 安装

将 busybox 安装到 rootfs 目录下。

- make CROSS_COMPILE=aarch64-linux-gnu- EXTRA_LDFLAGS="-Wl,-rpath,/lib64:/usr/lib64" CONFIG_PREFIX=/xxx/osdrv/rootfs_scripts/rootfs install

4.4 用户程序和 sample 编译

SDK 发布包中提供的 rootfs (osdrv/rootfs_scripts/rootfs.tgz)包含了初始化脚本和程序运行时 lib 库。包括 lib32 和 lib64 环境，默认为 lib32 环境。/lib 和/usr/lib 是 /lib32 和/usr/lib32 的符号链接。

注意：64 位的应用程序需要在编译选项增加 rpath。设置 LDFLAGS += -Wl,-rpath,/lib64:/usr/lib64。

编译 64 位示例程序 Sample：

- cd mpp/sample; APP_32BIT=n make; cd -

编译 32 位示例程序 Sample：

- cd mpp/sample; APP_32BIT=y make; cd -

5、 烧写

5.1 裸片烧写

裸片烧写请使用 SDK 发布包里面的 BurningTool 工具，目前仅支持烧写 boot.img。具体请参见 BurnTool 烧写使用指南。

5.2 uboot 下烧写 emmc boot0 区

在 uboot 下执行命令烧写 boot.img

5.2.1 tftp 方式

- setenv ipaddr 192.168.137.126
- setenv serveip 192.168.137.5

将 boot.img 放入 28000000 内存中，需要用户开启 tftp server

- tftpboot 28000000 boot.img

将 boot.img 写入 MMC BOOT0 分区

- mmc dev 0 1; mmc write 28000000 0 0x640

5.2.2 usb 存储

- usb start; usb reset
- fatload usb 0 28000000 boot.img
- mmc dev 0 1; mmc write 28000000 0 0x640

5.2.3 sd 卡存储

- fatload mmc 1 28000000 boot.img
- mmc dev 0 1; mmc write 28000000 0 0x640

5.3 uboot 下烧写 emmc user 区

在 uboot 下执行命令烧写 kernel、rootfs、dtb 等
mmc user 区，分区表如下：

start(blk)	size(blk)(byte)	说明
0x00080000(0x000400)	0x00080000(0x000400)(512K)	uboot env
0x00100000(0x000800)	0x00800000(0x004000)(8M)	kernel
0x00900000(0x004800)	0x00080000(0x000400)(512K)	kernel dtb
0x00980000(0x004c00)	0x04000000(0x020000)(64M)	rootfs
0x04980000(0x024c00)		other

5.3.1 tftp 方式

- setenv ipaddr 192.168.137.126
- setenv serverip 192.168.137.5

将 img 放入 0x28000000 内存中， 并写入 kernel 分区中

- mw.b 28000000 ff 800000; tftpboot 28000000 kernel.img

- mmc dev 0 0; mmc write 28000000 800 4000

将 dtb 放入 0x28000000 内存中，并写入 kernel dtb 分区中

- mw.b 28000000 ff 80000; tftpboot 28000000 smartchip-sca200v100-evb.dtb
- mmc dev 0 0; mmc write 28000000 4800 400

rootfs.img 是 ext4 sparse image 使用如下命令:

- mw.b 28000000 ff 4000000; tftpboot 28000000 rootfs.ext4.sp.img
- mmc dev 0 0; mmc swrite 28000000 4c00

5.3.2 usb 盘

- usb start; usb reset

将 img 放入 0x28000000 内存中， 并写入 kernel 分区中

- mw.b 28000000 ff 800000; fatload usb 0 28000000 kernel.img
- mmc dev 0 0; mmc write 28000000 800 4000

将 dtb 放入 0x28000000 内存中，并写入 kernel dtb 分区中

- mw.b 28000000 ff 80000; fatload usb 0 28000000 smartchip-sca200v100-evb.dtb
- mmc dev 0 0; mmc write 28000000 4800 400

rootfs.img 是 ext4 sparse image 使用如下命令:

- mw.b 28000000 ff 4000000; fatload usb 0 28000000 rootfs.ext4.sp.img
- mmc dev 0 0; mmc swrite 28000000 4c00

5.3.3 sd 卡存储

将 img 放入 0x28000000 内存中， 并写入 kernel 分区中

- mw.b 28000000 ff 800000; fatload mmc 1 28000000 kernel.img
- mmc dev 0 0; mmc write 28000000 800 4000

将 dtb 放入 0x28000000 内存中，并写入 kernel dtb 分区中

- mw.b 28000000 ff 80000; fatload mmc 1 28000000 smartchip-sca200v100-evb.dtb
- mmc dev 0 0; mmc write 28000000 4800 400

rootfs.img 是 ext4 sparse image 使用如下命令:

- mw.b 28000000 ff 4000000; fatload mmc 1 28000000 rootfs.ext4.sp.img
- mmc dev 0 0; mmc swrite 28000000 4c00

5.4 linux 烧写 emmc

- echo 0 > /sys/block/mmcblk0boot0/force_ro
- dd bs=4k of=/dev/mmcblk0boot0 if=boot.img

- sync