产品升级功能调研报告

作者：滑国青

历史：2023.07.31 v0.1 创建

# 前言

本文档目的是对当前所有产品的升级方案进行调研，总结各升级方案的现状。重点关注升级方案的实现关键技术及可能存在的缺陷。

被调研产品包括：反制枪，雷达linux

涉及产品固件的升级及相应的上位机或C2升级工具及升级方式。

# 雷达linux升级方案

## 升级存储的组织

### flash划分

Flash 64M

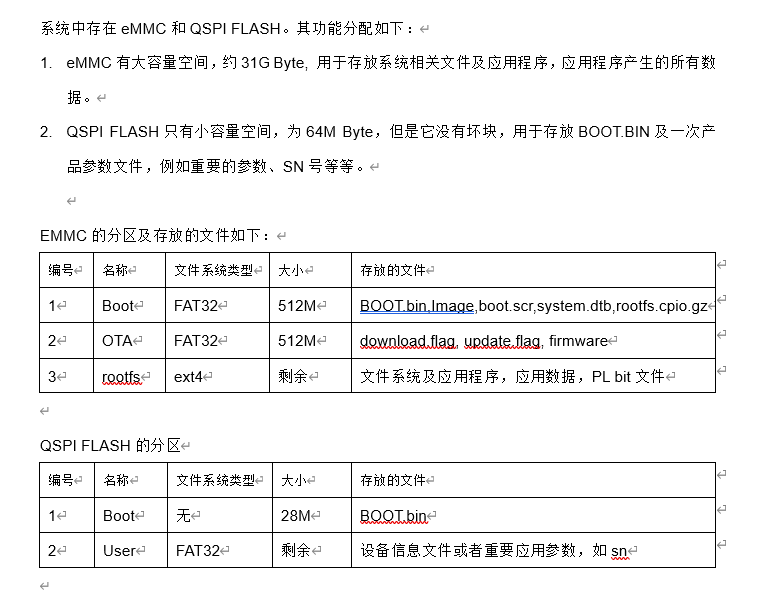
28M BOOT.bin 0x0地址烧

36M 数据区 FAT32文件系统 放设备重要参数信息如SN号。

设备是从flash引导，即从flash中读BOOT.BIN引导程序。

然后从emmc的首个分区读linux内核启动程序。

### Emmc的划分



## 升级包组织

### 项目代码组织

linux\_app/ 业务程序源码，cmake编译

alg/ 算法相关程序

fpga/ xsa文件

os/sdk petalinux的建构工程，由petalinux-create创建，由petalinux-build生成

生成uboot,kernel,rootfs,及基本的busybox,库，系统工具程序等等，

yocto建构的文件系统

os/modules/ 内核模块

wifi\_cypress/ wifi驱动

tools/

firmware\_package 多文件打包工具

### firmware\_package\_tool

tools\firmware\_package\_tool\firmware\_package

#### 说明

firmware\_package是一个将多个文件打包的工具。由main.c编译得到。

用法：

./firmware\_package 文件名1 文件名2 文件名n....

打包后输出为out.bin

将out.bin通过rtos\_app/build/build\_acur101\_OTA.py再次打包后，可得到用于OTA的升级包。

示例：

./tools/firmware\_package\_tool/firmware\_package sdk/images/linux/BOOT.BIN sdk/images/linux/boot.scr sdk/images/linux/xen sdk/images/linux/system.dtb sdk/images/linux/Image rootfs.cpio.gz

python rtos\_app/build/build\_acur101\_OTA.py rtos\_app/release/header\_info.ini out.bin V1.00

最后会生成V1.00\_OTA.bin

#### 打包原理

typedef struct file\_info\_s

{

    uint8\_t flag[4]; //“AAAA” magic值

    uint8\_t name[64]; //文件名称

    uint8\_t len[4]; //小字节序

}file\_info\_t;

1. 依次写各文件头信息file\_info\_t及文件数据。
2. 按字节写入，不对齐。

| file\_info\_t | fileA | file\_info\_t | fileB | file\_info\_t | fileC |…

### mk\_ota.sh

acur101\_ps\_linux\mk\_ota.sh

用法：mk\_ota.sh {版本号}

cd $(dirname $0)

cwd=$(pwd)

cd $(git rev-parse --show-toplevel)

VERSION=$(grep -E -o "EMBED\_SOFTWARE\_PS\_VERSION\_STR .\*" app/src/cfg/radar\_config.h | awk {'print $2'} | grep -o '[^"]\*')

rm -rf home/root/

mkdir -p home/root/

cp app/build/linux\_app.elf home/root

cp fpga/acur101\_fpga\_\*.bit home/root

cp os/sdk/mypl.dtbo home/root

tar -czf app.tar.gz home/root

image=os/sdk/images/linux

${cwd}/firmware\_package\_tool/firmware\_package ${image}/BOOT.BIN ${image}/boot.scr ${image}/system.dtb ${image}/Image ${image}/rootfs.cpio.gz app.tar.gz

python3 tools/package\_bin/build/build\_acur101\_OTA.py tools/package\_bin/release/header\_info.ini out.bin $VERSION

rm -rf out.bin home/root app.tar.gz

echo "........done........"

#### 业务程序打包

app.tar.gz 业务程序打成一个tgz包

linux\_app/build/linux\_app.elf 业务主程序

fpga/acur101\_fpga\_\*.bit

os/sdk/mypl.dtbo

#### 系统文件打包

将BOOT.BIN boot.scr system.dtb Image rootfs.cpio.gz app.tar.gz 用firmware\_package工具打包

BOOT.BIN uboot引导程序

boot.scr uboot引导脚本

Image linux kernel Image.

rootfs.cpio.gz initrd根文件系统

app.tar.gz 业务程序打包

#### 制作ota映像文件

acur101\_ps\_linux\rtos\_app\build\build\_acur101\_OTA.py

rtos\_app\release\header\_info.ini

headerLen:0X00000100

headerVersion:0X00000001

firmwareOffset:0X00001000

firmwareLen:0XFFFFFFFF

firmwareCrc:0XFFFFFFFF

corpInfoLen:0X00000020

pCorpInfo:Skyfend

devNamelen:0X00000020

pDevName:ACUR101

fwVersionLen:0X00000040

pFwVersion:

reserveLen:0X0000005C

headerCrc:0X00000000

数值采用的小序：

头部4KB大小。

非头部分用FF填充。

显然，header\_info.ini描述了：

（1）头长度，头版本号

（2）固件长度，固件文件偏移，CRC校验码

（3）公司名，公司名长度 Skyfend

（4）设备名，设备名长度 ACUR101

（5）固件版本号，版本号字符串长度

（7）保留长度

（8）头校验CRC

这显然是OTA头的文本描述，在生成OTA时，应该将这些信息，填到头结构中。然后是系统打包文件。

| OTA header info | 固件文件数据（firmware package格式）|

#### Uboot引导脚本boot.scr

# This is a boot script for U-Boot

# Generate boot.scr:

# mkimage -c none -A arm -T script -d boot.cmd.default boot.scr

#

################

fitimage\_name=image.ub

kernel\_name=Image

ramdisk\_name=ramdisk.cpio.gz.u-boot

rootfs\_name=rootfs.cpio.gz.u-boot

for boot\_target in ${boot\_targets};

do

    echo "Trying to load boot images from ${boot\_target}"

    if test "${boot\_target}" = "jtag" ; then

        booti 0x00200000 0x04000000 0x00100000

    fi

    if test "${boot\_target}" = "mmc0" || test "${boot\_target}" = "mmc1" || test "${boot\_target}" = "usb0" || test "${boot\_target}" = "usb1"; then

        if test -e ${devtype} ${devnum}:${distro\_bootpart} /uEnv.txt; then

            fatload ${devtype} ${devnum}:${distro\_bootpart} 0x00200000 uEnv.txt;

            echo "Importing environment(uEnv.txt) from ${boot\_target}..."

            env import -t 0x00200000 $filesize

            if test -n $uenvcmd; then

                echo "Running uenvcmd ...";

                run uenvcmd;

            fi

        fi

        if test -e ${devtype} ${devnum}:${distro\_bootpart} /${fitimage\_name}; then

            fatload ${devtype} ${devnum}:${distro\_bootpart} 0x10000000 ${fitimage\_name};

            bootm 0x10000000;

                fi

        if test -e ${devtype} ${devnum}:${distro\_bootpart} /${kernel\_name}; then

            fatload ${devtype} ${devnum}:${distro\_bootpart} 0x00200000 ${kernel\_name};

        fi

        if test -e ${devtype} ${devnum}:${distro\_bootpart} /system.dtb; then

            fatload ${devtype} ${devnum}:${distro\_bootpart} 0x00100000 system.dtb;

        fi

        if test -e ${devtype} ${devnum}:${distro\_bootpart} /${ramdisk\_name} && test "${skip\_tinyramdisk}" != "yes"; then

            fatload ${devtype} ${devnum}:${distro\_bootpart} 0x04000000 ${ramdisk\_name};

            booti 0x00200000 0x04000000 0x00100000

        fi

        if test -e ${devtype} ${devnum}:${distro\_bootpart} /${rootfs\_name} && test "${skip\_ramdisk}" != "yes"; then

            fatload ${devtype} ${devnum}:${distro\_bootpart} 0x04000000 ${rootfs\_name};

            booti 0x00200000 0x04000000 0x00100000

        fi

        booti 0x00200000 - 0x00100000

    fi

    if test "${boot\_target}" = "xspi0" || test "${boot\_target}" = "qspi" || test "${boot\_target}" = "qspi0"; then

        sf probe 0 0 0;

        sf read 0x10000000 0xF40000 0x6400000

        bootm 0x10000000;

        echo "Booting using Fit image failed"

        sf read 0x00200000 0xF00000 0x1D00000

        sf read 0x04000000 0x4000000 0x4000000

        booti 0x00200000 0x04000000 0x00100000;

        echo "Booting using Separate images failed"

    fi

    if test "${boot\_target}" = "nand" || test "${boot\_target}" = "nand0"; then

        nand info;

        nand read 0x10000000 0x4180000 0x6400000

        bootm 0x10000000;

        echo "Booting using Fit image failed"

        nand read 0x00200000 0x4100000 0x3200000

        nand read 0x04000000 0x7800000 0x3200000

        booti 0x00200000 0x04000000 0x00100000;

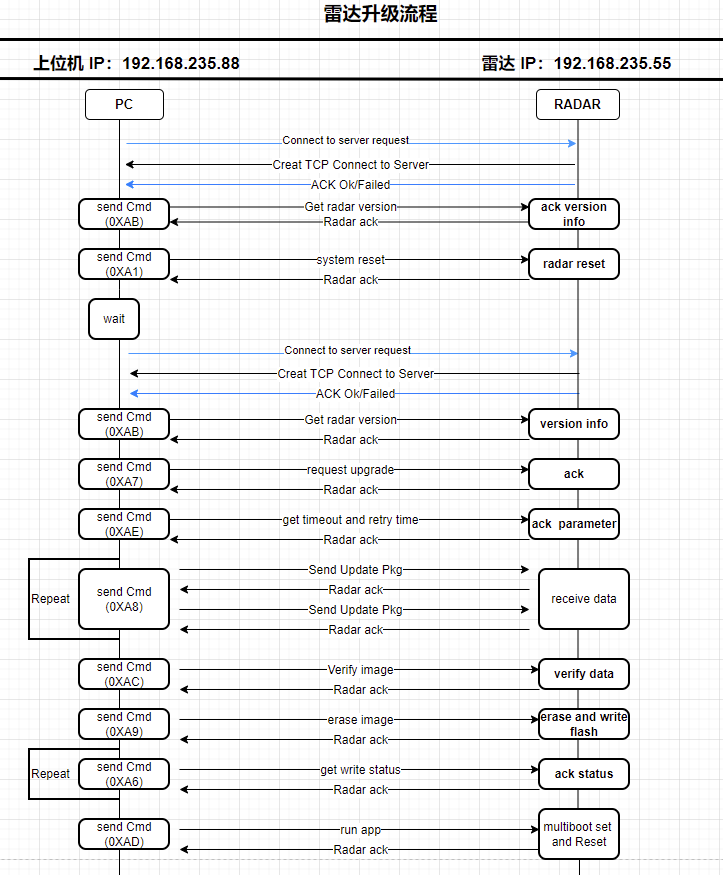
        echo "Booting using Separate images failed"

    fi

done

## 升级交互协议





## 升级方案

有两种情况，一种是完全在uboot下进行升级。

方案1：是在linux下完成下载固件到备份OTA区，然后重启由uboot完成将备份区更新到主区。

方案2：uboot启动后，检测到主区固件不完整，无法启动，直接进入OTA模式。然后通过网口完成下载固件并升级整个过程。

两个方案的区别在于谁下载固件，更新主分区固件的部分是一致的。

下载固件统一采用mavlink的升级消息进行下载。

## 方案的可能存在的缺陷

1. 缺少配置文件升迁机制。方案在更新主分区时，是将rootfs解压到ext4分区，然后以ext4分区作为根；来加快固件启动，但是由于旧固件文件系统内容的遗留，可能会造成新版本误用到旧版本的配置文件，导致固件运行异常。
2. 旧版本的根文件系统ext4未清理。

直接解压新版本覆盖旧根系统，可能存在旧系统文件残留。

# Pc端OTA工具

https://confluence.autel.com/display/APR100/acur100\_tool\_v2.2.2\_dev5

# 结束