**SHRD100-OTA升级-设计说明**

|  |  |
| --- | --- |
| 拟 制 |  |
| 审 核 |  |
| 会签 |  |
| 批 准 |  |

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **修订版本** | **日期** | **作者** | **修改描述** | **备注** |
| V1.0 | 2023.08.10 | 滑国青 | 初始版本 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 引言 3](#_Toc146550558)

[1.1 目的 3](#_Toc146550559)

[1.2 范围 3](#_Toc146550560)

[1.3 缩略语定义 3](#_Toc146550561)

[1.4 参考资料 3](#_Toc146550562)

[2 软件设计说明 3](#_Toc146550563)

[2.1 升级策略 3](#_Toc146550564)

[2.2 升级实现代码说明 4](#_Toc146550565)

[2.2.1 linux\_app程序中的处理 4](#_Toc146550566)

[2.3 关键问题 6](#_Toc146550567)

[2.3.1 固件完整性校验放哪里？ 6](#_Toc146550568)

[2.3.2 Rootfs是否需要展到emmc上加快启动？ 6](#_Toc146550569)

[2.3.3 Emmc格式化fat系统 6](#_Toc146550570)

[2.3.4 Uboot选择版本 12](#_Toc146550571)

[2.3.5 三个版本可以切换的思路 13](#_Toc146550572)

[2.3.6 Flash分区 13](#_Toc146550573)

[2.3.7 Uboot里如何获取引导方式 16](#_Toc146550574)

[2.3.8 固件文件的压缩格式确定 17](#_Toc146550575)

[2.3.9 Bootargs传参 18](#_Toc146550576)

[2.3.10 升级下载速度问题 18](#_Toc146550577)

[3 Issues解答 19](#_Toc146550578)

[3.1 如何生成ota版本 19](#_Toc146550579)

[3.2 升级中出现设备心跳发不出来，升级卡住问题 20](#_Toc146550580)

[3.3 升级后调用reboot重启时内核挂在wifi抓包处，导致无法重启 20](#_Toc146550581)

[4 附件 21](#_Toc146550582)

[4.1 雷达linux的升级机制分析 21](#_Toc146550583)

[4.2 uboot运行脚本 21](#_Toc146550584)

[4.3 uboot FIT 工具 22](#_Toc146550585)

[4.4 uboot的压缩解压命令 26](#_Toc146550586)

[4.4.1 lzmadec 26](#_Toc146550587)

[4.4.2 gzwrite 26](#_Toc146550588)

[4.4.3 unlz4 27](#_Toc146550589)

[4.4.4 unzip 27](#_Toc146550590)

[5 未完成的工作 28](#_Toc146550591)

[5.1 Uboot中如果要从flash中启动版本，需要实现一个sfloadimg命令 28](#_Toc146550592)

[6 结束 28](#_Toc146550593)

# 引言

## 目的

本文为“SHRD100-OTA升级-设计说明”，主要用于定义软件功能，供项目组开发人员和软件维护人员阅读。

## 范围

本文档只限于塞防科技项目组研发、测试、产品以及项目相关人员作为内部信息对齐使用，未经公司批准以及书面授权不允许任何人以任何形式对本文档复制、传播、改动。

## 缩略语定义

|  |  |  |
| --- | --- | --- |
| **缩略语** | **全称** | **描述** |
| OTA | On The Air | 指固件升级功能 |

## 参考资料

|  |  |
| --- | --- |
| **名称** | **版本** |
| SHRD100\_系统设计说明书 |  |
| SHRD100嵌入式软件设计说明V1.2.docx |  |
|  |  |
|  |  |
|  |  |

# 软件设计说明

## 升级策略

采用典型的AB面升级策略。

本设计的AB面升级策略。

1. 将emmc分成4个分区，其中前两个分区emmc1和emmc2是升级用分区，格式化为fat文件系统。后两个分区，备用，可存放日志，配置等文件。
2. 分区中含有版本文件：

├── BOOT.BIN 内含uboot,fsbl等引导文件，由petalinux-package生成

├── boot.scr uboot引导脚本

├── firmware.md5list 版本文件的md5校验信息

├── image.ub 内核和设备树文件，打成mkimage FIT格式

├── rootfs.cpio.gz initrd根文件系统 . 如果版本在flash里，由于空间有限，会压缩成rootfs.cpio.lzma格式。

├── system.bit.gz xilinx比特流文件，压缩成gz格式

└── version 版本号文件

└── ver\_selected.me 此文件不属于版本，在升级后，新版本位于哪个分区，则设置此标记文件。Uboot根据此标记决定从哪个分区版本引导。

（3）flash中可以存在一份备份版本，在出厂时烧录，当emmc分区中的版本都不可用时，可以回落到从flash中引导。

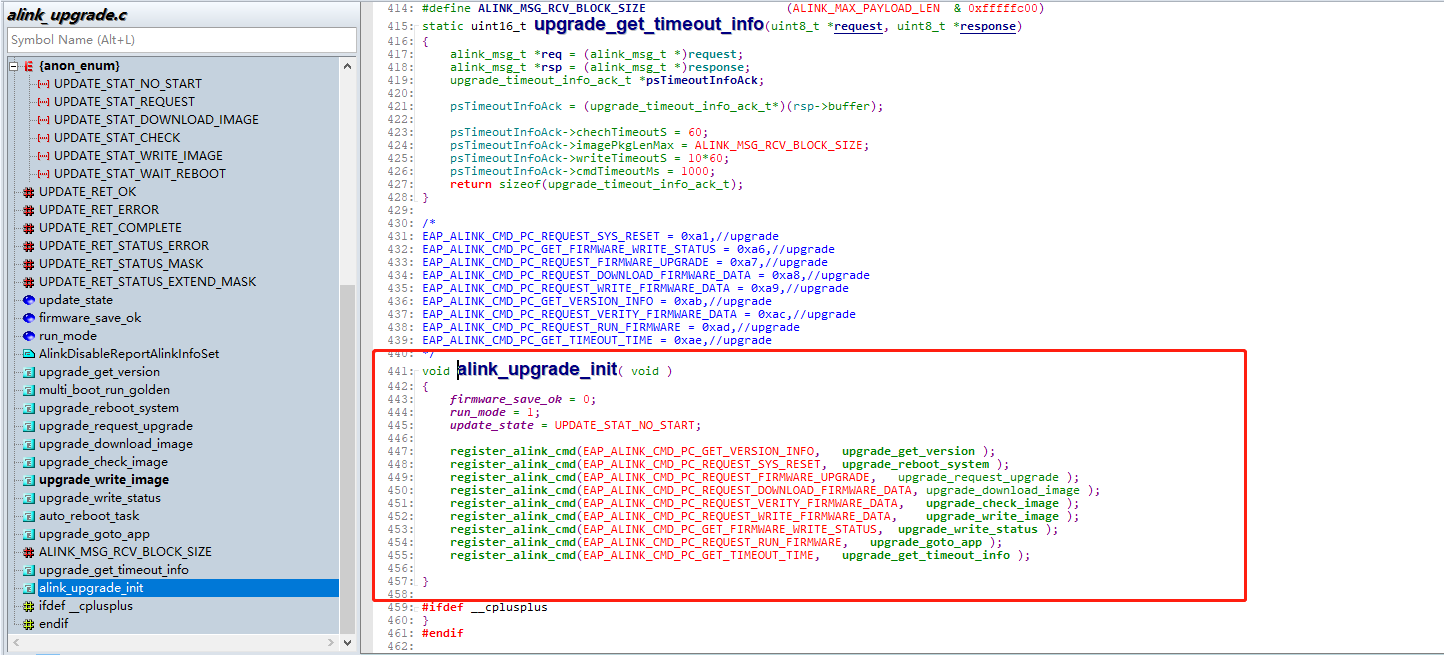
## 升级实现代码说明

### linux\_app程序中的处理

#### alink\_upgrade.c

linux\_app\src\tracer\hal\c2\_alink\upgrade

主要是实现alink升级协议的处理工作。



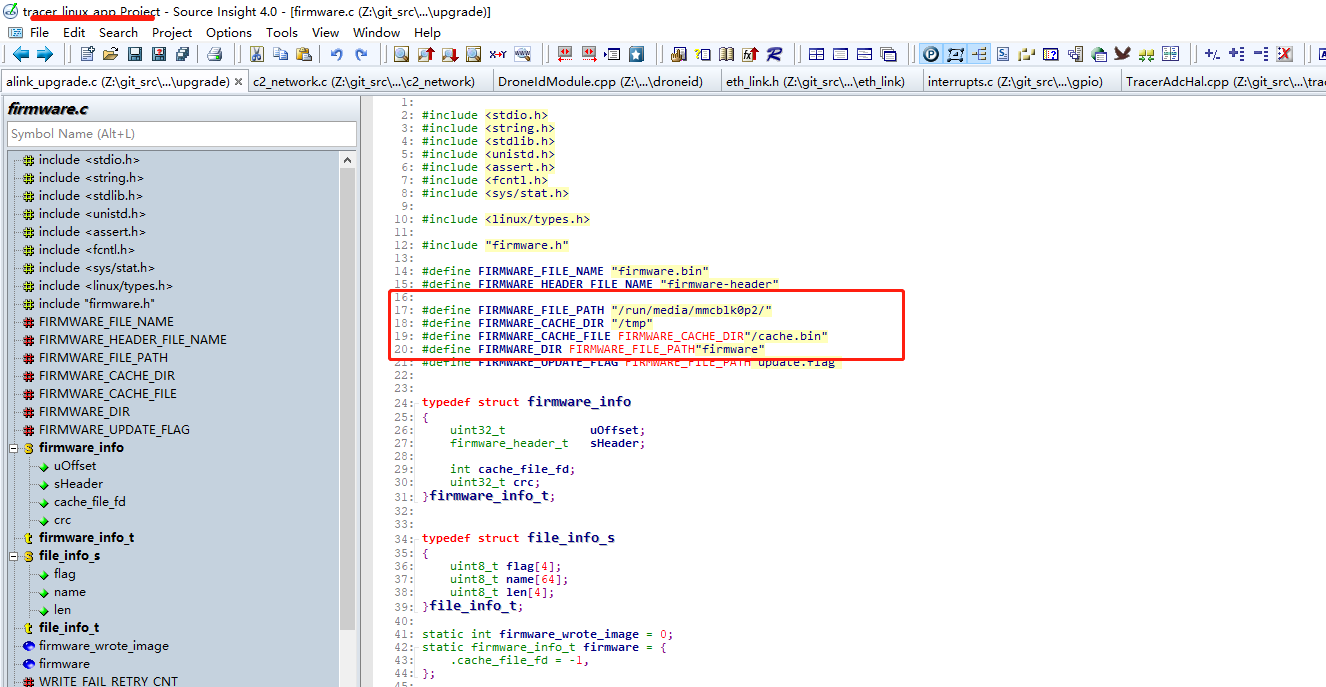
主要是处理以上图中几个升级命令。

#### firmware.c

linux\_app\src\tracer\hal\c2\_alink\upgrade

主要负责 升级ota文件的存储，校验和解压

解压后的文件，目前存放在：/run/media/mmcblk0p2/firmware



#### prepare\_to\_switch\_ver.sh

这是新旧版本切换脚本。

它在auto\_reboot\_task()（alink\_upgrade.c）中被调用。

主要工作是：

1. 检查当前旧版本的启动分区来决定新版本应该放在哪个分区。

比如：旧版本在emmc1分区，则新版本应该放在emmc2; 旧版本在emmc2分区，则新版本应该放在emmc1;

1. 将新版本解压后的文件从备份目录里，拷到对应分区的根目录下。
2. 在新版本分区上创建ver\_selected.me文件，然后删除旧版本分区上的ver\_selected.me文件

## 关键问题

### 固件完整性校验放哪里？

提供所有文件的校验列表文件

firmware.md5list

### Rootfs是否需要展到emmc上加快启动？

Rootfs使用ramdisk可能启动过慢，是否需要展到emmc分区，直接挂根。

当前项目紧，可以先不考虑展到emmc分区3中。优化时，可以实现此功能。

### Emmc格式化fat系统

生产时，可以烧录flash版本，flash版本首次启动后，可以对emmc格式化。

Emmc1,emmc2分区必须格式化为fat32,否则无法引导。

#### Uboot相关命令

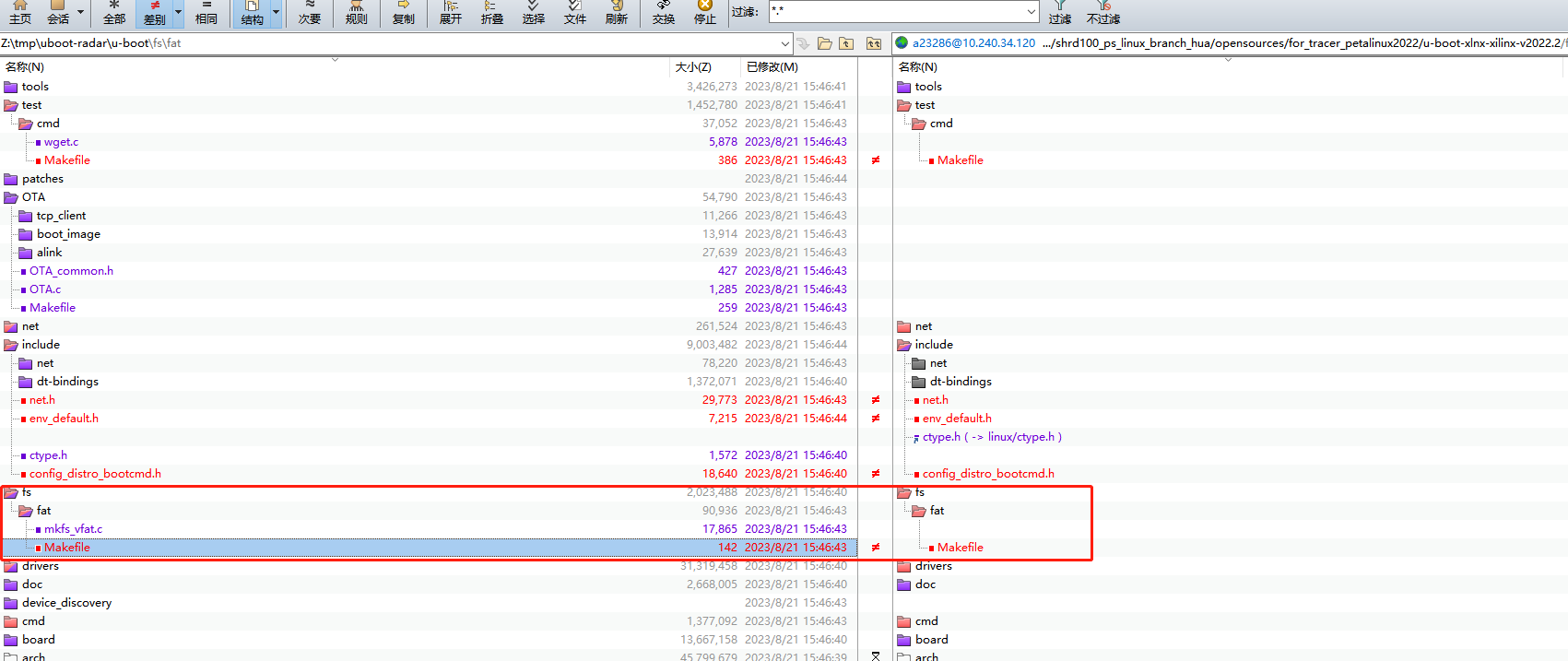
Uboot支持的fat,mbr命令

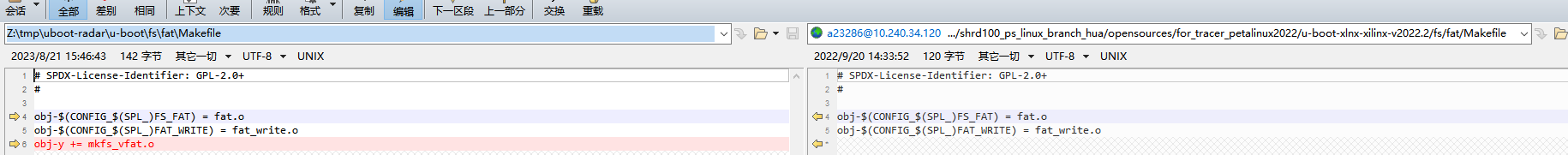
fatinfo fatload fatls fatmkdir fatrm fatsize fatwrite

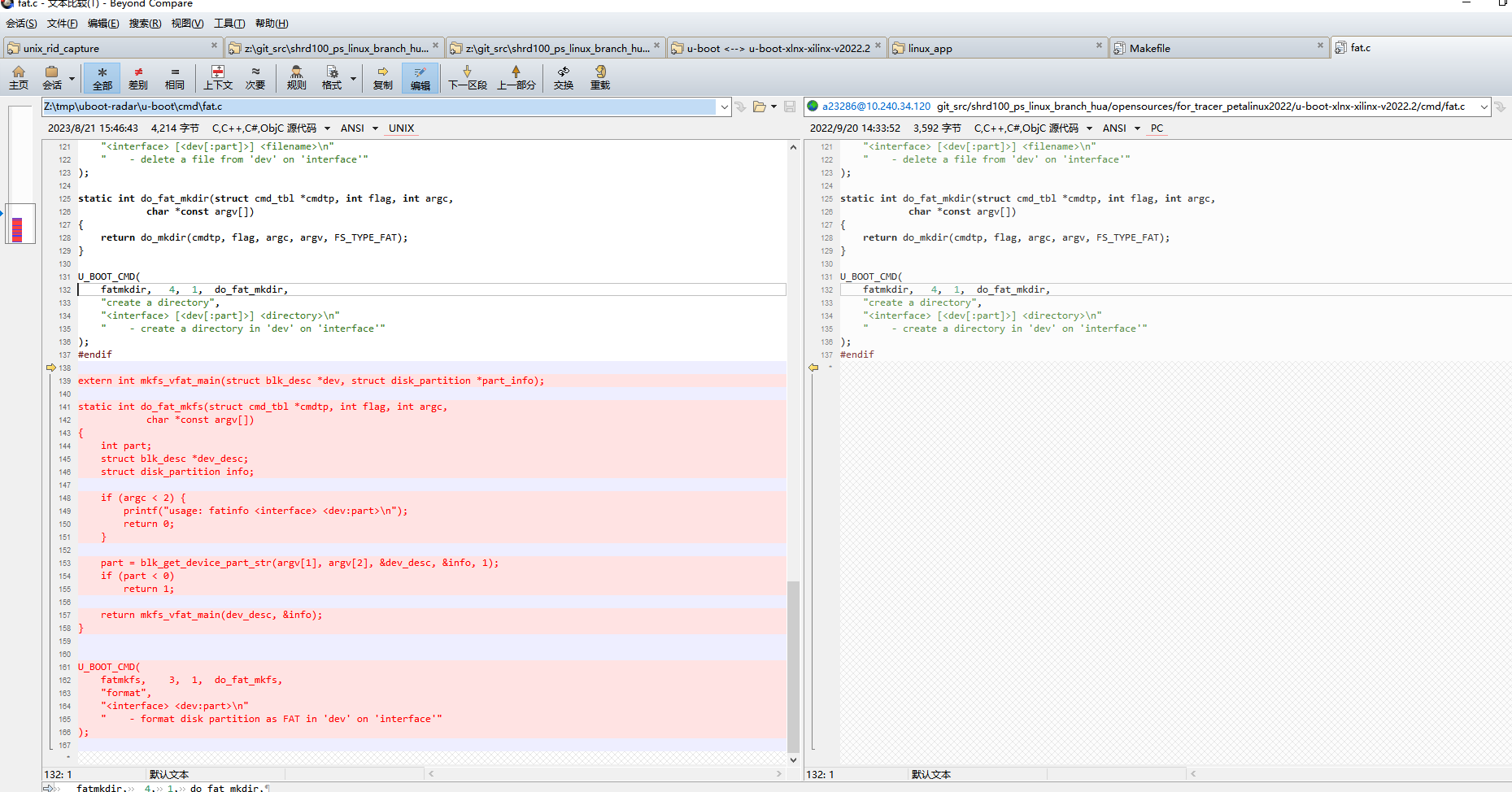
#### Uboot增加配置

CONFIG\_CMD\_MBR=y

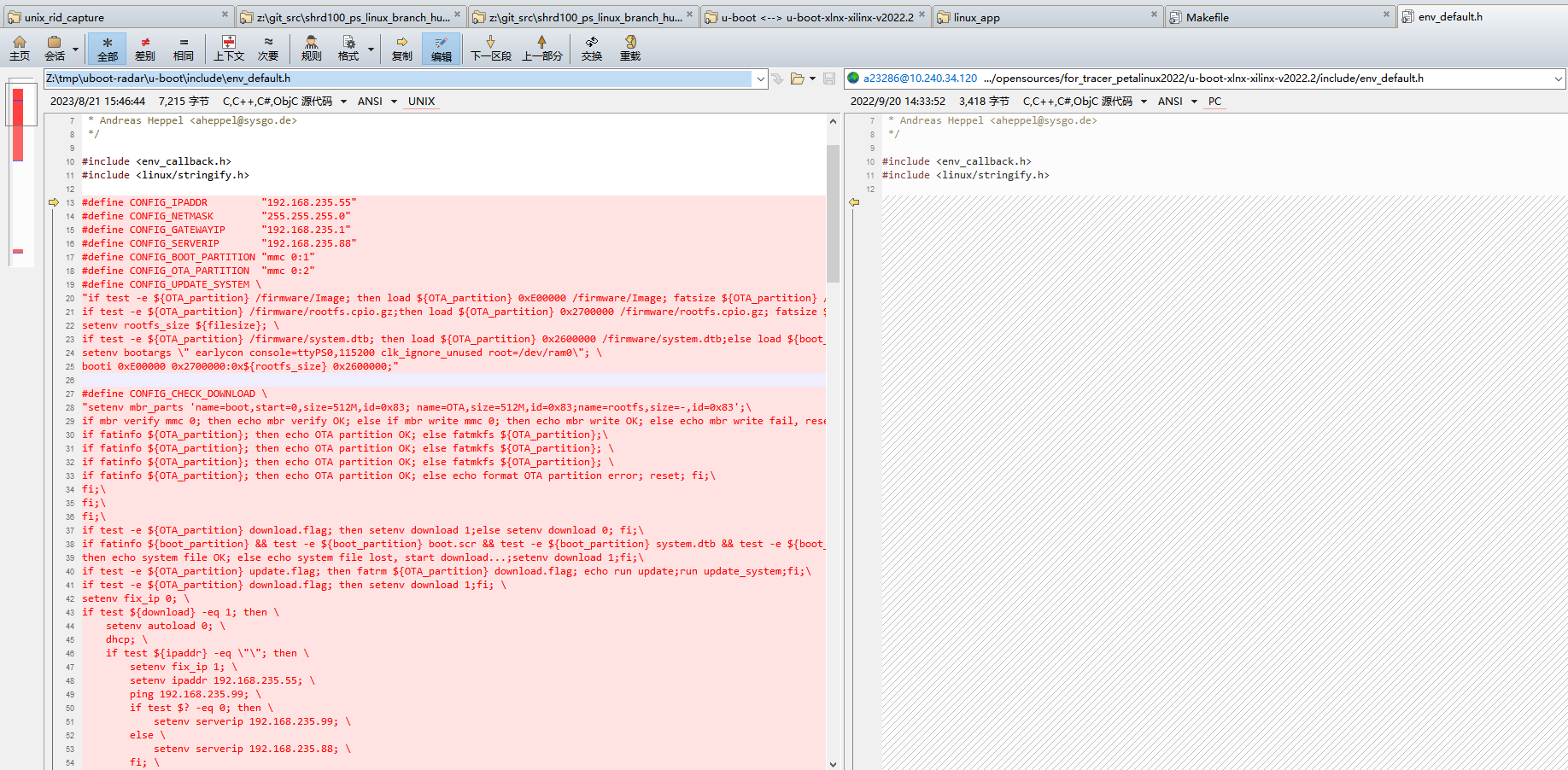
#### Uboot修改







#### 格式化脚本



#define CONFIG\_IPADDR "192.168.235.55"

#define CONFIG\_NETMASK "255.255.255.0"

#define CONFIG\_GATEWAYIP "192.168.235.1"

#define CONFIG\_SERVERIP "192.168.235.88"

#define CONFIG\_BOOT\_PARTITION "mmc 0:1"

#define CONFIG\_OTA\_PARTITION "mmc 0:2"

#define CONFIG\_UPDATE\_SYSTEM \

"if test -e ${OTA\_partition} /firmware/Image; then load ${OTA\_partition} 0xE00000 /firmware/Image; fatsize ${OTA\_partition} /firmware/Image; else load ${boot\_partition} 0xE00000 Image;fatsize ${boot\_partition} Image;fi; \

if test -e ${OTA\_partition} /firmware/rootfs.cpio.gz;then load ${OTA\_partition} 0x2700000 /firmware/rootfs.cpio.gz; fatsize ${OTA\_partition} /firmware/rootfs.cpio.gz; else load ${boot\_partition} 0x2700000 rootfs.cpio.gz;fatsize ${boot\_partition} rootfs.cpio.gz;fi; \

setenv rootfs\_size ${filesize}; \

if test -e ${OTA\_partition} /firmware/system.dtb; then load ${OTA\_partition} 0x2600000 /firmware/system.dtb;else load ${boot\_partition} 0x2600000 system.dtb; fi; \

setenv bootargs \" earlycon console=ttyPS0,115200 clk\_ignore\_unused root=/dev/ram0\"; \

booti 0xE00000 0x2700000:0x${rootfs\_size} 0x2600000;"

#define CONFIG\_CHECK\_DOWNLOAD \

"setenv mbr\_parts 'name=boot,start=0,size=512M,id=0x83; name=OTA,size=512M,id=0x83;name=rootfs,size=-,id=0x83';\

if mbr verify mmc 0; then echo mbr verify OK; else if mbr write mmc 0; then echo mbr write OK; else echo mbr write fail, reset....; reset; fi;fi;\

if fatinfo ${OTA\_partition}; then echo OTA partition OK; else fatmkfs ${OTA\_partition};\

if fatinfo ${OTA\_partition}; then echo OTA partition OK; else fatmkfs ${OTA\_partition}; \

if fatinfo ${OTA\_partition}; then echo OTA partition OK; else fatmkfs ${OTA\_partition}; \

if fatinfo ${OTA\_partition}; then echo OTA partition OK; else echo format OTA partition error; reset; fi;\

fi;\

fi;\

fi;\

if test -e ${OTA\_partition} download.flag; then setenv download 1;else setenv download 0; fi;\

if fatinfo ${boot\_partition} && test -e ${boot\_partition} boot.scr && test -e ${boot\_partition} system.dtb && test -e ${boot\_partition} Image; \

then echo system file OK; else echo system file lost, start download...;setenv download 1;fi;\

if test -e ${OTA\_partition} update.flag; then fatrm ${OTA\_partition} download.flag; echo run update;run update\_system;fi;\

if test -e ${OTA\_partition} download.flag; then setenv download 1;fi; \

setenv fix\_ip 0; \

if test ${download} -eq 1; then \

setenv autoload 0; \

dhcp; \

if test ${ipaddr} -eq \"\"; then \

setenv fix\_ip 1; \

setenv ipaddr 192.168.235.55; \

ping 192.168.235.99; \

if test $? -eq 0; then \

setenv serverip 192.168.235.99; \

else \

setenv serverip 192.168.235.88; \

fi; \

fi; \

fi;\

\

while test ${download} -eq 1; do \

if test ${fix\_ip} -eq 1; then \

echo seriver ip is ${serverip}; \

else \

setenv serverip 0.0.0.0 ; \

device\_discovery; \

if test ${serverip} = 0.0.0.0 ; then \

ping 192.168.235.99; \

if test $? -eq 0; then \

setenv serverip 192.168.235.99 ; \

else \

ping 192.168.235.88; \

if test $? -eq 0; then \

setenv serverip 192.168.235.88 ; \

fi; \

fi; \

fi; \

fi; \

\

OTA; \

if test -e ${OTA\_partition} update.flag; then \

setenv download 0; \

else \

sleep 3; \

if test ${fix\_ip} -eq 1; then \

ping 192.168.235.99; \

if test $? -eq 0; then \

setenv serverip 192.168.235.99; \

else \

ping 192.168.235.88; \

if test $? -eq 0; then \

setenv serverip 192.168.235.88; \

fi; \

fi;\

fi; \

fi;\

done;\

if test -e ${OTA\_partition} update.flag; then fatrm ${OTA\_partition} download.flag; echo run update;run update\_system;fi;\

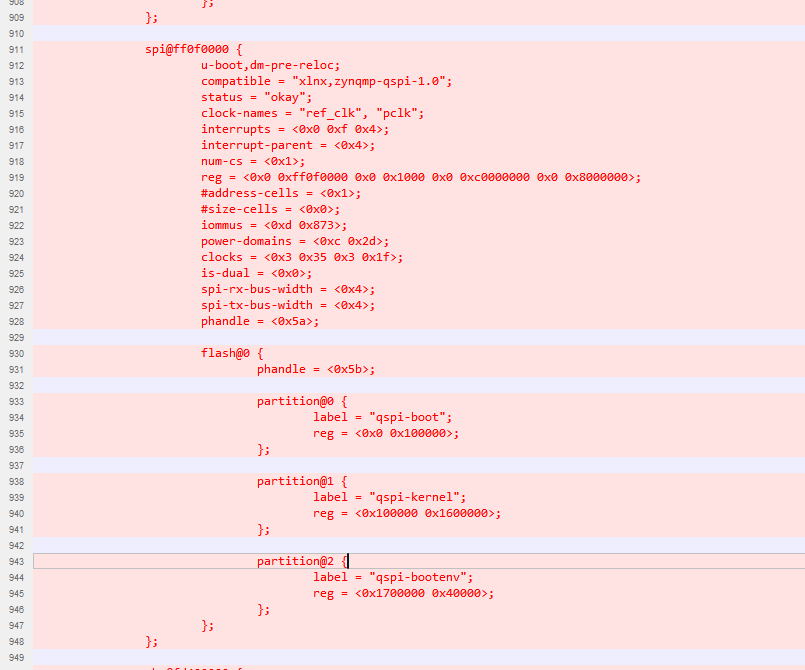
"

#### Flash

这是xsa生成的flash信息

ZynqMP> sf probe 0

SF: Detected is25wp512 with page size 256 Bytes, erase size 64 KiB, total 64 MiB



### Uboot选择版本

请参见：kernel/sdk\_tracer\_petalinux20222/boot\_cfg/uboot.script

基本原理是：emmc分成4个分区；其中第1，2用于版本启动，采用A，B面倒换升级策略。

ver\_selected.me文件在哪个分区，则从哪个分区的版本引导系统。

启动时，会向bootargs中注入当前启动分区的变量modeboot和ver\_selected来提示当前从哪个分区启动。

启动后，可以查看/proc/cmdline文件，来了解启动信息。

如：

root@sdktracerpetalinux20222:~# cat /proc/cmdline

initrd=0x20000000,0x373c4ad modeboot=sdboot ver\_selected=emmc2

#### 通过bootargs传参版本选择结果

setenv bootargs “${ bootargs } bootfrom=[emmcA|emmcB|flash]

### 三个版本可以切换的思路

（1）测试时，可以在同一设备上，进行最多三个版本的切换，达到对比测试目的。

（2）升级时flash中的版本，作为首个烧录版本，作为后备版本，在emmc不可用时，仍可以运行。

（3）每个版本有带有一个boot.scr，来实现uboot与后序引导的解耦。实现uboot之后加载方式的不断迭代优化能力。

（4）工厂生产时，通过flash烧录，得到flash上的首个版本，然后之后升级会使用emmc保存后序版本。

（5）可以实现RTOS与linux双版本的测试。通过设置从flash引导，然后根据引导标记，进行引导重定向。

Boot.scr实现在3个版本之间的跳转。 优先emmcA,其次emmcB,最后flash

通过引导标记文件，可以实现3者仲裁。

### Flash分区

BOOT.BIN 2M 0x0(0x200000)

boot.scr 128K 0x200000(0x20000)

system.bit.gz 4M – 128K 0x2200000(0x1e0000)

image.ub(不含rootfs) 10M 0x400000(0xa00000)

rootfs.cpio.lzma 46M 0x1000000(0x2e00000)

rsv 2M 0x3e00000(0x200000)

目前由于rootfs太大，rootfs.cpio.gz 有60多M

1743064 8月 21 20:44 BOOT.BIN

154664960 8月 21 20:30 rootfs.cpio

56029875 8月 21 20:30 rootfs.cpio.gz

36187017 8月 16 21:25 rootfs.cpio.lzma

7113279 8月 16 21:17 system.bit

2160313 8月 16 21:17 system.bit.gz

1733899 8月 16 21:17 system.bit.lzma

65626312 8月 21 20:31 image.ub

$dumpimage -l images/linux/image.ub

FIT description: Kernel fitImage for PetaLinux/5.15.36+git999/zynqmp-generic

Created: Mon Dec 20 10:57:12 2021

Image 0 (kernel-1)

Description: Linux kernel

Created: Mon Dec 20 10:57:12 2021

Type: Kernel Image

Compression: gzip compressed

Data Size: 9241226 Bytes = 9024.63 KiB = 8.81 MiB

Architecture: AArch64

OS: Linux

Load Address: 0x00200000

Entry Point: 0x00200000

Hash algo: sha256

Hash value: a78620252e708557840d1a7e697c166b0814b16f785f1dbce19d723c0eee3113

Image 1 (fdt-system-top.dtb)

Description: Flattened Device Tree blob

Created: Mon Dec 20 10:57:12 2021

Type: Flat Device Tree

Compression: uncompressed

Data Size: 48563 Bytes = 47.42 KiB = 0.05 MiB

Architecture: AArch64

Hash algo: sha256

Hash value: 221dbd3ca6d44bea77df62e90c1baefbdd9ddd62de385987f0dbba604e0ea266

Image 2 (ramdisk-1)

Description: petalinux-image-minimal

Created: Mon Dec 20 10:57:12 2021

Type: RAMDisk Image

Compression: uncompressed

Data Size: 56334402 Bytes = 55014.06 KiB = 53.72 MiB

Architecture: AArch64

OS: Linux

Load Address: unavailable

Entry Point: unavailable

Hash algo: sha256

Hash value: 78ad5f839919bc8acb25ab22ebf796afde76feab24049972384a2b2c4e2c4895

Default Configuration: 'conf-system-top.dtb'

Configuration 0 (conf-system-top.dtb)

Description: 1 Linux kernel, FDT blob, ramdisk

Kernel: kernel-1

Init Ramdisk: ramdisk-1

FDT: fdt-system-top.dtb

Hash algo: sha256

Hash value: unavailable

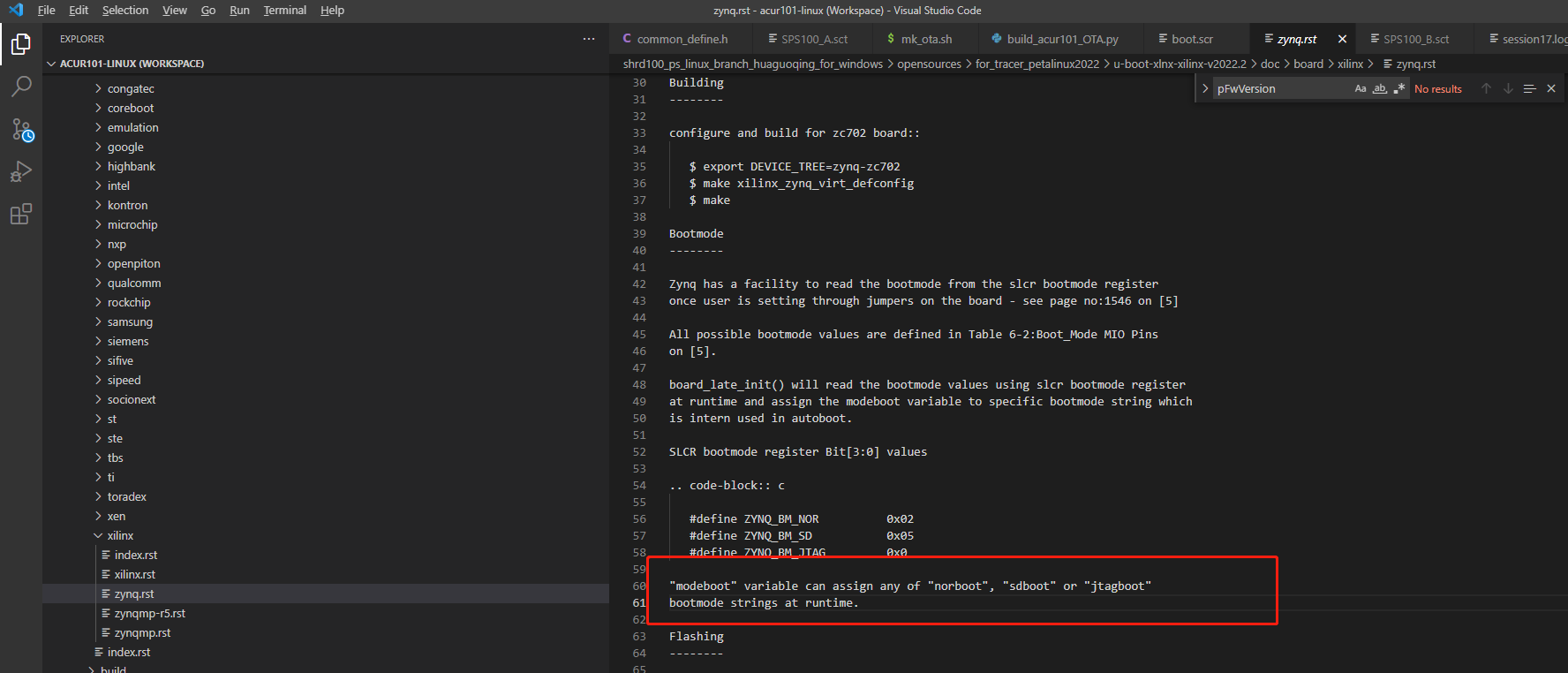
### Uboot里如何获取引导方式

opensources\for\_tracer\_petalinux2022\u-boot-xlnx-xilinx-v2022.2\doc\board\xilinx\zynq.rst

uboot里有个环境变量

ZynqMP> printenv modeboot

modeboot=sdboot



### 固件文件的压缩格式确定

System.bit 采用gzip压缩

7113279 8月 16 21:17 system.bit

2160313 8月 16 21:17 system.bit.gz

2451247 8月 22 10:42 system.bit.lz4

1733899 8月 16 21:17 system.bit.lzma

Rootfs 在flash中，采用lzma压缩,以获得最大压缩率，节省flash空间

Emmc版本，采用gzip压缩，以获得最快的解压速度。

Image.ub在压缩时，要生成两个版本，一个是flash的版本，一个是emmc的版本。

#### Rootfs使用cpio.gz格式。

55M，用时2秒。

[ 2.681729] Trying to unpack rootfs image as initramfs...

[ 2.687570] armv8-pmu pmu: hw perfevents: no interrupt-affinity property, guessing.

[ 2.694905] hw perfevents: enabled with armv8\_pmuv3 PMU driver, 7 counters available

[ 4.882836] Freeing initrd memory: 55060K

[ 4.939475] Initialise system trusted keyrings

#### Rootfs使用cpio.lzma

37M,用时30秒

[ 2.681704] Trying to unpack rootfs image as initramfs...

[ 2.687089] armv8-pmu pmu: hw perfevents: no interrupt-affinity property, guessing.

[ 4.546412] hw perfevents: enabled with armv8\_pmuv3 PMU driver, 7 counters available

[ 31.562976] Freeing initrd memory: 37164K

通过降低压缩率，看能不能加速解压速度。

采用-0级，变为45M，但是解压速度使用33秒，还变慢了4秒。

lzma -k -z -0 -T 0 rootfs.cpio

[ 2.681705] Trying to unpack rootfs image as initramfs...

[ 2.687070] armv8-pmu pmu: hw perfevents: no interrupt-affinity property, guessing.

[ 2.748162] hw perfevents: enabled with armv8\_pmuv3 PMU driver, 7 counters available

[ 35.783929] Freeing initrd memory: 45120K

### Bootargs传参

setenv bootargs initrd=<initrd\_addr>,<initrd\_size>

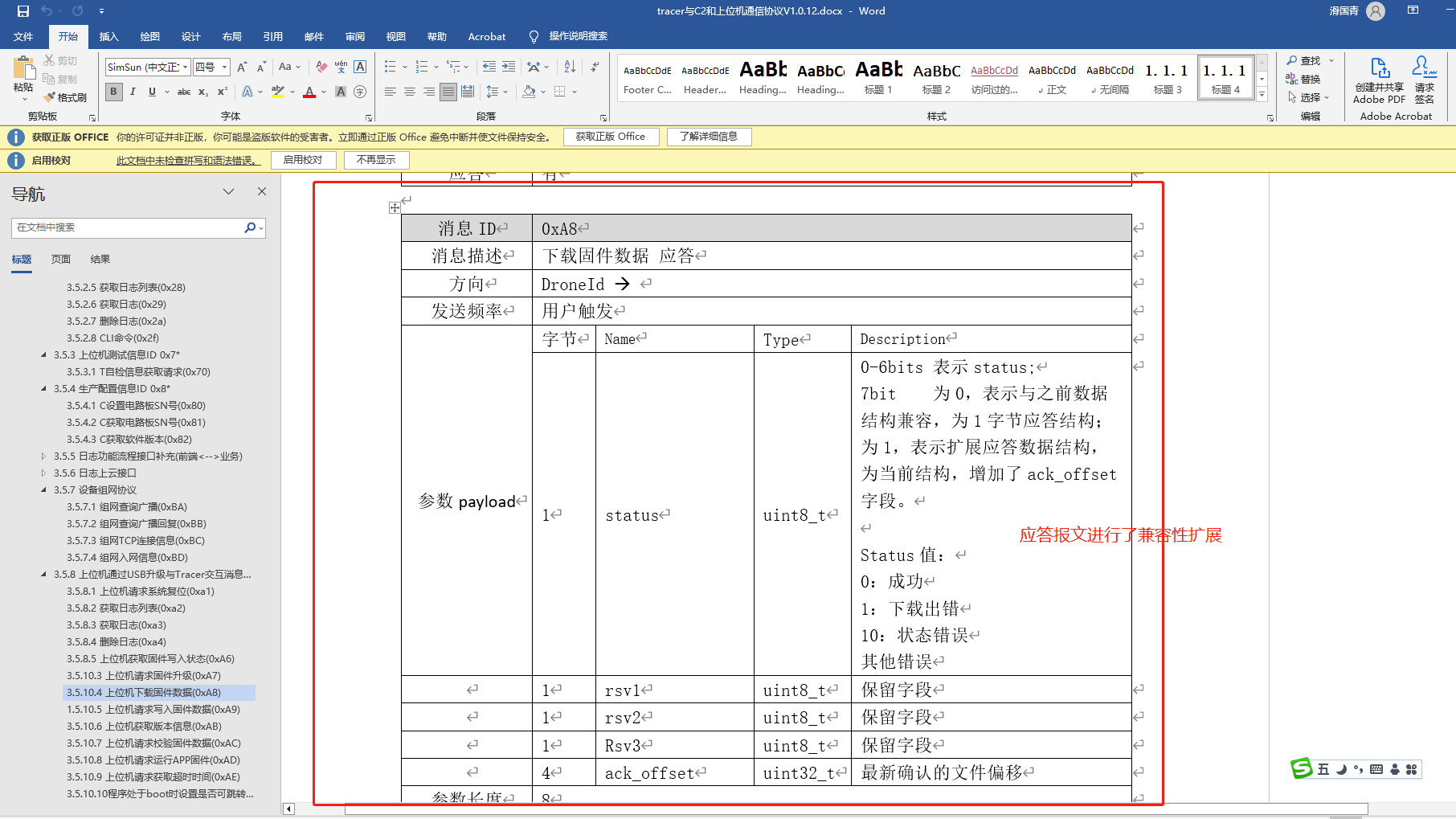
### 升级下载速度问题

目前升级下载采用一应一答的串行方式传输文件，下载时间对于wifi大约在3-4分钟，typec大约在2分钟。

但如果采用一次发送多个分片，即流水式并发发送分片，通过限制并发度，来进行简单的流控。实际测试是可以大大加快下载速度，目前测试wifi下载可以在30秒完成，但由于C2方面认为方案实现有一定的困难，所以最终未实现。

为了支持流水式并发下载，对alink的a8消息，进行了扩展改造

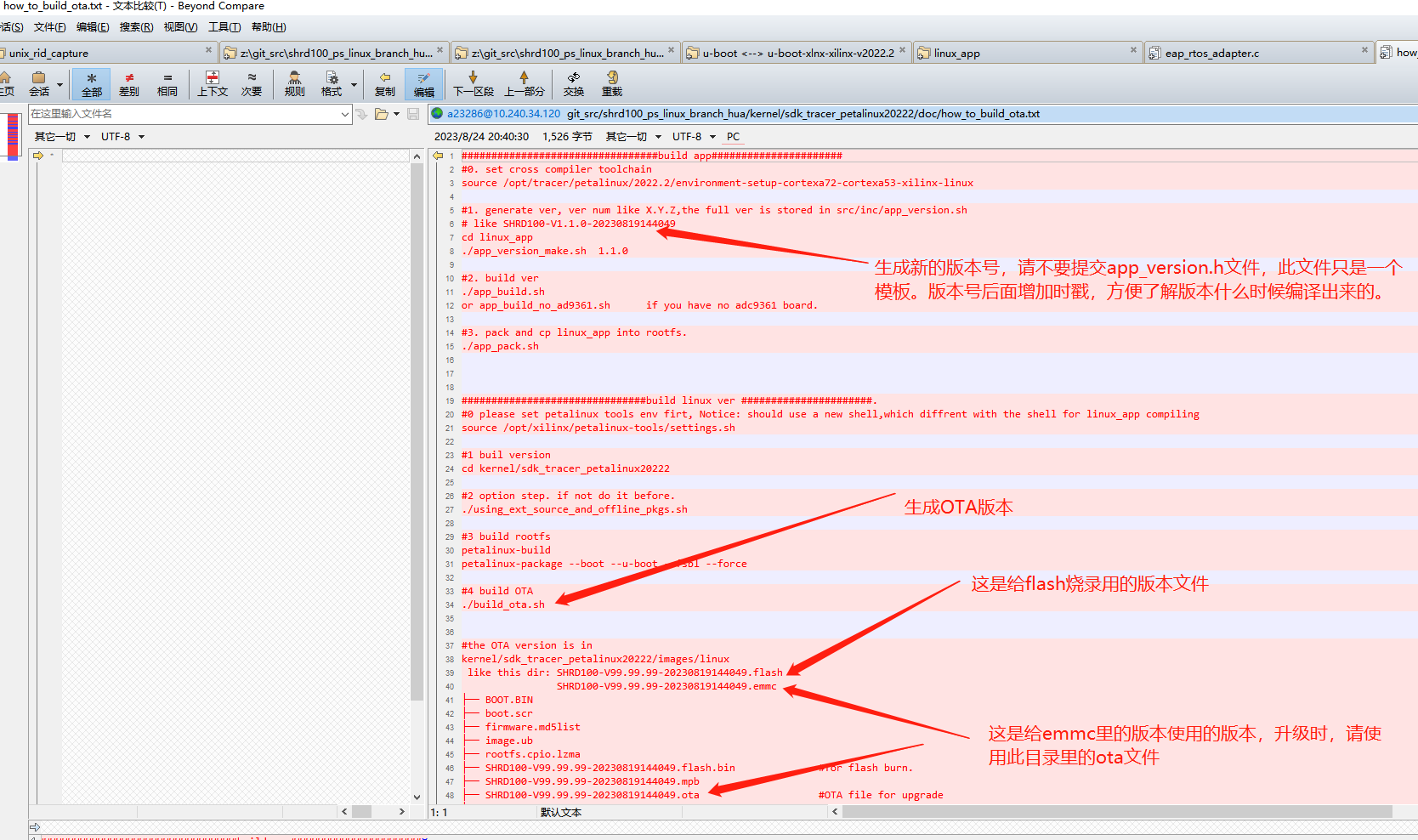
请参见：tracer与C2和上位机通信协议V1.0.12.docx



# Issues解答

## 如何生成ota版本

请参见kernel/sdk\_tracer\_petalinux20222/ how\_to\_unpack\_ota.txt



## 升级中出现设备心跳发不出来，升级卡住问题

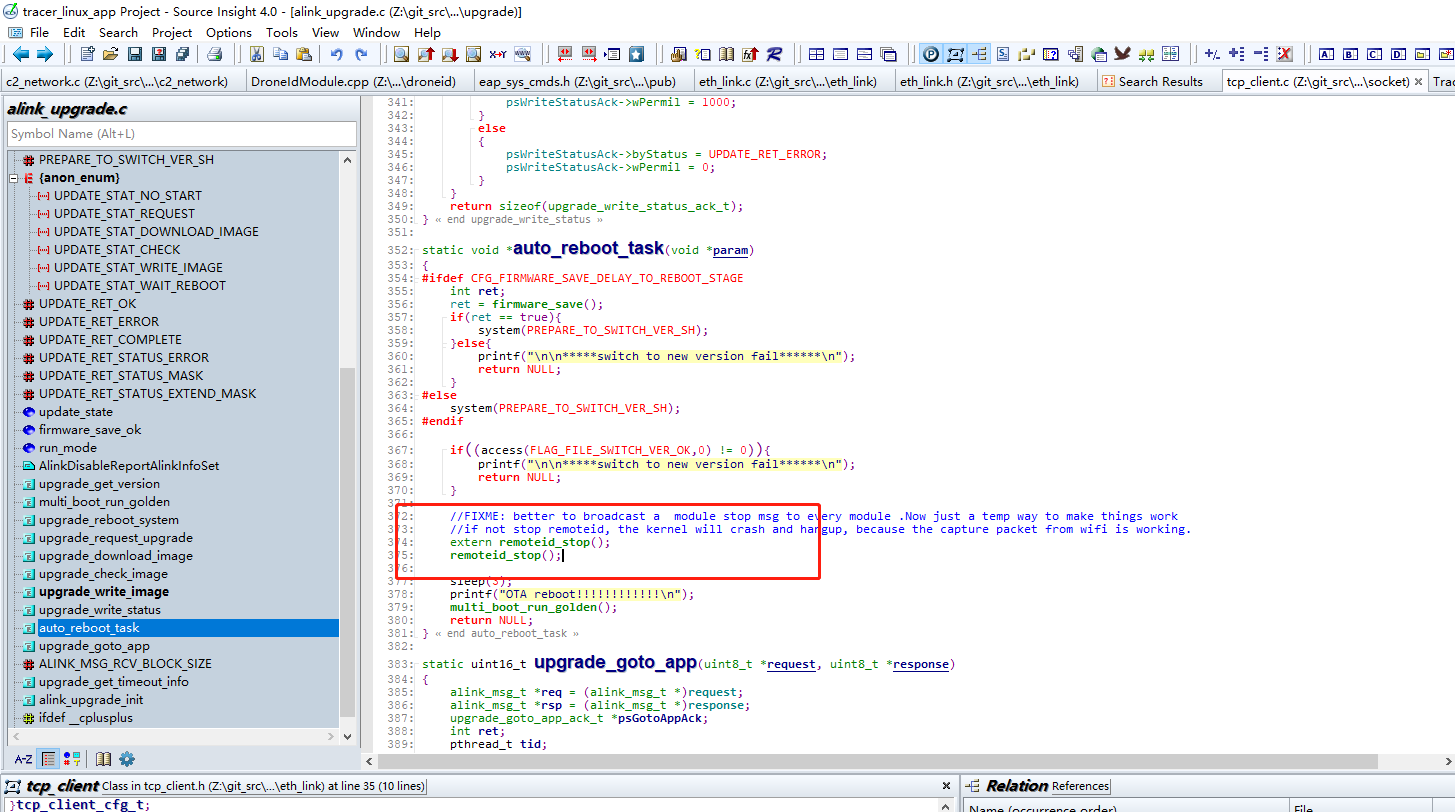
经过分析，是升级文件写emmc导致，将升级临时文件放在内存中，并关闭日志记录功能，问题解决。

说明周期的文件系统写缓冲，会导致app运行时的消息收发出现暂停问题。

## 升级后调用reboot重启时内核挂在wifi抓包处，导致无法重启

原因：remoteId在抓包状态，重启时抓包没有提前停掉。

解决办法：在升级完成重启前，需要给各模块发停止工作的通知，其中重点要保证remoteId退出抓包。目前先实现一个让remoteId停止工作的接口，保证重启可以不被挂住，至于发重启广播，因为当前模块的消息广播机制还没有，所以暂不实现。



# 附件

## 雷达linux的升级机制分析

[Merge branch 'user/chenyili\_ota' into 'feature/embed' (697fb010) · Commits · acur / acur101\_ps\_linux · GitLab (autel.com)](https://adasgitlab.autel.com/acur/acur101_ps_linux/-/commit/697fb0102decc893268bf1316c1376c28153a0fc)

[**app/src/srv/alink/command/upgrade/alink\_upgrade.c**](https://adasgitlab.autel.com/acur/acur101_ps_linux/-/commit/697fb0102decc893268bf1316c1376c28153a0fc#cf1e95fef4b0dcae7dd878ec8d7aed73e4a2dffc)

[**app/src/srv/alink/command/upgrade/firmware.c**](https://adasgitlab.autel.com/acur/acur101_ps_linux/-/commit/697fb0102decc893268bf1316c1376c28153a0fc#8af593606dbbcc4b6ece0754c4929c37f71d44b2)

## uboot运行脚本

1， 在uboot记录命令的运行时间，可以打开CONFIG\_CMD\_TIME。

2， autoscr已经改变成了source命令，可以打开CONFIG\_SOURCE，source可以执行内存中脚本，该脚本是由mkimage生成的。

#!/bin/sh  
file=$1  
result=$2  
if [ $# -lt 1 ]  
then  
        echo "$0 file\_name result\_file\_name"                                                                                                                                                                 
fi  
mkimage -A arm -O linux -T script -C none -n "uboot-nor script" -d $file $result

3，mkimage在Host端的安装：

sudo apt-get install u-boot-tools

4， 可以生成一个txt文件记录要执行的脚本内容，由于是txt文件所以没有for循环，如果要这样做的话，可以由外部的脚本帮你生成多少遍的循环内容，自己就不用麻烦做循环的考虑了。

5，对uboot添加了ext4支持的话，可以由以下命令加载mkimage生成的内容到内存中去。

ext4load mmc 0:5 0x00001000 xxx.img

6，调用source

source 0x00001000

## uboot FIT 工具

在U-Boot中，FIT（Flattened Image Tree）工具是用于处理和生成FIT格式镜像的工具。FIT格式是一种用于存储多个固件（如内核、设备树、ramdisk等）的容器格式，它可以方便地管理和引导多个固件。

U-Boot FIT工具提供了以下主要功能：

1. 创建FIT配置文件：使用mkimage命令创建FIT配置文件，该文件描述了FIT镜像中包含的固件及其相关属性。
2. 添加固件：使用mkimage命令的-f选项，可以将固件添加到FIT配置文件中。
3. 生成FIT镜像：使用mkimage命令的-E选项，可以根据FIT配置文件生成FIT镜像。
4. 验证FIT镜像：使用fit\_check\_sign命令可以验证FIT镜像的签名。
5. 引导FIT镜像：使用bootm命令可以引导FIT镜像，U-Boot会根据FIT配置文件中的属性选择合适的固件进行引导。

以下是使用U-Boot FIT工具的一些常用命令示例：

1. 创建FIT配置文件：

复制代码

mkimage -f fit.conf fit.itb

其中，fit.conf是FIT配置文件的内容，fit.itb是生成的FIT镜像文件。

1. 添加固件：

复制代码

mkimage -f fit.conf -F <firmware\_type> -n <firmware\_name> -p <firmware\_path> fit.itb

其中，<firmware\_type>是固件类型（如kernel、ramdisk、dtb等），<firmware\_name>是固件名称，<firmware\_path>是固件文件的路径。

1. 生成FIT镜像：

复制代码

mkimage -E -f fit.conf fit.itb

其中，fit.conf是FIT配置文件的内容，fit.itb是生成的FIT镜像文件。

1. 验证FIT镜像：

复制代码

fit\_check\_sign fit.itb

其中，fit.itb是要验证的FIT镜像文件。

1. 引导FIT镜像：

复制代码

bootm <fit\_addr>

其中，<fit\_addr>是FIT镜像在内存中的起始地址。

注意：在使用FIT工具之前，需要确保U-Boot已经正确配置了相关的编译选项，并且支持FIT格式镜像的处理和引导功能。

$dumpimage -l image.ub

FIT description: Kernel fitImage for PetaLinux/5.15.36+git999/zynqmp-generic

Created: Mon Dec 20 10:57:12 2021

Image 0 (kernel-1)

Description: Linux kernel

Created: Mon Dec 20 10:57:12 2021

Type: Kernel Image

Compression: gzip compressed

Data Size: 9241226 Bytes = 9024.63 KiB = 8.81 MiB

Architecture: AArch64

OS: Linux

Load Address: 0x00200000

Entry Point: 0x00200000

Hash algo: sha256

Hash value: 6bc240c6404821a7ea143ffaf4121ac5205100a667e3ec8f6a4eeef0397ab34c

Image 1 (fdt-system-top.dtb)

Description: Flattened Device Tree blob

Created: Mon Dec 20 10:57:12 2021

Type: Flat Device Tree

Compression: uncompressed

Data Size: 48563 Bytes = 47.42 KiB = 0.05 MiB

Architecture: AArch64

Hash algo: sha256

Hash value: 221dbd3ca6d44bea77df62e90c1baefbdd9ddd62de385987f0dbba604e0ea266

Image 2 (ramdisk-1)

Description: petalinux-image-minimal

Created: Mon Dec 20 10:57:12 2021

Type: RAMDisk Image

Compression: uncompressed

Data Size: 56334401 Bytes = 55014.06 KiB = 53.72 MiB

Architecture: AArch64

OS: Linux

Load Address: unavailable

Entry Point: unavailable

Hash algo: sha256

Hash value: 14a04a37eb1b34f8ceac614a796a95392263659e6d152d352bf5c3acf5c5e029

Default Configuration: 'conf-system-top.dtb'

Configuration 0 (conf-system-top.dtb)

Description: 1 Linux kernel, FDT blob, ramdisk

Kernel: kernel-1

Init Ramdisk: ramdisk-1

FDT: fdt-system-top.dtb

Hash algo: sha256

Hash value: unavailable

dumpimage -T flat\_dt -p 0 -o tracer.kernel image.ub

dumpimage -T flat\_dt -p 1 -o tracer.dtb image.ub

dumpimage -T flat\_dt -p 2 -o tracer.rootfs.cpio.gz image.ub

## uboot的压缩解压命令

解压后文件大小，存放在filesize环境变量里。十六进制数。

echo ${filesize}

6c8a3f

### lzmadec

lzmadec - lzma uncompress a memory region

Usage:

lzmadec srcaddr dstaddr [dstsize]

### gzwrite

gzwrite - unzip and write memory to block device

Usage:

gzwrite <interface> <dev> <addr> length [wbuf=1M [offs=0 [outsize=0]]]

wbuf is the size in bytes (hex) of write buffer

and should be padded to erase size for SSDs

offs is the output start offset in bytes (hex)

outsize is the size of the expected output (hex bytes)

and is required for files with uncompressed lengths

4 GiB or larger

### unlz4

unlz4 - lz4 uncompress a memory region

Usage:

unlz4 srcaddr dstaddr dstsize

NOTE: Specify the destination size that is sufficiently larger

than the source size.

### unzip

unzip - unzip a memory region

Usage:

unzip srcaddr dstaddr [dstsize]

# 未完成的工作

## Uboot中如果要从flash中启动版本，需要实现一个sfloadimg命令

flash中存放的版本文件，一般是放在一个个flash分区中，由于分区内文件大小不知道，所以一般需要使用mkimage工具，给文件先增加一个mkimage格式的头，这样通过先读取文件头，便可以知道文件大小，从而可以读取完整文件。

为此，我原计划增加一个sfloadimg命令，来方便的读入含有mkimage格式头的文件到内存。但由于项目进度紧急，工作被打断，且从flash中引导版本的优先级低，所以此命令一直没有实现。

实现方法很简单，就是要求先解析mkimage头（uboot里，有相应的mkimage头的处理代码，参考使用即可。），然后再读入完整文件即可。

# 结束