XOS 快速启动优化指南

1. 简介

本文档将对 XOS 系统的快速启动优化做一个系统的介绍。介绍了系统各启动阶段的优化措施及优化效果，客户可以根据实际应用，选择各个阶段的优化措施，完成 XOS 系统的启动优化。XOS 快速启动优化适用于 10XV、10XD、10XH 等芯片的启动优化。

XOS 快启优化一般包括如下几个方面：

1）启动流程优化

2）存储设备读写速度优化

2）启动 log 优化

3）Uboot 优化

4）kernel 优化

5）init 优化

6）多媒体启动优化

7）LVGL APP 启动优化

8) RTT 系统启动优化

2. 启动流程优化

启动流程优化一般是指跳过 uboot 启动，直接通过 DDRBoot 或 Ramboot 引导 OS 系统（Linux 或 RTT），该项优化一般可以优化 100ms 左右。

2.1 10XD 启动流程优化

1）优化方案

10XV 一般启动流程如下：

BootRom->RamBoot->Uboot->Kernel->Init->APP

快速启动的启动流程

BootRom->RamBoot->Kernel->Init->APP

2）具体实现方法：

待实现

2.2 10XV 启动流程优化

2.2.1 优化方案

10XV 一般启动流程如下：

BootRom->RamBoot->Uboot->Kernel->Init->APP

快速启动的启动流程

BootRom->RamBoot->DDRBoot->Kernel->Init->APP

2.2.2 具体实现方法：

首先要配置编译 DDRboot，然后在使用 mkflashimgv4 制作 xmodem.img 时快速启动文件 QM102V_linux_fast.ini 参数。

1）编译 DDRboot，生成 DDRboot.img

进入./base/soc/qm10xv/linux/board_support/ddrboot/

编译 ddrboot

```
C
make distclean
cp configs/xxxx_cfg .config
make menuconfig ,退出保存
make， 生成 debug/DDRBoot.bin
```

制作镜像 DDRBoot.img

```
C
根据需要修改 load_info.ini 中的加载的镜像文件、加载地址信息
运行 ./patch_bin.py load_info.ini debug/DDRBoot.bin 得
到 DDRBoot.img
```

2）制作可以跳过 uboot 启动的 xmodem.img

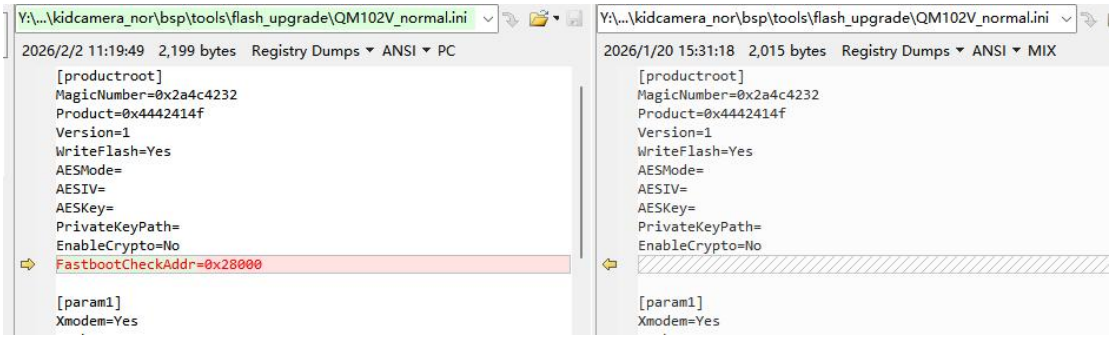
将 DDRBoot.img 复制到

./base/soc/qm10xv/linux/prebuilts/${CONFIG_XOS_PROJECT_TYPE}/bsp/tools/flash_upgrade

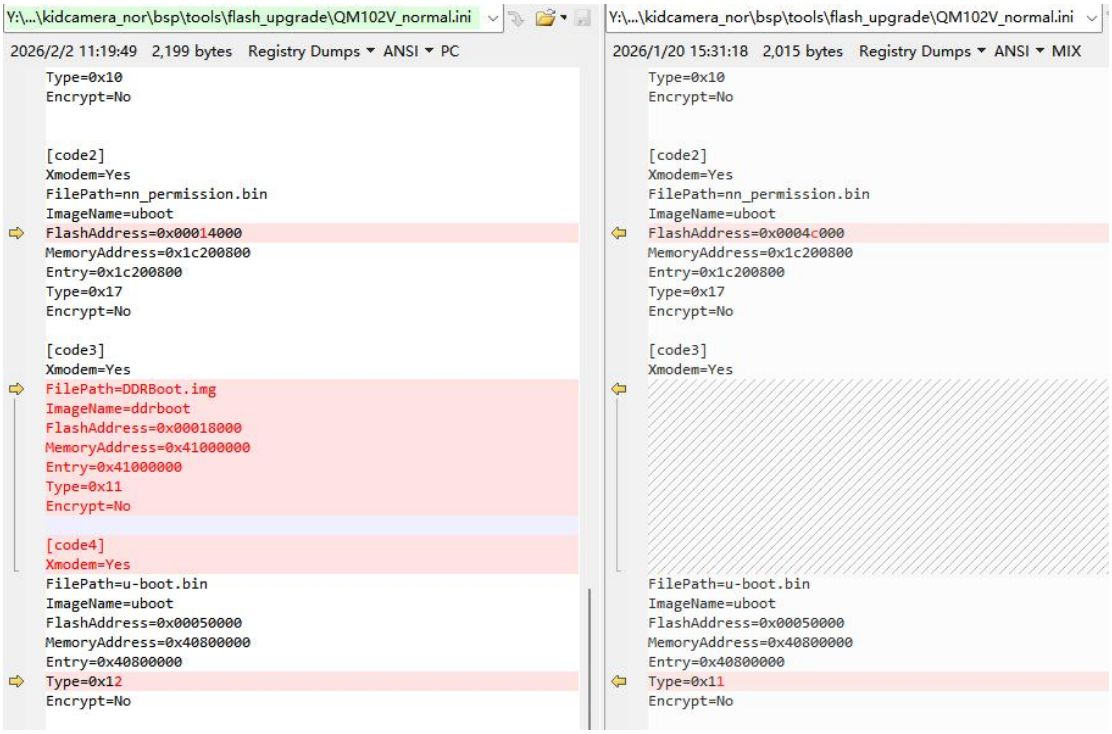
使用快启的 ini 文件 QM102V_linux_fast.ini 来制作 xmodem.img

QM102V_linux_fast.ini 与 QM102V_normal.ini 不同主要有下面两点：

增加 FastbootCheckAddr： 0x28000



增加了一个代码段 code3 来放 DDRBoot.img



进入./base/soc/qm10xv/linux/prebuilts/${CONFIG_XOS_PROJECT_TYPE}/bsp/tools/flash_upgrade

运行下面命令生成 xmodem.img

```
C
./mkflashimgv4 QM102V_linux_fast.ini
```

3)切换系统启动方式

烧好支持快启的各个镜像文件后，在快速启动应用的开发中，需要经常更新镜像进行系统调试，因此，经常需要在 fastboot 流程及 u-boot 中进行切换，其切换流程如下：

- **切换到 fastboot 的流程**

在 uboot 下运行如下命令切换到快启模式：

```
C
//在 uboot 下运行：
fastbootcmd 1; save;
```

重启板子后自动进入快启模式

- **由 fastboot 流程切换为 uboot 启动**

需要在 linux 下运行 fastbootcmd 命令，具体如下：

```
C
./fastbootcmd 0x28000 0xffffffff 0x20000
reboot
```

即可切换为 uboot 启动。

2.3 10XH 启动流程优化

待补充

3. 储存设备读取速度优化

根据客户使用的存储类型建议客户优先选择支持以下配置的存储设备：

存储类型	选择项	备注
SPI NOR	4 线，可配的最高频率、DTR	
SPI NAND	4 线，可配的最高频	
EMMC	8 线，可配的最高频率	
DDR	clk 时钟、初始化代码	

4. 启动 log 优化

XOS 系统启动过程中，一般会打印一些调试信息到调试串口，这些调试信息会占用一定的启动时间，故需要对 log 信息进行优化。

4.1 减少输出 log

XOS 系统默认的调试串口波特率为 115200bps，串口输出 10 个字符约占 1ms 时间，所以需要尽量减少启动阶段的串口 log。

4.2 提高串口 log 的波特率

有些客户，希望保留多一些必要的启动 log，为了不影响启动时间，建议提高调试串口的波特率到 1500000bps。

4.3 关闭 linux kernel 的串口 log 输出

一般是通过配置 linux kernel 的 loglevel 值，来减少 linux kernel 的串口 log 输出。如可以设置 loglevel=2，设置方法：

将 loglevel=2 添加到 uboot 的 bootargs 变量里，并保存。

```
C
uboot#set bootargs 'loglevel=2 mtdswap.partitions=10 ubi.mtd=9
ubi.mtd=12 root=/dev/ram0 rw init=/init mem=32M earlycon
console=ttyS0,115200 mtdparts=spi_nfc:128k@0(boot-
spl),128k@128k(boot-env),768k@256k(boot-
uboot),1m@1m(logo),1m@2m(padding),1m@3m(misc),3m@4m(recovery),10m@
7m(recovery-
rootfs),8m@17m(boot),20m@25m(system),6m@45m(swapfile),10m@51m(root
fs),6m@61m(data),-@67m(usrdata) lcm=1'
uboot#saveenv
```

客户也可以选择直接关闭 linux kernel ,设置方法：

将 quiet 添加到 uboot 的 bootargs 变量里，并保存。

```
C
uboot#set bootargs 'quiet mtdswap.partitions=10 ubi.mtd=9
ubi.mtd=12 root=/dev/ram0 rw init=/init mem=32M earlycon
console=ttyS0,115200 mtdparts=spi_nfc:128k@0(boot-
spl),128k@128k(boot-env),768k@256k(boot-
uboot),1m@1m(logo),1m@2m(padding),1m@3m(misc),3m@4m(recovery),10m@
7m(recovery-
rootfs),8m@17m(boot),20m@25m(system),6m@45m(swapfile),10m@51m(root
fs),6m@61m(data),-@67m(usrdata) lcm=1'
```

```
uboot#saveenv
```

5. uboot 优化

5.1 精简代码

移除所有非必要的功能，去除不用的驱动及组件，以减小 uboot image 的大小及初始化时间。如不需要网络功能，可以去除网络相关的代码；不用 SD 卡可以驱动 emmc 驱动等。

5.2 跳过刷机检测

XOS 可以配置启动时自动检测是否有 USB 或 SD 卡，然后进入自动刷机，方便开发阶段程序更新。检测 U 盘需耗时 1.5s，SD 卡检测也会耗时一定时间。快启时一定需要关闭该功能。一般可以通过设置 skipupdate 环境变量来跳过自动刷机功能。命令如下：

```
C
uboot#set skipupdate 1
uboot#setenv
```

5.3 去除 bootdelay 时间

快启模式，需要将 bootdelay 设置为 0，以减少等待时间，具体命令如下：

```
C
uboot#set  bootdelay 0
uboot#saveenv
```

5.4 按下一级 image 实际大小加载

修改 bootcmd 按 image 的实际大小，如果 bootcmd 中加载的值很大，超过下一级 image 的大小，会增加启动时间。如果内核实际只有 3MB，但 bootcmd 中加载大小设置为了 6MB，则加载时间会加倍。

6. linux 内核启动优化

6.1 裁剪内核大小

内核大小直接决定了内核 image 的加载时间，一般通过以下方法来减小内核 image 的大小：

- 精简内核配置，去掉不需要配置及功能
- 移除非必要的驱动，只保留启动阶段需要的驱动，首界面显示后，后续需要的驱动可以以 ko 形式手动加载
- 移除 dts 中不需要项
- 减少 tty 设备节点的数量

6.2 选择内核的压缩方式

XOS 支持的 kernel 压缩方式有以下几种：

Gzip、LZMA、XZ、LZO、LZ4、None（不压缩）

他们的压缩比率：

XZ ≈ LZMA > Gzip > LZO > LZ4 > None

解压速度：

None > LZ4 > LZO > Gzip > XZ > LZMA

需要权衡 flash 大小占用和启动速度。如果不考虑 flash 占用，推荐使用 None。

对 QM10XV+spi nor flash 的产品，实测也是采用非压缩方案，启动比较快。

6.3 打包到内核的 ramdisk 压缩方式选择

同上对于打包到内核里的 ramdisk 压缩方式，在不考虑 flash 占用的情况下，推荐使用 None。

6.4 内核启动 log 优化

减少内核打印 log，同样也可以减少启动时间，方法如下：

设置 loglevel 值配置 kernel 的打印等级，或设置 quiet 关闭内核 log。

6.5 kernel 启动期间加载分区优化

kernel 启动结束后，一般会加载 rootfs 分区，有的项目还会加载其他分区。kernel 启动阶段分区加载主要优化下面两点：

- 减少加载分区数量，只加载必要的分区
- 减小必需加载的分区，分区大小

加载多一个 ubi 分区，一般会 100 多毫秒的耗时。

7. linux init 阶段优化

XOS SDK Linux Init 阶段主要完成，分区加载、多媒体 ko 加载、动画加载（可选）、app 加载、wifi 加载等工具。一般分成两个阶段，第一阶段一般指从进入 linux shell 到多媒体 ko 加载完成，第二阶段一般指 ko 加载后到 app 加载完成。init 阶段的优化主要有 init 脚本优化和分区加载优化。

7.1 init 脚本优化

init 脚本优化主要有以下两点：

- 减少 init 脚本文件的数量，多一个 init 脚本文件一般会增加 20ms 延时。
- 精简 init 脚本加载的内容，只加载必要的脚本（有开机动画时，只加载开机动画相关的脚本，无开机动画时只加载与首页面相关的脚本）。
- 将非必要的脚本都放到 UI 首界面显示以后再加载。

需要特别关注 wifi 的加载时间比较长，即使放在后台运行也影响启动时间，建议通过在 wifi 加载脚本中加 sleep 来延时到到 UI 启动完成再加载 wifi。

7.2 init 阶段分区加载优化

只加载第一阶段需要的分区，并尽量减小分区的大小。

8. 多媒体启动优化

多媒体部分的优化也遵循“必要的先加载，非必要的延后加载”。

显示相关及其依赖的驱动 ko 要先加载，像 fhfb.ko、vou.ko、lcm.ko、g2d.ko、media_process.ko、sys.ko、fybase.ko、mmz.ko、fyosal.ko 等。若是要播放开机动画，则还需 vdu.ko、vppu.ko、acw.ko 等。它们都是放在 pre_loadko.sh 脚本里加载的，这个脚本要在开机动画服务程序 bootanimation_server 启动前执行，见开机启动脚本 S11bootanimation。

其他的多媒体驱动 ko，像 venc.ko、vgs.ko、jpeg.ko、vpu.ko、vicap.ko、isp.ko 等，都放在 loadko.sh 脚本里加载，这个脚本可以放在应用程序 qxosui 启动后加载，见开机脚本 S20init_rootfs。

在有开机动画的情况下，因为多媒体内存池的配置以及显示设备 vo 的使能都是在开机动画服务程序 bootanimation_server 里完成的，所以应用程序 qxosui 要与开机动画服务程序 bootanimation_server 做好同步，并尽可能同步执行，以优化启动速度。具体

的同步点有以下这些：

- 应用程序 qxosui 里打开 fb，要在开机动画服务程序 bootanimation_server 配置完多媒体内存池后才能进行；
- 应用程序 qxosui 里应等待开机动画服务程序 bootanimation_server 播放完动画，停留在最后一帧时，才显示 fb，并让开机动画服务程序 bootanimation_server 释放视频层；
- 应用程序 qxosui 里调用 vo 视频层的接口，要在开机动画服务程序 bootanimation_server 释放视频层之后才能进行。

9. LVGL APP 启动优化

9.1 资源加载优化

一个 LVGL 应用往往包含很多资源文件，而首页一般只使用到部分资源文件。加载资源越多，需要的延时也越大。为优化启动时间，可以将资源文件分两个部分，分开存放，分开加载。首页使用的资源文件在启动 app 之前加载，其他文件在启动 UI 后，后台加载。

9.2 APP 程序优化

- alignto 对齐耗时优化
- 空间字符串优化
- 为了尽快显示出第一帧 UI 界面，建立应用第一个的界面要尽量简单，比如只放一个 image 控件配一张 jpeg 的背景图（jpeg 才能硬解，快；png 只能软解，慢）。然后起个定时器，延后加载其他界面和图片。
- 切记：不要在主线程中执行耗时操作。阻塞了主线程将导致界面得不到及时绘制。可以另建线程执行耗时操作。
- QXOSUI size 优化：

10. RTT 系统优化

待补充

11. XOS 系统优化示例

11.1 10XD linux 启动优化示例

X-AIOS-10XD SDK 提供一个 10XD nand linux 快启优化的示例，可以通过一下编译命令可以编译快启的烧写包：

```
C
make distclean
make project_demo_ld_fbnand_defconfig
make xos -j10
```

该示例 image 可在 10XD EVB（102D_REF_V10 模组）直接运行，启动时间分解见下表：

启动阶段	耗时	备注
BootRom+DDRAM 阶段	0.130s	
Uboot 阶段	0.673s	logo 显示：0.300s kernel（3.6M）加载：0.273s
kernel 阶段	1.362s	压缩 kernel+ramdisk
Init shell+多媒体 ko	0.805s	多媒体 ko 加载约 0.6s
动画+app 首界面	2.060s	
合计	4.970s	

11.2 10XD RTT 启动优化示例

待补充

11.3 10XV linux 启动优化示例

待补充

11.4 10XH linux 启动优化示例

待补充

附录 1：快启优化检查项表

编号	优化检查项	优化措施	备注（收益、
一	SPL(Ramboot)		
1	精简代码	移除所有非必要的功能 去除不用的驱动及组件	减小img size，
3	尽量减少串口打印	去除非必要的串口打印	串口log，10个
4	存储设备读速度	SPINor：4线，可配的最高频率、DTR	
		SPINand：4线，可配的最高频	
		EMMC：8线，可配的最高频率	
5	是否跳过U-BOOT	调过U-BOOT，直接加载kernel	宏控，是否调过 加载kernel；
二	Uboot快启优化		

点击图片可查看完整电子表格