

快启

使用指南

文档版本 V1.0

发布日期 2025-09-18

概述

本文档主要介绍快速启动方案的基本原理、操作步骤及使用注意事项等，为软件开发人员实现产品的快速启动提供必要的指导。



说明

FLASH 器件对快启时间影响较大，本文默认以 SPI NOR FLASH 为例。

为了便于说明，本文定义了以下术语：

术语	说明
<platform>	芯片平台代号，代表某个系列的芯片，例如 xmfalcon、xmorca
<soc>	芯片名，代表某个具体的芯片，例如 xm7606v13、xm7206v11a
<toolchain>	代表工具链，例如 arm-gcc12.2.0-linux、arm-gcc12.2.0-linux-uclibceabi

关于<platform>：

<platform>	说明	对应的<soc>
xmfalcon	xm7606v1x 系列芯片平台代号	xm7606v13 xm7605v13 xm7605v12 xm7605v11 xm7506v13

<platform>	说明	对应的<soc>
xmorca	xm7206v1x 系列芯片平台代号	xm7206v11a xm7206v10b xm7206v10 xm7206v12a

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本

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

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。

修订日期	版本	修订说明
2025-09-18	V1.0	第一次正式版本发布。

目 录

1 系统启动及优化概述	1
1.1 免责声明	1
1.2 快速启动概述	1
1.3 快速启动的主要目标	1
1.4 快速启动总体参考方案	2
1.4.1 系统启动总体流程	2
1.4.1.1 CPU 启动流程	3
1.4.1.2 MCU 启动流程	3
2 MCU 快启开发指南	4
2.1 MCU 开发概述	4
2.2 MCU 镜像分段	4
2.3 MCU sample 开发	6
2.4 MCU 阶段 sensor 移植	6
2.5 MCU 调试指南	7
2.5.1 MCU DEBUG	7
2.5.2 MCU 收敛调试	8
2.5.3 注意事项	8
3 LINUX 快启开发指南	10
3.1 Uboot	10
3.2 Kernel	10
3.2.1 镜像调整	10
3.3 文件系统	11
3.3.1 Initramfs	11
3.3.2 Rootfs	12
4 Sample_Quickstart	14
4.1 概述	14
4.2 媒体软件方案	15
4.3 快启音频	15

4.4 编译运行	15
4.4.1 选择快启相关配置参数	16
4.4.2 选择 MCU 快启支持	16
4.4.3 gmp 驱动配置	16
4.4.4 配置 initramdisk 启动项	16
4.4.5 配置 rootfs 启动项	17
4.4.6 修改 bootargs 镜像参数	18
4.4.7 快启编译脚本	18
4.4.8 系统启动说明	18
5 优化指南	20
5.1 优化概述	20
5.2 图像收敛	20
5.2.1 图像收敛概述	20
5.2.2 Sensor 提前启动	20
5.2.3 AE 标定	20
5.2.4 AE、AWB 提前收敛	21
5.3 Flash 性能优化	21
5.4 量产性能优化	21
5.5 调试日志优化	21
5.5.1 Bootrom 日志关闭	21
5.5.2 Auxcode 日志关闭	22
5.5.3 Kernel 日志关闭	22
5.6 Sensor 数据传输优化	22
5.7 媒体通路优化	22
5.8 缩减分区空间	22
6 FAQ	23

插图目录

图 1-1 快速启动方案图.....	2
图 1-2 快速启动流水线图.....	2
图 2-1 MCU 快启方案图.....	4
图 2-2 MCU 媒体业务分段图	5
图 3-1 initramfs 流程.....	12
图 4-1 媒体软件方案流程图	15

表格目录

表 2-1 MCU sensor 适配表..... 7

表 2-2 MCU 7206 系列 UART 管脚复用示例 7

表 2-3 MCU 7606v1 系列 UART 管脚复用示例 8

表 4-1 启动流程各点说明 18

1 系统启动及优化概述

1.1 免责声明

本文中提供的快速启动方法仅供客户开发参考。在使用这些优化方法之前，请您务必仔细理解并分析其可能带来的风险。您可以选择不使用本文档中的快速启动优化手段，但您如果使用，您的使用行为将被视为已认可本声明，并知晓快速启动方法可能存在的风险。

1.2 快速启动概述

对于消费类 IP Camera 电池系列产品，系统启动时间的长短直接影响用户体验、电池使用时长，因此缩短系统时间是一个非常关键的技术点。

快启本文指系统启动过程中的媒体采集处理业务快速启动，它与正常模式启动的差异在系统启动中尽早初始化启动媒体采集模块，采用包括主 CPU 从 bootrom 到 OS 启动同时 MCU 执行初始化媒体采集和处理过程，达到缩短媒体业务从启动到媒体通道采集收敛和处理到出视频/图片码流时间的目的。

SDK 开发包提供了基于快速启动整体解决方案，下文会具体描述快速启动用到的优化方案，帮助客户缩短开发时间，提高产品竞争力。由于每款具体的产品规格不同，启动流程会有相应的变动，启动时间也有长短之分，因此本文介绍的快速启动方案仅供参考，请根据产品特性整体选用或部分选用。

1.3 快速启动的主要目标

快速启动的主要目标包括：

- 快速获取到第一帧可用的视频。

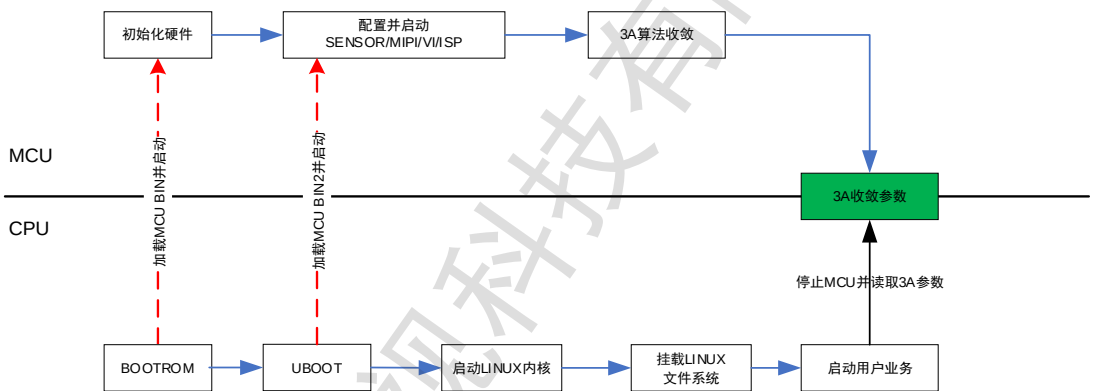
- 开机后可快速抓拍或录像，缩短启动时长。

本文所描述的快速启动主要围绕上述第一点目标进行优化，其余目标需要用户根据实际业务场景自行优化，下文会进行展开说明。

1.4 快速启动总体参考方案

在主 CPU 正常启动业务前，同时在另外的处理器上快速执行媒体采集和处理，提前进行 3A 收敛，以缩短业务正式启动过程中媒体的采集和 3A 收敛时间，达到快速输出正常图像的目的。具体方案请参考图 1-1 所示：

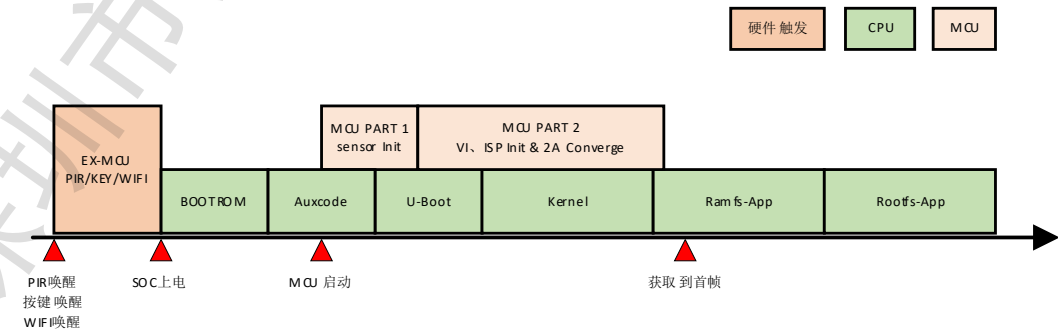
图1-1 快速启动方案图



1.4.1 系统启动总体流程

上电后主核执行 BootRom，之后主核启动 MCU 运行 MCU part 1，part2，而主核则启动 Linux 开启正常业务。如图 1-2 所示：

图1-2 快速启动流水线图



1.4.1.1 CPU 启动流程

步骤 1 Auxcode

主要包括 MCU part 1 加载、U-Boot 加载。

步骤 2 U-Boot

主要包括 MCU part 2 加载、Kernel 镜像加载与解压。

步骤 3 Kernel

主要包括 Shell 初始化、Mod drv 初始化。

步骤 4 Ramfs-App

主要包括媒体通路创建、编码数据缓存。

步骤 5 Rootfs-App

主要包括编码数据读取。

----结束

1.4.1.2 MCU 启动流程

步骤 1 Part 1

主要包括 mipi 初始化、sensor 序列、硬光敏、红外灯、IR_CUT 初始化。

步骤 2 Part 2

主要包括 vi、isp 初始化及收敛结果保存。

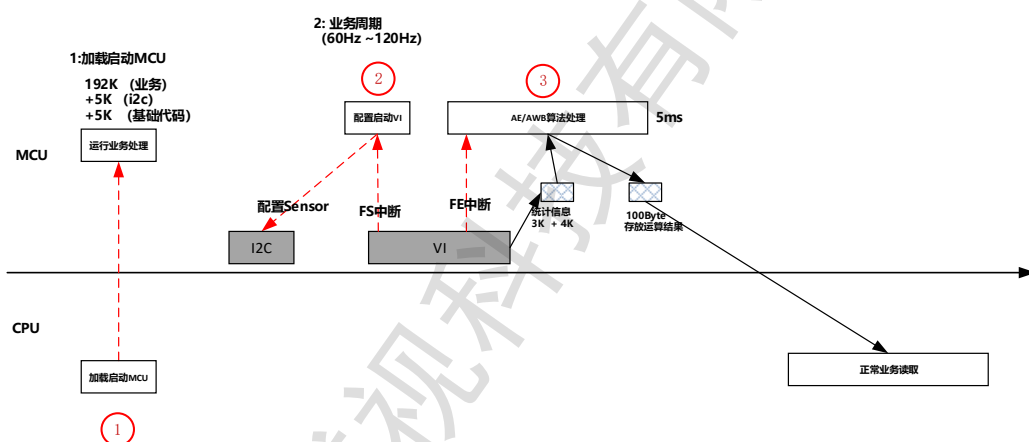
----结束

2 MCU 快启开发指南

2.1 MCU 开发概述

在快启方案中，MCU（Micro Controller Unit）实现了 SoC 系统中的 sensor 初始化，AE/AWB 快速启动，收敛，如图 2-1 所示：

图2-1 MCU 快启方案图



其中：

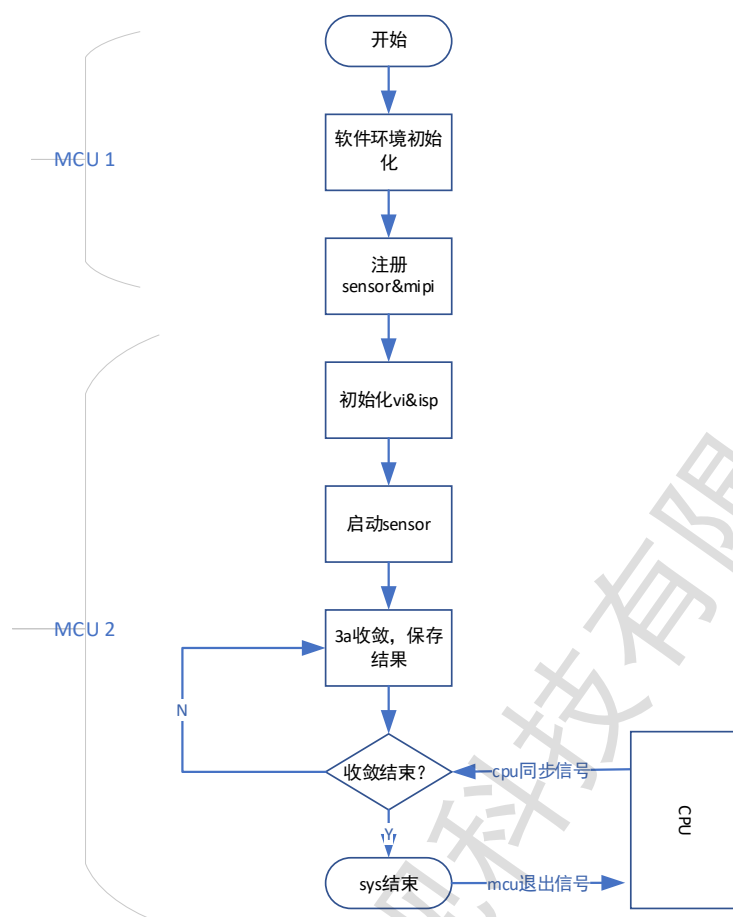
- ① 主核将 AE/AWB Firmware 加载到 MCU，开启 MCU 运行；
- ② MCU 配置初始化 sensor, mipi，通过 VI 接收曝光图像数据；
- ③ 通过 AE/AWB 算法处理曝光图像数据，产生 AE/AWB 参数，完成业务后等 CPU 处理。

2.2 MCU 镜像分段

为兼顾快启整体耗时及 MCU 收敛帧数，MCU 镜像采用分段的方式加载，本章节针对分段加载进行详细说明，请根据实际需求选择。

MCU 镜像分为两段加载，Auxcode 阶段加载 Part 1, U-Boot 阶段加载 Part2。各段媒体业务功能如图 2-2：

图2-2 MCU 媒体业务分段图



- MCU Part1

Part1 由 Auxcode 阶段加载，FLASH 处于 boot 模式，加载速度~8.25M/s。为增加 MCU 曝光帧数，且 Part1 由于加载速度较慢，不宜将所有 MCU 镜像在此处加载，以免影响快启时间，故将 sensor 序列初始化和 mipi 初始化放到此段执行，以提前初始化 sensor。

此段主要包括 mipi 初始化、sensor 序列、硬光敏、红外灯、IR_CUT 初始化。

- MCU Part2

Part2 由 U-Boot 阶段加载，FLASH 处于 normal 模式，加载速度~60M/s。

此段主要包括 vi、isp 初始化及 AE/AWB 收敛结果保存。

说明

- 上述描述的 AE/AWB 收敛只针对芯片本身的 isp 算法收敛情况，不含 sensor 自行收敛的情况。
- 上述描述的 MCU 分段只适用于 nor flash，不适用于 nand，emmc 闪存。

2.3 MCU sample 开发

MCU 的系统方案主要流程集中在 sdk/mcu/ riscv/。

MCU 的媒体方案主要流程集中在 sdk/mcu/media/。

MCU 的 sample 主要集中在 sdk/mcu/media/sample/media_quickstart.c。

2.4 MCU 阶段 sensor 移植

Mcu 的 sensor 驱动是通过注册的方式供 framework 使用。

MCU 阶段 sensor 初始化因方案设计放在 part1 中，故此处需要指定相应函数，变量等为 Part 1 段用。其中相关定义请参考 sdk/mcu/media/include/sns_comm.h

```
#ifndef __linux__ //linux平台
#define SENSOR_PRIORITY_FUNC
#define SENSOR_PRIORITY_DATA
#define SENSOR_PRIORITY_RODATA
#else //mcu平台
#define SENSOR_PRIORITY_FUNC STAGE1_FUNC
#define SENSOR_PRIORITY_DATA STAGE1_GLOBAL
#define SENSOR_PRIORITY_RODATA STAGE1_CONST_GLOBAL
#endif
```

- SENSOR_PRIORITY_FUNC: 代码段(.text)，用来存放程序执行代码的一块内存区域，主要修饰函数。
- SENSOR_PRIORITY_DATA: 数据段(.data)，通常是指用来存放程序中已初始化的全局变量的一块内存区域，数据段属于静态内存分配。
- SENSOR_PRIORITY_RODATA: 只读数据段(.rodata)，程序运行时不可修改。

具体参考 sdk\mcu\media\sensors\smartsens_sc485slsc485sl.c, sc485_ctrl.c:

1. sensor 属性集只读，SENSOR_PRIORITY_RODATA 修饰：

```
SENSOR_PRIORITY_RODATA g_sc485sl_property;
```

2. sensor 注册时用的全局变量，SENSOR_PRIORITY_DATA 修饰：

```
SENSOR_PRIORITY_DATA g_sc485sl_dev_map
SENSOR_PRIORITY_DATA g_sc485sl_ctx
SENSOR_PRIORITY_DATA g_sc485sl_i2c_addr
SENSOR_PRIORITY_DATA g_sc485sl_i2c_fd
```

3. sensor 注册函数，SENSOR_PRIORITY_FUNC 修饰：

包括 sc485sl_register 调用流程的所有函数：

```
SENSOR_PRIORITY_FUNC sc485sl_search_dev
SENSOR_PRIORITY_FUNC sc485sl_ctx_init
SENSOR_PRIORITY_FUNC sc485sl_init_register_func
SENSOR_PRIORITY_FUNC xmedia_isp_register_sensor
SENSOR_PRIORITY_FUNC sc485sl_master_init_reg_info
SENSOR_PRIORITY_FUNC sc485sl_set_mipi_lanes
.....
```

说明

- 因 mcu 资源受限，调试比较困难，可先在 linux 下调通相应 sensor 驱动。保证出图和图像正常。在按以上步骤移植到 mcu，linux 下 sensor 调试请参考<< Sensor 调试指南>>。
- 目前 mcu 已适配的 sensor 如表 2-1 所示：

表2-1 MCU sensor 适配表

Sensor	Type(BIT/FPS/WDR)	Lan	Width	Height	Status
sc485sl	4M_30FPS_12BIT	4L	2688	1520	√
	4M_30FPS_12BIT	2L	2560	1440	√
	4M_60FPS_12BIT	4L	2560	1440	√
sc235hai	2M_30FPS_10BIT	2L	1920	1080	√
	2M_60FPS_10BIT	2L	1920	1080	√

2.5 MCU 调试指南

2.5.1 MCU DEBUG

Boot 表格配置 MCU UART pinmux，打开 MCU 串口输出。

eg1：选择对应表格文件 sdk/configs/xm7206v11a/reg/

XM7206V11A_RB_6L_DDR3_2133M_128M_1x16bit.xlsm，pinout 表格中修改管脚复用，如表 2-2 所示：

表2-2 MCU 7206 系列 UART 管脚复用示例

iocfg_reg21	0x12090004	0x1002	0	write	31
iocfg_reg22	0x12090008	0x1002	0	write	31

eg2: 选择对应表格文件 sdk/configs/ xm7506v13 /reg/
XM7506V13_EVB_6L_DDR4_2666M_2GB_2x16bit.xlsm, pinout 表格中修改管脚复
用, 如表 2-3 所示:

表2-3 MCU 7606v1 系列 UART 管脚复用示例

iocfg_reg113	0x112C00F0	0x1502	0	write	31
iocfg_reg18	0x119800E0	0x1502	0	write	31

具体表格修改方法可参考<<U-boot 移植应用开发指南>>中 4.1 章节。



此处会因硬件电路设计产生差异, 请根据具体情况修改。

2.5.2 MCU 收敛调试

AE/AWB 收敛结果保存在如下结构体中, 可根据需求在 drv_isp_process()函数中打印
相关参数, 以便于图像调试。

```
typedef struct {
    xmedia_bool ir_mode;
    xmedia_u32  exp_time;
    xmedia_u32  again;
    xmedia_u32  dgain;
    xmedia_u32  ispdgain;
    xmedia_u32  exposure;
    xmedia_u32  init_iso;
    xmedia_u32  piris_fno;
    xmedia_u16  wb_rgain;
    xmedia_u16  wb_ggain;
    xmedia_u16  wb_bgain;
    xmedia_u16  sample_rgain;
    xmedia_u16  sample_bgain;
    xmedia_u16  ccm[XMEDIA_SENSOR_CCM_MATRIX_SIZE];
} xmedia_sensor_init_param;
```

其中重点关注如下参数:

- AE 快启输出参数: exposure、exp_time、again、dgain、ispgain。
- AWB 输出参数: wb_rgain、wb_ggain、wb_bgain, ccm。

2.5.3 注意事项

- sensor 驱动大小略有差异, 使用分段加载时, 请根据实际 stage1 实际大小配置 PART1 size。其中 stage1 基地址 0x0402 0000

eg: sdk/mcu/riscv/configs/xmorca_defconfig

```
CONFIG_STAGE2_START = 0x402A000    40KB    //40k
```

```
CONFIG_STAGE2_START = 0x4028000    32KB    //32k
```

- 若 MCU 出现异常打印，请排查分段 1 sensor 相关接口是否被 SENSOR_PRIORITY_FUNC 所修饰。参考 [2.4 章节](#)。
- 其他注意配置选项

```
CONFIG_PARAM_MEM_SIZE = 4096        // AE/AWB参数分区大小
```

```
CONFIG_SERIAL_DISABLE = y           // mcu 串口软控开关
```

```
CONFIG_DEBUG_INFO = y               // 调试信息输出
```

```
CONFIG_MEDIA_SUPPORT = y            //快启媒体驱动编译开关
```

```
CONFIG_MEDIA_SAMPLE_QUICKSTART = y  //快启sample编译开关
```

```
CONFIG_STOPWATCH_SUPPORT = y        //mcu侧sensor初始化，收敛耗时统计开关
```


3 LINUX 快启开发指南

3.1 Uboot

U-boot 在快启方案中与普通方案区别需要加载 MCU part2 段固件。另外一些 bootrom 或者辅助代码区的配置也放在了 uboot 的配置文件中。

U-boot 启动时间主要由驱动初始化时间、所加载的 kernel 镜像大小、镜像加载速率等因素影响。快速启动主要对以上几点进行优化。

快启大部分手段均已在 U-boot 版本中实现，详情请参考 `sdk/source/bootloader/u-boot-2020.01/configs/<platform>_quickstart_defconfig`。

3.2 Kernel

Kernel 主要进行 sdk mod 初始化、外设初始化、媒体业务初始化、文件系统初始化。

Kernel 的启动时间主要由内置驱动的大小、内置驱动的初始化时间等因素影响，所以针对的内核的裁剪至关重要，此处针对内核优化如下：

- 减小内核体积，通过 kconfig 精简驱动与子系统，可参考相应配置文件 `sdk/source/kernel/linux-5.10.y/arch/arm/configs/ <platform>_quickstart_defconfig`
- 加快业务第一级 APP 启动，将慢速且开机业务不依赖的模块做成 KO 模块，放到后期文件系统挂载后加载，如 USB、SD 卡驱动、网口驱动等；

3.2.1 镜像调整

kernel 镜像大小对 uboot 阶段内核拷贝的耗时强相关，参考 sdk 默认内核分区为 6M，但实际内核经过裁剪处理后大概在 3~4M 左右，对时间和 flash 内存都有一定的影响。本节介绍 kernel 分区的大小调整。

内核编译结束之后根据生成的 kernel 文件大小修改 `sdk/configs/<soc>/`

`prebuilts/spi_bootargs_quickstart.txt`，并重新编译 bootargs。具体修改规则如下，

以 kernel 文件大小为 3825KB 为例：

`bootargs= ... 512K(bootargs),4M (kernel) ,...` 将 kernel 大小修改为大于等于当前镜像大小的 64KB 最小倍数，因此 kernel 大小修改为 3840K($3825 < 60 \times 64$)。

bootargs 修改：

```
bootargs= ... 512K(bootargs),3840K (kernel) ,....
```

bootcmd 修改:

```
bootcmd=... boothz 40FFFC0 0x40208000 0x100000 0x3C0000(3840*1024)
```

Kernel 烧录镜像大小修改:

根据生成的 Kernel 镜像大小, 修改 xml 文件 spi_partitions_quickstart.xml, 修改 kernel 对应分区大小为 3840K。

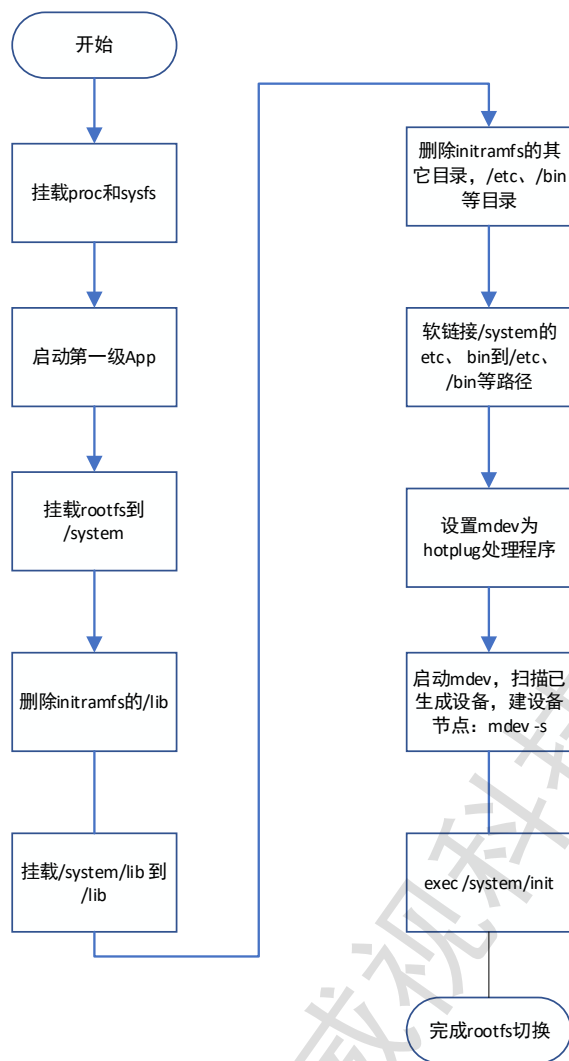
3.3 文件系统

3.3.1 Initramfs

对于快速启动的场景, 考虑到 Flash 读性能远低于 DDR 的读性能, 为了加快业务 APP 的启动, 基于 linux 的 Initramfs 机制, 仅将启动必要的功能 buildin 到内核中并随内核镜像一起压缩, boot 加载内核时一并解压, 然后由 linux 做展开并挂载。

Initramfs 需要完成媒体 APP 加载, 然后切换到 flash 上的 rootfs。其工作流程如图 3-1 所示:

图3-1 initramfs 流程



为了尽可能减少 flash 占用和 boot 加载的时间，initramfs 做如下优化：

- 放到 initramfs 的 APP 采用静态链接，最终的 initramfs 不需要放相应的动态库。
- Initramfs 中的 init 进程采用 toybox，toybox 是与 busybox 类似的 linux 命令工具集，相比 busybox 能其功能更为精简，更节省 flash 空间，适合不需要过多业务功能的 init 阶段。
- 去除 udev/mdev 组件，减少创建设备的时间：对于固定设备号的设备节点采用静态创建，即在编译构建阶段 tools/linux/utils/bin/fakeroot-scripts-initramdisk 中创建；对于动态分配设备号的设备节点，直接从/sys 中读取主次设备号调用 mknod 创建。具体参考[章节 4.4.4](#)。

3.3.2 Rootfs

toybox 的命令数只有 busybox 的一半左右，考虑到兼容性和功能完备性，因此 flash 上的 rootfs 默认采用 busybox：

1. 设备节点管理采用 busybox 的 mdev。
2. 快启场景使用 mdprobe 方式加载外设驱动或自行手动加载。

仅限深圳市安远威视科技有限公司使用

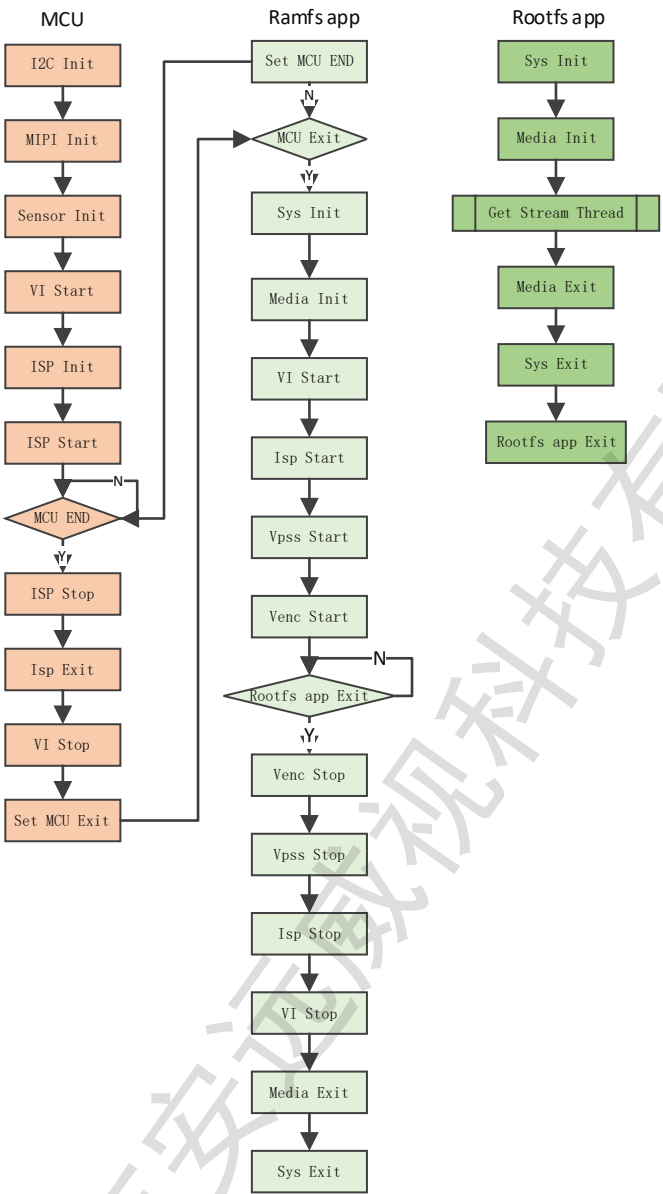
4 Sample_Quickstart

4.1 概述

sdk/mcu/media/media_sample/media_quickstart.c 提供 MCU 快启方案示例。
sdk/sample/quickstart/目录下提供 linux 侧方案示例。

4.2 媒体软件方案

图4-1 媒体软件方案流程图



4.3 快启音频

略。

4.4 编译运行

本小节主要介绍快启与普通版本之间的编译区别。

4.4.1 选择快启相关配置参数

sdk 根目录下执行以下操作，选择快启默认参数。

```
cp configs/ <soc>/ <soc>..._linux-5.10_quickstart_cfg.mk ./cfg.mk
```

4.4.2 选择 MCU 快启支持

步骤 1 sdk/mcu/riscv/configs/<platform>_defconfig 选择快启支持

```
CONFIG_MEDIA_SAMPLE_QUICKSTART = y
```

步骤 2 sdk/mcu/media/sensors/Makefile.riscv 选择对应 sensor 支持

```
SRCS-y += smartsens_sc485sl/
```

步骤 3 sdk/mcu/media/include/ media_sample.h 选择对应 sensor 宏或者序列

```
#define SENSOR_SC485SL_4LANE_60FPS_SUPPORT
```

4.4.3 gmp 驱动配置

用户可通过修改以下文件，筛选掉当前快启应用不涉及的媒体驱动。

sdk/source/gmp/drv/init/xmedia_drv_init.c

eg:

```
#if 0
    vo_driver_init();
    gfbg_driver_init();
    acomm_driver_init();
    aiao_driver_init();
    ao_driver_init();
    ai_driver_init();
    aenc_driver_init();
    adec_driver_init();
#endif
```

4.4.4 配置 initramdisk 启动项

步骤 1 编译结束之后，进行以下操作修改 initramdisk 字符设备默认挂载参数。

- a. 编译结束，烧写镜像，进入系统。
- b. `ls -l /sys/dev/char`
- c. 根据 sample_quickstart 所需字符设备修改添加 sdk/tools/utls/bin/fakeroot-scripts-initramdisk

eg: **字符设备可能略有差异，请按实际需求进行配置。**

```

mknod dev/mmez_userdev c 10 63
mknod dev/mem c 1 1
mknod dev/sys c 218 8
mknod dev/vb c 218 9
mknod dev/mipi_rx c 218 63
mknod dev/vi c 218 62
mknod dev/viproc c 218 61
mknod dev/isp c 218 20
mknod dev/vpu c 218 32
mknod dev/gdc c 218 27
mknod dev/vgs c 218 19
mknod dev/ive c 218 17
mknod dev/rgn c 218 16
mknod dev/venc c 218 2
mknod dev/log c 218 12
mknod dev/vpss c 218 10
mknod dev/venc c 218 2
mknod dev/npu.0 c 218 48
mknod dev/ddr_ost_qos c 218 49
mknod dev/i2c-5 c 89 0

```

步骤 2 修改 initramdisk.tgz，完成对 sample_quickstart 启动参数配置。

- a. 解压 sdk/source/initramdisk/scripts/initramdisk.tgz。
- b. 根据 sample_quickstart 具体名称，修改解压目录下 initramdisk/init 启动参数，配置对应启动文件。
eg: test -f /root/sample_quickstart && /root/sample_quickstart &
- c. 将编译完成的 sample_quickstart 拷贝到解压目录下/initramdisk/root/，重新压缩打包。

步骤 3 sdk 目录下执行以下命令对 kernel 进行重新打包，完成编译。

- a. make initrd_clean
- b. make linux

4.4.5 配置 rootfs 启动项

本小节主要描述双进程快启方案 rootfs 配置方法，非双进程方案不涉及。

步骤 1 编译完成后添加 rootfs 进程启动项

- a. sdk/out/<soc>/rootfs/etc/init.d/目录下创建 S02quickstart 自启动文件
- b. 根据 rootfs sample_quickstart_rootfs 名称添加启动信息

eg:

```
#!/bin/sh
echo -n "1" >/proc/stopwatch #打印进程二启动时间，可不添加
test -f /usr/bin/sample_quickstart_rootfs && /usr/bin/sample_quickstart_rootfs &
cat /proc/stopwatch #输出启动时间戳，可不添加
```

步骤 2 拷贝进程二 sample 到 rootfs 分区

将 sample_quickstart_rootfs 拷贝至 sdk/out/<soc>/rootfs/usr/bin/

步骤 3 重新打包 rootfs

- a. make jffs2_clean
- b. make jffs2

4.4.6 修改 bootargs 镜像参数

本文 3.2.1 章节已提供如何根据 kernel 镜像大小，修改 bootargs 及 bootcmd 相关参数，此处不再赘述。修改完成之后，需要进入 sdk 根目录执行如下操作，完成 bootargs 打包：

make bootargs

4.4.7 快启编译脚本

为方便客户开发，针对快启做了相应的编译脚本和说明，参考路径

sdk/sample/quickstart/quickstart_build，但脚本受硬件型号和软件配置影响，请熟悉整个快启编译后，修改部分编译选项酌情使用。

4.4.8 系统启动说明

SDK 快启用例默认关闭了大部分的打印，只保留了应用启动的一些打印和 stopwatch 时间点，其中各关键时间节点对应描述如表 4-1 所示。

表4-1 启动流程各点说明

计时点	描述	阶段
0	cpu 初始完;	bootrom
1	flash 初始完;	bootrom
2	boot 头区读&校验完成;	bootrom
3	auxcode 区读&校验完成; 开始执行 auxcode;	bootrom

计时点	描述	阶段
4	boot 表格配置完成;	auxcode
5	ddr training 完成	auxcode
6	MCU 固件加载完成	auxcode
7	auxcode 执行完成	bootrom
8	boot 代码区读并校验完成	bootrom
9	boot 初始化并完成内核加载, 开始跳转内核;	boot
10	内核初始化完成, 启动 app	linux

5.1 优化概述

本章节结合 Sample 已实现流程对快速启动和图像优化进行说明。

1. 图像收敛。
2. flash 性能优化。
3. 量产性能优化。
4. 调试日志优化。
5. Sensor 数据传输优化。
6. 媒体通路优化
7. 缩减分区空间

5.2 图像收敛

5.2.1 图像收敛概述

通过以下措施对起始图像进行收敛优化：

1. MCU 中提前启动 sensor
2. AE 标定
3. AE、AWB 提前收敛

5.2.2 Sensor 提前启动

在快启中，MCU 采用分段加载提前初始化 sensor 序列，增加 MCU 收敛帧数。

5.2.3 AE 标定

通过硬光敏标定 AE，进一步提升 MCU 收敛效果。

5.2.4 AE、AWB 提前收敛

在快启中，使用 MCU 提前收敛 AE、AWB，媒体业务启来后继承收敛结果。

5.3 Flash 性能优化

快启在冷启动过程中，从 flash 中读取 uboot、auxcode、ddr param，提高 flash 的频率能够有效缩短工作耗时。请客户根据 flash 型号选择 flash 频率，修改方式如下：

可参考<OTP API 参考>实现。（仅支持操作一次，操作后无法还原，请谨慎操作。）

- xmedia_otp_init
- xmedia_otp_program_enable
- xmedia_otp_fmc_freq_sel
- xmedia_otp_exit

5.4 量产性能优化

用户可关闭串口烧录功能，优化耗时 50ms。修改方式如下：

可参考<OTP API 参考>实现。（仅支持操作一次，操作后无法还原，请谨慎操作。）

- xmedia_otp_init
- xmedia_otp_program_enable
- xmedia_otp_uart_burn_mode_sel
- xmedia_otp_exit

5.5 调试日志优化

5.5.1 Bootrom 日志关闭

可参考<OTP API 参考>实现。（仅支持操作一次，操作后无法还原，请谨慎操作。）

- xmedia_otp_init
- xmedia_otp_program_enable
- xmedia_otp_bootrom_debug_disable
- xmedia_otp_exit

5.5.2 Auxcode 日志关闭

用户可通过以下操作，关闭 Auxcode 阶段的调试日志：

```
sdk/source/bootloader/u-boot-2020.01/configs/<soc>_quickstart_defconfig  
CONFIG_AUXCODE_LOG_CTRL=0
```

5.5.3 Kernel 日志关闭

用户可通过修改 bootargs loglevel=0，来关闭 Kernel 日志。

5.6 Sensor 数据传输优化

Sensor 配置为高速率低帧率（60fps 序列跑 30fps），对比低速率相同帧率（30fps 序列跑 30fps），能够有效优化单帧传输时间。

5.7 媒体通路优化

用户创建媒体通路时，需尽量使用媒体通路在线或低延时，降低媒体通路耗时。

5.8 缩减分区空间

SDK 发布包默认为各个分区预留了一些余量，在实际使用中可能会有相应的出入，此时用户可以根据自身定制镜像大小来调整分区大小，注意，需要考虑 Flash 块大小对齐。

具体可参考<<U-boot 移植应用开发指南 >>中 2.5.2 章节。

6

FAQ

仅限深圳市安远威视科技有限公司使用