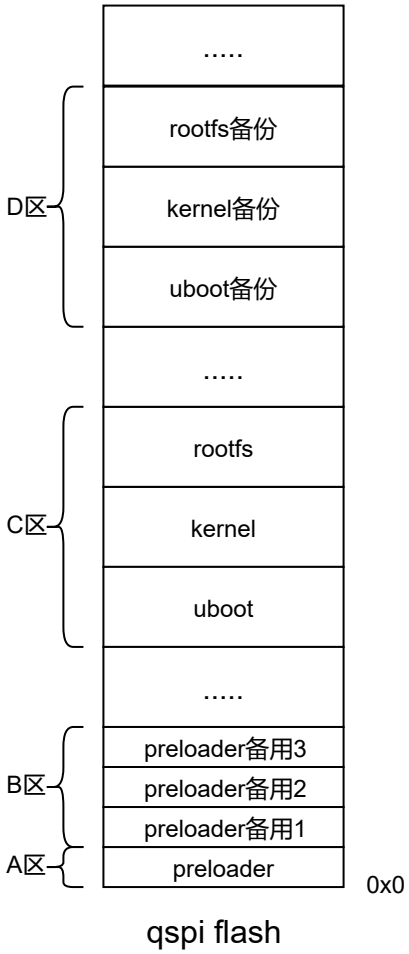
双boot启动设计

一、双boot启动概述

因为flash有可能会出现某些区域坏死的现象，因此为了防止镜像存放的区域坏死，要在flash中再开辟存放备份镜像的区域，如果正常启动是由于镜像所在的flash区域损坏，则通过备份镜像启动。



上图所示是各部分镜像在qspi中的大致分布，并不是准确的分布，比如C区和D区还有dtb没有画出来，镜像之间不一定紧挨着，还可能有空隙，这些都可以通过代码指定。

二、启动流程

1. 首先开机启动的是固化在芯片rom中的启动代码，然后rom启动代码将A区preloader加载到sram中，然后根据preloader镜像头部信息验证镜像是否可启动，如果不可启动则加载B区备用镜像，直到可以启动或者退出。**rom阶段的代码是芯片厂商固化好的，因此这部分流程我们无法更改，preloader和其备用也需要按照芯片手册烧写到固定的位置。**

2. preloader主要负责初始化ddr，最后将uboot从C区中加载到ddr，然后通过镜像头部验证uboot镜像完整性，如果镜像不可启动，说明是C区中的uboot损坏，然后从D区加载uboot备份到ddr启动。

3. uboot加载C区kernel到ddr中，然后通过镜像头部验证kernel镜像完整性，如果镜像不可启动，说明是C区中的uboot损坏，然后从D区加载uboot备份到ddr启动。**无论是从C区加载的uboot还是D区加载的uboot都是先加载启动C区的kernel。**

4. linux启动后在挂载根文件系统前验证C区rootfs区域是否损坏，如果没有问题则挂载C区的rootfs，否则验证D区然后挂载D区的rootfs。

三、启动流程代码分析和补丁

1. preloader启动uboot

```
void board_init_r(gd_t *dummy1, ulong dummy2)
{
    ...
    /* 从flash中加载uboot到ddr */
    if (boot_from_devices(&spl_image, spl_boot_list,
                ARRAY_SIZE(spl_boot_list))) {
        puts(SPL_TPL_PROMPT "failed to boot from all boot devices\n");
        hang();
    }
    ...
    /* 填充spl_image余下的主要内容 */
    spl_board_prepare_for_boot();
    /* 直接跳转到uboot入口地址 */
    jump_to_image_no_args(&spl_image);
}
```

```
static int boot_from_devices(struct spl_image_info *spl_image,
                u32 spl_boot_list[], int count)
{
    int i;

    for (i = 0; i < count && spl_boot_list[i] != BOOT_DEVICE_NONE; i++) {
        struct spl_image_loader *loader;

        loader = spl_ll_find_loader(spl_boot_list[i]);
        ...
        /* 从flash加载uboot到ddr,同时可能根据配置校验 */
        if (loader && !spl_load_image(spl_image, loader)) {
            spl_image->boot_device = spl_boot_list[i];
            return 0;
        }
    }

    return -ENODEV;
}
```

```
static int spl_load_image(struct spl_image_info *spl_image,
              struct spl_image_loader *loader)
```

```
{
    int ret;
    struct spl_boot_device bootdev;

    bootdev.boot_device = loader->boot_device;
    bootdev.boot_device_name = NULL;
    /* 加载镜像到ddr，同时将头部读取到spl_image中 */
    ret = loader->load_image(spl_image, &bootdev);
    /* 如果开启了crc校验配置则进行校验 */
#ifdef CONFIG_SPL_LEGACY_IMAGE_CRC_CHECK
    if (!ret && spl_image->dcrc_length) {
        /* check data crc */
        ulong dcrc = crc32_wd(0, (unsigned char *)spl_image->dcrc_data,
                      spl_image->dcrc_length, CHUNKSZ_CRC32);
        if (dcrc != spl_image->dcrc) {
            puts("SPL: Image data CRC check failed!\n");
---         ret = -EINVAL;
+++         ret = spl_uboot_secondry_boot();
        }
    }
#endif
    return ret;
}
```

在双boot启动中打开`CONFIG_SPL_LEGACY_IMAGE_CRC_CHECK`配置，同时打入补丁，如上所示，当校验不通过时则通过`spl_uboot_secondry_boot()`尝试加载启动备份uboot镜像，而不是直接返回错误码。

```
static int spl_uboot_secondry_boot()
{
    /* 将备份uboot从qspi D区加载到ddr */
    secondry_boot_spl_spi_load_image();
    /* 通过crc校验后返回0 */
    ret = secondry_boot_crc_check();
    return ret;
}
```

`spl_uboot_secondry_boot`设计大体如上所示，先从qspi备份区D区加载备份uboot镜像到ddr，然后校验后返回(或者挂起)，具体设计要再从长计议，看是否查考老版spl或者按照新的spl设计等等。

2. uboot启动kernel

```
bootcmd=mw.b 0x100 0xff 0x700000;tftp 0x8000 zImage;tftp 0x100
socfpga_cyclone5_socdk.dtb;run qspiboot
...
fdtaddr=0x00000100
...
loadaddr=0x8000
...
qspiboot=setenv bootargs console=ttyS0,115200 root=${qspiroot} rw
rootfstype=${qspirootfstype};bootz ${loadaddr} - ${fdtaddr}
...
qspiroot=/dev/mtdblock1
...
qspirootfstype=jffs2
...
```

bootcmd是uboot最后启动linux的环境变量，首先从ddr地址0x100开始写0x700000字节的0xff，然后通过tftp服务将内核镜像zImage和设备树socfpga_cyclone5_socdk.dtb从主机分别下载到ddr的0x8000和0x100地址，最后通过run qspiboot命令启动内核。qspiboot首先设置通过setenv bootargs设置内核启动参数，通过源码分析这里其实修改了设备树中的内核参数属性，最后通过bootz启动之间下载到指定地址的内核。由此可见bootz是uboot最后一条命令，我们重点关注bootz做了什么。其中bootz命令会首先调用do_bootz函数，**之后就从do_bootz函数修改启动流程使得支持双boot启动。**