程序项目代做，有需求私信（vue、React、Java、爬虫、电路板设计、嵌入式linux等）

大奥特曼打小怪兽

随笔 - 349 文章 - 0 评论 - 304 阅读 - 130万

Mini2440之uboot移植流程之linux内核启动分析（六）

目录

公告

公告&打赏

创作不易，喜欢的话，请考虑支持一下

昵称： 大奥特曼打小怪兽
园龄： 6年6个月
粉丝： 891
关注： 12
+加关注

积分与排名

积分 - 524277
排名 - 1232

随笔分类 (492)

bigdata(4)
deep learning(38)
H3(1)
java(14)
javascript(1)
linux alsa(15)
linux blk(7)
linux debug(1)
linux drm(13)
linux dts(13)
linux embedded environment(6)
linux gpio(6)
linux gpu(2)
linux gui(2)
Linux i2c(7)
linux interrupt(5)
linux network(5)
linux ota(4)
linux rootfs(15)
linux shell(2)
更多

在前面的章节关于u-boot的源码，以及u-boot的移植这一块我们介绍完了。接下来，我们应该开始进入第二个阶段，linux内核移植，以及驱动开发。

但是在这之前，我们遗漏了u-boot中的一个重要环节没有介绍，就是u-boot如何执行bootm命令，如何实现linux内核启动。

我们在Mini440之uboot移植之源码分析命令解析（五） 介绍过如果配置了CONFIG_BOOTCOMMAND宏：

```
#define CONFIG_BOOTCOMMAND "nand read 0x30000000 kernel; bootm 0x30000000" //bootcmd
```

那么在执行autoboot_command函数的时候，将会执行bootcmd中保存的命令。

- nand read 0x30000000 kernel：这里将Nand Flash kernel分区的内核镜像加载到地址0x30000000；
- bootm 0x30000000：启动linux内核；bootm这个命令用于启动一个内核镜像，这个镜像就是uImage文件，它会从uImage镜像文件的头部取得一些信息，这些信息包括：CPU架构、操作系统类型、文件类型、压缩方式、加载地址、运行的入口地址等；

回到顶部

一、内核镜像

前面我们说了u-boot启动linux内核，从Nand Flash内核分区中加载uImage镜像，那么什么是uImage镜像呢？说到这个我们就不得不聊一聊内核镜像的几种文件格式了。

1.1 介绍

linux内核编译之后一般会生成一下两个文件，一个是Image，一个是zImage，其中Image为内核镜像文件（可以直接在芯片上运行原生二进制文件），而zImage为内核的镜像压缩文件（但它不仅爱是一个压缩文件，在文件的开头部分内嵌有gzip解压缩代码）。

但是这两种镜像的格式并没有办法提供给u-boot的足够的信息来进行load、jump或者验证操作等等。因此，u-boot提供了mkimage工具，来将zImage制作为u-boot可以识别的格式，将生成的文件称之uImage。

需要注意的是：这里并不是说u-boot一定不支持zImage镜像文件的启动，一般可以通过配置CONFIG_ZIMAGE_BOOT使u-boot支持zImage启动。

1.1.1 Legacy uImage

Legacy uImage它是在zImage之前加上了一个长度为64字节的头，说明这个内核的CPU架构、操作系统类型、文件类型、压缩方式、加载地址、运行的入口地址等。

1.1.2 FIT uImage

FIT uImage是在Legacy uImage的基础上，为了满足Linux Flattened Device Tree（FDT）的标准，而重新改进和定义出来的一种镜像文件格式；它一般将kernel、fdt、ramdisk等等镜像打包到一个itb镜像文件中；u-boot只要获得了这个镜像文件，就可以得到kernel、fdt、ramdisk等等镜像的具体信息和内容。

1.1.3 Image、zImage、uImage发展历程

为什么会出现 Legacy uImage ，然后又出现 FIT uImage ?

最开始出现的是Image，就一个普通的内核镜像。然后为了节省空间，有了 zImage，进行了压缩可以节省空间。

u-boot启动一个Image或者 zImage，还必须要给它传递一些参数；

- 镜像文件的类型，如kernel image、dtb文件、ramdisk image等等?
- 镜像文件需要放在内存的的哪个位置（加载地址）？
- 镜像文件需要从内存哪个位置开始执行（入口地址）？
- 镜像文件是否有压缩?
- 镜像文件是否有一些完整性校验的信息（如CRC）？

这种方式的不足就在于，镜像本身没有带有这些参数的，用工具制作完镜像后，还需要另外再向u-boot提供这些参数，才能正常启动（就是比较麻烦）。

随笔档案 (349)

2024年8月(1)
2024年7月(7)
2024年6月(6)
2024年4月(1)
2024年3月(4)
2024年2月(5)
2024年1月(2)
2023年12月(4)
2023年11月(9)
2023年10月(4)
2023年9月(10)
2023年8月(1)
2023年7月(11)
2023年6月(7)
2023年5月(14)
2023年4月(7)
2023年3月(7)
2023年2月(12)
2023年1月(1)
2022年10月(3)
更多

阅读排行榜

1. 第六节、双目视觉之相机标定(69614)
2. 第三十七节、人脸检测MTCNN和人脸识别Facenet(附源码)(51290)
3. 第十九节、基于传统图像处理的目标检测与识别(HOG+SVM附代码)(41911)
4. 第七节、双目视觉之空间坐标计算(41718)
5. 第九节、人脸检测之Haar分类器(40311)

推荐排行榜

1. 第七节、双目视觉之空间坐标计算(14)
2. 第十一节、Harris角点检测原理(附源码)(11)
3. 第九节、人脸检测之Haar分类器(10)
4. 第三十三节，目标检测之选择性搜索-Selective Search(10)
5. 第三十七节、人脸检测MTCNN和人脸识别Facenet(附源码)(8)

最新评论

1. Re:第二十五节，初步认识目标定位、特征点检测、目标检测
@大奥特曼打小怪兽 谢谢博主回复o(￣▽￣)ブ...
--au3h2o
2. Re:第二十五节，初步认识目标定位、特征点检测、目标检测
@au3h2o 吴恩达的视频...
--大奥特曼打小怪兽
3. Re:第二十五节，初步认识目标定位、特征点检测、目标检测
写的真好，另外麻烦问一下作者，这个ppt是哪里的呀，有出处吗，谢谢分享
--au3h2o
4. Re:Rockchip RK3566 - orangepi-build脚本分析

公告&打赏

如果可以把这些参数，在制作镜像的时候就一起弄到镜像里面，然后u-boot一读取镜像，就马上可以知道这些参数了。不需要在制作好镜像之后再另外告诉u-boot这些参数。这种带有以上参数的镜像格式就是Legacy uImage。

最后一个 FIT uImage ，其主要目的是为了支持基于device tree的unify kernel。

1.2 Legacy uImage

1.2.1 配置

使能需要打开的宏：

```
CONFIG_IMAGE_FORMAT_LEGACY=y
```

注意，这个宏在自动生成的autoconf.mk中会自动配置，不需要额外配置。

1.2.2 制作

编译完u-boot之后，使用u-boot目录下tools/mkimage工具来制作uImage。命令如下：

```
mkimage -A arm -O linux -C none -T kernel -a 0x20008000 -e 0x20008040 -n Linux_Image -d

各个参数意义如下
Usage: mkimage -l image
          -l ==> list image header information
       mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n name -d data_file[:da
          -A ==> set architecture to 'arch'  // 架构
          -O ==> set operating system to 'os' // 操作系统
          -T ==> set image type to 'type' // 镜像类型
          -C ==> set compression type 'comp' // 压缩类型
          -a ==> set load address to 'addr' (hex) // 加载地址
          -e ==> set entry point to 'ep' (hex) // 入口地址
          -n ==> set image name to 'name' // 镜像名称，注意不能超过32B
          -d ==> use image data from 'datafile' // 输入文件
          -x ==> set XIP (execute in place)
```

1.2.3 使用

将生成的Legacy uImage下载到内存中，使用bootm <download addr>命令启动kernel中。

但是注意，如果使用Legacy uImage后面还需要根据实际情况决定是否需要传入ramdisk以及dtb的在内存中的地址，否则可能会导致bootm失败。

格式如下：

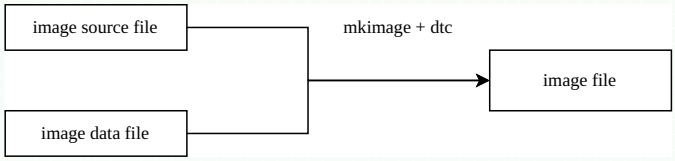
```
bootm <Legacy uImage addr> <ramdisk addr>  <dtb addr>
```

1.3 FIT uImage

1.3.1 FIT介绍

FIT是flattened image tree的简称，它采用了device tree source filse（DTS）的语法，生成的image文件也和dtb文件类似（称做itb）。

其通过一定语法和格式将一些需要使用到的镜像（例如kernel、dtb以及文件系统）组合到一起生成一个image file。其生成步骤如下图所示：



其中image source file(.its)和device tree source file(.dts)类似，负责描述要生成的image file的信息。mkimage和dtc工具，可以将.its文件以及对应的image data file，打包成一个image file。

这里我们有必要对涉及到的这几类文件进行一个总结：

- its文件： image source file，类似于dts文件，负责描述要声称的image的的信息。需要自行进行构造；
- itb文件： 最终得到的image文件，类似于dtb文件，也就是uboot可以直接对其进行识别和解析的FIT u Image；

@超超加油 有，csdn，不过那个只会同步一次，文章都不是最新的...
--大奥特曼打小怪兽
5. Re:Rockchip RK3566 - orangepi-build脚本分析
博主有其他平台账号同步发文章吗
--超超加油

公告&打赏

- mkimage： mkimage则负责dtc的角色，用于通过解析its文件、获取对应的镜像，最终生成一个u-boot可以直接进行识别和解析的itb文件；
- image data file： 实际使用到的镜像文件；

mkimage将its文件以及对应的image data file，打包成一个itb文件，也就是u-boot可以识别的image file（FIT uImage）。我们将这个文件下载到么内存中，使用bootm命令就可以执行了。

1.3.2 配置

使能需要打开的宏：

```
CONFIG_FIT=y
```

1.3.3 创建its文件

因为mkimage是根据its文件中的描述来打包镜像生成itb文件（FIT uImage），所以首先需要制作一个its文件，在its文件中描述需要被打包的镜像，主要是kernel镜像，dtb文件，ramdisk镜像。

关于its文件的语法，可以参考这篇博客FIT介绍。简单的例子如下：

```
/*
 * U-Boot uImage source file for "X project"
 */
/dts-v1/;

/ {
    description = "U-Boot uImage source file for X project";
    #address-cells = <1>;

    images {
        kernel@tiny210 {
            description = "Unify(TODO) Linux kernel for project-x";
            data = /incbin/("/home/hlos/code/xys/temp/project-x/build/out/linux/arch/arı
            type = "kernel";
            arch = "arm";
            os = "linux";
            compression = "none";
            load = <0x20008000>;
            entry = <0x20008000>;
        };
        fdt@tiny210 {
            description = "Flattened Device Tree blob for project-x";
            data = /incbin/("/home/hlos/code/xys/temp/project-x/build/out/linux/arch/arı
            type = "flat_dt";
            arch = "arm";
            compression = "none";
        };
        ramdisk@tiny210 {
            description = "Ramdisk for project-x";
            data = /incbin/("/home/hlos/code/xys/temp/project-x/build/out/rootfs/initra
            type = "ramdisk";
            arch = "arm";
            os = "linux";
            compression = "gzip";
        };
    };

    configurations {
        default = "conf@tiny210";
        conf@tiny210 {
            description = "Boot Linux kernel with FDT blob";
            kernel = "kernel@tiny210";
            fdt = "fdt@tiny210";
            ramdisk = "ramdisk@tiny210";
        };
    };
};
```

注意，可以有多个kernel节点或者fdt节点等等，兼容性更强。同时，可以有多种configurations，来对kernel、fdt、ramdisk来进行组合，使FIT-uImage可以兼容于多种板子，而无需重新进行编译烧写。

1.3.4 生成FIT uImage

公告&打赏

生成的命令相对Legacy uImage较为简单，因为信息都在its文件里面描述了：

```
${UBOOT_OUT_DIR}/tools/mkimage -f ${UIMAGE_ITS_FILE} ${UIMAGE_ITB_FILE}
```

其中 -f 指定需要编译的source文件，并在后面指定需要生成的image文件（一般以.itb为后缀，例如u-boot.itb）。

1.3.5 使用

将生成的FIT uImage下载到内存某个地址，使用bootm <FIT uImage addr>命令启动。

u-boot会自动解析出FIT uImage中，kernel、ramdisk、dtb的信息，使用起来相当方便。

1.4 其他

更多关于内核镜像的介绍以及制作的内容参考：Rockchip RK3399 - 移植linux 5.2.8。

回到顶部

二、linux内核启动

2.1 autoboot_command

autoboot_command函数定义在common/autoboot.c文件中：

```
void autoboot_command(const char *s)
{
    debug("### main_loop: bootcmd=\"%s\"\n", s ? s : "<UNDEFINED>");

    if (stored_bootdelay != -1 && s && !abortboot(stored_bootdelay)) {
        run_command_list(s, -1, 0);
    }
}
```

如果在u-boot启动倒计时结束之前，没有按下任何键，将会执行那么将执行run_command_list，此函数会执行参数s指定的一系列命令，也就是bootcmd中配置中的命令，bootcmd中保存着默认的启动命令。

在默认环境变量default_environment中定义有：

```
#ifdef    CONFIG_BOOTCOMMAND
    "bootcmd="    CONFIG_BOOTCOMMAND        "\0"
#endif
```

2.2 do_bootm

由于要执行bootm命令，所以我们需要打开与bootm命令相关的文件进行分析，bootm命令定义在cmc/bootm.c文件中：

```
U_BOOT_CMD(
    bootm,    CONFIG_SYS_MAXARGS,    1,    do_bootm,
    "boot application image from memory", bootm_help_text
);
```

找到对应的do_bootm函数，去除无用的代码：

```
/*******************************************************************/
/* bootm - boot application image from image in memory */
/*******************************************************************/

int do_bootm(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    /* determine if we have a sub command */
    argc--; argv++;
    if (argc > 0) {
        char *endp;

        simple_strtoul(argv[0], &endp, 16);
        /* endp pointing to NULL means that argv[0] was just a
         * valid number, pass it along to the normal bootm processing
         *
         * If endp is ':' or '#' assume a FIT identifier so pass
         * along for normal processing.
         *
```

公告&打赏