(1条消息) 设备树基础知识_m0_46703823的博客-CSDN博客_fdt_subnode_offset

特殊节点
在根节点"/“中有两个特殊的子节点: aliases和chosen
aliases子节点 aliases的含义就是"别名”
通常定义形式如下
aliases {
can0 = &flexcan1; ==> 这里就是将 &flexcan1节点的别名设置为can0
… …
}
但是实际引用节点时，我们不怎么会使用别名，而是直接使用&label的形式(&flexcan1)来引用该节点

chosen子节点
chosen子节点并不是一个真实的设备，该节点的主要目的是为了实现uboot传递数据到linux内核，
前面说了 uboot会将环境变量bootargs中的值作为参数传递给linux内核
在我们imx6ill-alientek-emmc.dts文件中定义的chosen节点如下
chosen {
stdout-path = &uart1
}

我们在文件系统中的 /proc/device-tree/chosen中发现该目录下有以下文件
bootargs name stdout-path
cat bootargs ==>发现bootargs文件的内容就是我们在uboot中设置的bootargs环境变量的值
问题如下，问题1: 我们在设备树中的chosen中没有定义bootargs属性，但是现实的文件系统中却存在bootargs文件
问题2: bootargs文件中的内容为什么和我们在uboot中设置的bootargs环境变量的值一致
得出结论，bootargs文件中的内容和uboot有关，而且不是在linux内核中实现bootargs文件的，而是uboot过程中想chosen节点中添加了bootargs属性
uboot知道bootargs环境变量的值和设备树执行文件.dtb的位置，所以就有可能是uboot将bootargs环境变量作为chosen节点的属性添加到设备树中

代码实现
在uboot源码中全局搜索"chosen"字符串，来寻找相关操作
~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “chosen” ./
结果:
./common/fdt_support.c:227: nodeoffset = fdt_find_or_add_subnode(fdt, 0, “chosen”); //fdt来发现or新增chosen子节点 差不多就是这里了
int fdt_chosen(void *fdt)*
{
……
/ find or create “/chosen” node. /
nodeoffset = fdt_find_or_add_subnode(fdt, 0, “chosen”);
|
/*
* fdt_find_or_add_subnode() - find or possibly add a subnode of a given node 查找或可能添加给定节点的子节点
*
* @fdt: pointer to the device tree blob 指向设备树blob的指针
* @parentoffset: structure block offset of a node 节点的结构块偏移量
* @name: name of the subnode to locate 要定位的子节点的名称
*
* fdt_subnode_offset() finds a subnode of the node with a given name.
* If the subnode does not exist, it will be created.
*/

```
str = getenv("bootargs");                       //获取环境变量bootargs的值
if (str) {
        //给设备树fdt下的chosen节点(nodeoffset)添加属性名称为 bootargs 的属性，属性值为booatargs环境变量的值(str)
        err = fdt_setprop(fdt, nodeoffset, "bootargs", str,
```

```
                        strlen(str) + 1);
        ... ...
}

return fdt_fixup_stdout(fdt, nodeoffset);
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

```
}
```

开始方向追踪
(1) ./common/fdt_support下的fdt_chosen()

```
~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “fdt_chosen” ./
==>
./common/image-fdt.c:507: if (fdt_chosen(blob) < 0) {
(2) ./common/image-fdt.c下的image_setup_libfdt()
int image_setup_libfdt(bootm_headers_t *images, void *blob,int of_size, struct lmb *lmb)
{
if (fdt_chosen(blob) < 0) {
printf(“ERROR: /chosen node create failed\n”);
goto err;
}
… …
}
```

```
~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “image_setup_libfdt” ./
==>
./common/image.c:1376: ret = image_setup_libfdt(images, *of_flat_tree, of_size, lmb);
int image_setup_linux(bootm_headers_t *images)
{

... ...
if (IMAGE_ENABLE_OF_LIBFDT && of_size) {
        ret = image_setup_libfdt(images, *of_flat_tree, of_size, lmb);
        if (ret)
                return ret;
}

return 0;
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

```
}
(3) ./common/image.c下的image_setup_linux()
```

```
~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “image_setup_linux” ./
==>
./arch/arm/lib/bootm.c:211: if (image_setup_linux(images)) {
static void boot_prep_linux(bootm_headers_t *images)
{
char *commandline = getenv(“bootargs”);

if (IMAGE_ENABLE_OF_LIBFDT && images->ft_len) {
```

```
1
```

#ifdef CONFIG_OF_LIBFDT
debug(“using: FDT\n”);
if (image_setup_linux(images)) {
printf(“FDT creation failed! hanging…”);
hang();
}
#endif
… …
}
(4) ./arch/arm/lib/bootm.c 下的 boot_prep_linux()

~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “boot_prep_linux” ./
==>
./arch/mips/lib/bootm.c:338: boot_prep_linux(images);
int do_bootm_linux(int flag, int argc, char * const argv[],bootm_headers_t *images)
{

```
if (flag & BOOTM_STATE_OS_PREP) {
        boot_prep_linux(images);
        return 0;
}
... ...
```

```
1
2
3
4
5
```

}
(5) ./arch/arm/lib/bootm.c 下的 do_bootm_linux() //到这里这个函数是不是就熟悉了？ 前面uboot分析时，说过uboot的流程

~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “do_bootm_linux” ./
==>
./common/bootm_os.c:438: [IH_OS_LINUX] = do_bootm_linux,
static boot_os_fn *boot_os[] = { //不同的os对应不同的操作函数
#ifdef CONFIG_BOOTM_LINUX
[IH_OS_LINUX] = do_bootm_linux,
#endif
… …
};
int boot_selected_os(int argc, char * const argv[], int state, bootm_headers_t *images, boot_os_fn *boot_fn)
{
arch_preboot_os();
boot_fn(state, argc, argv, images); ==> do_bootm_linux(state, argc, argv, images);

… …

```
1
```

}
(6) ./common/bootm_os.c 下的 boot_selected_os()

~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “boot_selected_os” ./
==>
./common/bootm.c:684: ret = boot_selected_os(argc, argv, BOOTM_STATE_OS_FAKE_GO,

int do_bootm_states(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[],int states, bootm_headers_t *images, int boot_progress)
{
/ Now run the OS! We hope this doesn’t return */
if (!ret && (states & BOOTM_STATE_OS_GO))
ret = boot_selected_os(argc, argv, BOOTM_STATE_OS_GO,

images, boot_fn);

… … …

- 1

}
(7) ./common/bootm_os.c 下的 do_bootm_states()

~/linux/IMX6ULL/uboot/uboot-imx-rel_imx_4.1.15_2.1.0_ga_alientek$ grep -rn “do_bootm_states” ./
==>
./cmd/bootm.c:639: ret = do_bootm_states(cmdtp, flag, argc, argv,

int do_bootz(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
… …
ret = do_bootm_states(cmdtp, flag, argc, argv,
BOOTM_STATE_OS_PREP | BOOTM_STATE_OS_FAKE_GO |
BOOTM_STATE_OS_GO,
&images, 1);

```
return ret;
```

- 1

}
(8) ./cmd/bootm.c:639 下的 do_bootz()

==>跟到这里 差不多了， 因为是bootz命令才会调用do_bootz函数 所以 实际上是在bootz命令时实现的将bootargs环境变量作为chosen节点的属性