[uboot] （番外篇）uboot之fdt介绍_ooonebook的博客-CSDN博客

以下例子都以project X项目tiny210（s5pv210平台,armv7架构）为例

[uboot] uboot流程系列：
[project X] tiny210(s5pv210)上电启动流程（BL0-BL2）
[project X] tiny210(s5pv210)从存储设备加载代码到DDR
[uboot]（第一章）uboot流程——概述
[uboot]（第二章）uboot流程——uboot-spl编译流程
[uboot]（第三章）uboot流程——uboot-spl代码流程
[uboot]（第四章）uboot流程——uboot编译流程
[uboot]（第五章）uboot流程——uboot启动流程
[uboot]（番外篇）global_data介绍
[uboot]（番外篇）uboot relocation介绍

建议先看《**[uboot]** （番外篇）**uboot relocation**介绍》和《**[uboot]** （第四章）**uboot**流程——**uboot**编译流程》

==========

因为在学习**uboot**的**driver module**，发现有必要先把**uboot**的**fdt**整明白点。所以这里就先学习一下**fdt**咯。

一、介绍

FDT，flatted device tree，扁平设备树。熟悉linux的人对这个概念应该不陌生。
简单理解为将部分设备信息结构存放到device tree文件中。
uboot最终将其device tree编译成dtb文件，使用过程中通过解析该dtb来获取板级设备信息。
uboot的dtb和kernel中的dtb是一致的。这部分建议直接参考wowo的dtb的文章
Device Tree（一）：背景介绍
Device Tree（二）：基本概念
Device Tree（三）：代码分析

关于**uboot**的**fdt**，可以参考**doc/README.fdt-control**。

二、**dtb**介绍

1、**dtb**结构介绍

结构体如下
DTB header
alignment gap
memory reserve map
alignment gap
device-tree structure
alignment gap
device-tree string

dtb header结构如下：

结构体如下
magic
totalsize
off_dt_struct
off_dt_strings
off_mem_rsvmap
version
……

其中，magic是一个固定的值，0xd00dfeed（大端）或者0xedfe0dd0（小端）。
以s5pv210-tiny210.dtb为例：
执行”hexdump -C s5pv210-tiny210.dtb | more”命令

```
@:dts$ hexdump -C s5pv210-tiny210.dtb | more
00000000  d0 0d fe ed 00 00 5a cc  00 00 00 38 00 00 58 14  |......Z....8..X.|
00000010  00 00 00 28 00 00 00 11  00 00 00 10 00 00 00 00  |...(............|
```

- 1
- 2
- 3

可以看到dtb的前面4个字节就是0xd00dfeed，也就是magic。
综上，我们只要提取待验证**dtb**的地址上的数据的前四个字节，与**0xd00dfeed**（大端）或者**0xedfe0dd0**（小端）进行比较，如果匹配的话，就说明对应待验证**dtb**就是一个合法的**dtb**。

2、**dtb**在**uboot**中的位置

dtb可以以两种形式编译到uboot的镜像中。

- **dtb**和**uboot**的**bin**文件分离
  - 如何使能
    需要打开CONFIG_OF_SEPARATE宏来使能。
  - 编译说明
    在这种方式下，uboot的编译和dtb的编译是分开的，先生成uboot的bin文件，然后再另外生成dtb文件。
    具体参考《[uboot]（第四章）uboot流程——uboot编译流程》。
  - 最终位置
    dtb最终会追加到**uboot**的**bin**文件的最后面。也就是**uboot.img**的最后一部分。
    因此，可以通过uboot的结束地址符号，也就是_end符号来获取dtb的地址。
    具体参考《[uboot]（第四章）uboot流程——uboot编译流程》。
- **dtb**集成到**uboot**的**bin**文件内部
  - 如何使能
    需要打开CONFIG_OF_EMBED宏来使能。
  - 编译说明
    在这种方式下，在编译uboot的过程中，也会编译dtb。
  - 最终位置
    注意：最终**dtb**是包含到了**uboot**的**bin**文件内部的。
    dtb会位于uboot的.dtb.init.rodata段中，并且在代码中可以通过__dtb_dt_begin符号获取其符号。
    因为这种方式不够灵活，文档上也不推荐，所以后续也不具体研究，简单了解一下即可。
- 另外，也可以通过**fdtcontroladdr**环境变量来指定**dtb**的地址
  可以通过直接把dtb加载到内存的某个位置，并在环境变量中设置fdtcontroladdr为这个地址，达到动态指定dtb的目的。
  在调试中使用。

三、**uboot**中如何支持**fdt**

1、相关的宏

- CONFIG_OF_CONTROL
  用于配置是否使能FDT。
  ./source/configs/tiny210_defconfig:312:CONFIG_OF_CONTROL=y
- CONFIG_OF_SEPARATE、CONFIG_OF_EMBED
  配置dtb是否集成到uboot的bin文件中。具体参考上述。一般都是使用分离的方式。

2、如何添加一个**dtb**

以tiny210为例，具体可以参考project X项目中uboot的git记录：8a371676710cc0572a0a863255e25c35c82bb928
（1）在Makefile中添加对应的目标dtb
arch/arm/dts/Makefile

```
dtb-$(CONFIG_TARGET_TINY210) += \
       s5pv210-tiny210.dtb
```

（2）创建对应的dts文件
arch/arm/dts/s5pv210-tiny210.dts,注意文件名要和Makefile中的dtb名一致

```
/dts-v1/;
/{
};
```

（3）打开对应的宏
configs/tiny210_defconfig

```
CONFIG_OF_CONTROL=y
CONFIG_OF_SEPARATE=y
```

（4）因为最终的编译出来的dtb可能会多个，这里需要为tiny210指定一个dtb
configs/tiny210_defconfig

```
CONFIG_DEFAULT_DEVICE_TREE="s5pv210-tiny210"
```

编译，解决一些编译错误，就可以发现最终生成了u-boot.dtb文件。
通过如下“hexdump -C u-boot.dtb | more”命令可以查看我们的dtb文件，得到部分内容如下：

```
hlos@node4:u-boot$ hexdump -C u-boot.dtb | more
00000000  d0 0d fe ed 00 00 01 a4  00 00 00 38 00 00 01 58  |...........8...X|
00000010  00 00 00 28 00 00 00 11  00 00 00 10 00 00 00 00  |...(............|
00000020  00 00 00 4c 00 00 01 20  00 00 00 00 00 00 00 00  |...L... ........|
00000030  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00  |................|
```

四、**uboot**中如何获取**dtb**

1、整体说明

在uboot初始化过程中，需要对dtb做两个操作:

- 获取dtb的地址，并且验证dtb的合法性
- 因为我们使用的dtb并没有集成到uboot的bin文件中，也就是使用的CONFIG_OF_SEPARATE方式。因此，在relocate uboot的过程中并不会去relocate dtb。因此，这里我们还需要自行为dtb预留内存空间并进行relocate。关于uboot relocate的内容请参考《[uboot] （番外篇）uboot relocation介绍》。
- relocate之后，还需要重新获取一次dtb的地址。

这部分过程是在init_board_f中实现，参考《[uboot] （第五章）uboot流程——uboot启动流程》。

对应代码common/board_f.c

```
static init_fnc_t init_sequence_f[] = {
...
#ifdef CONFIG_OF_CONTROL
    fdtdec_setup,
#endif
...
    reserve_fdt,
...
    reloc_fdt,
...
}
```

后面进行具体函数的分析。

2、获取dtb的地址，并且验证dtb的合法性（fdtdec_setup）

对应代码如下:
lib/fdtdec.c

```
int fdtdec_setup(void)
{
#if CONFIG_IS_ENABLED(OF_CONTROL)

# ifdef CONFIG_OF_EMBED

    gd->fdt_blob = __dtb_dt_begin;


# elif defined CONFIG_OF_SEPARATE

    gd->fdt_blob = (ulong *)&_end;

# endif


    gd->fdt_blob = (void *)getenv_ulong("fdtcontroladdr", 16,
                        (uintptr_t)gd->fdt_blob);

#endif


    return fdtdec_prepare_fdt();

}


int fdtdec_prepare_fdt(void)
{
    if (!gd->fdt_blob || ((uintptr_t)gd->fdt_blob & 3) ||
        fdt_check_header(gd->fdt_blob)) {
        puts("No valid device tree binary found - please append one to U-Boot binary, use u-boot-dtb.bin or define CONFIG_OF_EMBED. For sandbox, use -d <file.dtb>\n");
        return -1;


    }
    return 0;
}
```

```
    • 23
    • 24
    • 25
    • 26
    • 27
    • 28
    • 29
    • 30
    • 31
    • 32
    • 33
    • 34
    • 35
    • 36
    • 37
    • 38
```

验证dtb的部分可以参考《[kernel 启动流程] （第四章）第一阶段之——dtb的验证》。

3、为**dtb**分配新的内存地址空间（**reserve_fdt**）

relocate的内容请参考《[uboot] （番外篇）uboot relocation介绍》。
common/board_f.c中

```
static int reserve_fdt(void)
{
#ifndef CONFIG_OF_EMBED


    if (gd->fdt_blob) {
        gd->fdt_size = ALIGN(fdt_totalsize(gd->fdt_blob) + 0x1000, 32);


        gd->start_addr_sp -= gd->fdt_size;
        gd->new_fdt = map_sysmem(gd->start_addr_sp, gd->fdt_size);

        debug("Reserving %lu Bytes for FDT at: %08lx\n",
              gd->fdt_size, gd->start_addr_sp);
    }
#endif

    return 0;
}
```

```
    • 1
    • 2
    • 3
    • 4
    • 5
    • 6
    • 7
    • 8
    • 9
    • 10
    • 11
    • 12
    • 13
    • 14
    • 15
    • 16
    • 17
    • 18
    • 19
```

4、**relocate dtb**（**reloc_fdt**）

relocate的内容请参考《[uboot] （番外篇）uboot relocation介绍》。
common/board_f.c中

```
static int reloc_fdt(void)
{
#ifndef CONFIG_OF_EMBED


    if (gd->flags & GD_FLG_SKIP_RELOC)

        return 0;
    if (gd->new_fdt) {
        memcpy(gd->new_fdt, gd->fdt_blob, gd->fdt_size);

        gd->fdt_blob = gd->new_fdt;

    }
#endif

    return 0;
}
```

```
    • 1
    • 2
    • 3
    • 4
    • 5
    • 6
    • 7
    • 8
    • 9
    • 10
    • 11
    • 12
    • 13
    • 14
    • 15
```

- 16
- 17
- 18

五、**uboot**中**dtb**解析的常用接口

gd->fdt_blob已经设置成了dtb的地址了。
注意，fdt提供的接口都是以gd->fdt_blob（dtb的地址）为参数的。

1、接口功能

以下只简单说明几个接口的功能，没有深究到实现原理。先说明几个，后续继续补充。
另外，用节点在dtb中的偏移地址来表示一个节点。也就是节点变量node中，存放的是节点的偏移地址

- lib/fdtdec.c中
  - **fdt_path_offset**
    int fdt_path_offset(const void *fdt, const char *path)
    eg：node = fdt_path_offset(gd->fdt_blob, “/aliases”);
    功能：获得dtb下某个节点的路径path的偏移。这个偏移就代表了这个节点。
  - **fdt_getprop**
    const void *fdt_getprop(const void *fdt, int nodeoffset, const char *name, int *lenp)
    eg：mac = fdt_getprop(gd->fdt_blob, node, “mac-address”, &len);
    功能：获得节点node的某个字符串属性值。
  - **fdtdec_get_int_array**、**fdtdec_get_byte_array**
    int fdtdec_get_int_array(const void *blob, int node, const char *prop_name, u32 *array, int count)
    eg：ret = fdtdec_get_int_array(blob, node, “interrupts”, cell, ARRAY_SIZE(cell));
    功能：获得节点node的某个整形数组属性值。
  - **fdtdec_get_addr**
    fdt_addr_t fdtdec_get_addr(const void *blob, int node, const char *prop_name)
    eg：fdtdec_get_addr(blob, node, “reg”);
    功能：获得节点node的地址属性值。
  - **fdtdec_get_config_int**、**fdtdec_get_config_bool**、**fdtdec_get_config_string**
    功能：获得config节点下的整形属性、bool属性、字符串等等。
  - **fdtdec_get_chosen_node**
    int fdtdec_get_chosen_node(const void *blob, const char *name)
    功能：获得chosen下的name节点的偏移
  - **fdtdec_get_chosen_prop**
    const char *fdtdec_get_chosen_prop(const void *blob, const char *name)
    功能：获得chosen下name属性的值
- lib/fdtdec_common.c中
  - **fdtdec_get_int**
    int fdtdec_get_int(const void *blob, int node, const char *prop_name, int default_val)
    eg：bus->udelay = fdtdec_get_int(blob, node, “i2c-gpio,delay-us”, DEFAULT_UDELAY);
    功能：获得节点node的某个整形属性值。
  - **fdtdec_get_uint**
    功能：获得节点node的某个无符号整形属性值。