(1条消息) u-boot-2021.01（imx6ull）启动流程分析之一：分析启动流程之前的准备知识和工作___ASDFGH的博客-CSDN博客

前言

本文章主要以基于ARM® Cortex-A7内核的Freescale i.MX6ULL这款SoC探索u-boot-2021.01启动linux内核的流程。在了解启动流程之后，如果需要移植不同版本的u-boot或移植到其他平台的SoC，相信也能按照同样的思路去排查定位问题所在。

在查看文章的过程中，如果遇到描述有错误的地方，还希望大家悉心指出。

1、关于Freescale i.MX6ULL

i.mx6ull这款SoC到底是如何一款芯片，这里不过多地描述，随便一个搜索引擎都能找到比较详细的描述，这里主要还是以分析u-boot启动出发，了解这款SoC与u-boot相关的一些内容，或者说与一般芯片的差异部分。

- **u-boot**程序的格式：与其他厂家芯片（如s3c24x0）不一样，烧录到i.mx6ull的u-boot程序不是bin格式，而是imx格式，这是因为芯片的启动方式有所不同。i.mx6ull的u-boot程序是通过使用imxdownload程序在u-boot程序的基础上添加了**“头部”**，这个“头部”就是由”IVT+BootData+DCD“组成，假设生成的bin文件为u-boot.bin，那么用于烧录的u-boot.imx就等于“IVT+BootData+DCD+u-boot.bin”。
- **“头部”**的内容：（了解即可，一般不用太深入探究。参考：《IMX6ULL参考手册.pdf》）

① **IVT**（**Image vector table**）：包含IVT头部、程序入口地址、DCD地址、Boot Data地址等信息，如表格展示：

Table 8-26. IVT format

header
entry: Absolute address of the first instruction to execute from the image
reserved1: Reserved and should be zero
dcd: Absolute address of the image DCD. The DCD is optional so this field may be set to NULL if no DCD is required. See Device Configuration Data (DCD) for further details on the DCD.
boot data: Absolute address of the boot data
self: Absolute address of the IVT. Used internally by the ROM.
csf: Absolute address of the Command Sequence File (CSF) used by the HAB library. See High-Assurance Boot (HAB) for details on the secure boot using HAB. This field must be set to NULL when not performing a secure boot
reserved2: Reserved and should be zero

其中header部分内容：Tag固定为0xD1、Length为2个字节大端保存IVT长度、Version为0x40或0x41，结构如图：

② **BootData**：包含镜像要拷贝到的目的地址和大小（与**plugin**标志位均用32bit空间表示），格式如图：

Table 8-27. Boot data format

start	Absolute address of the image
length	Size of the program image
plugin	Plugin flag (see Plugin image)

③ **DCD**（**Device configuration data**）：MMDC 控制器， DDR等外设的寄存器初始化数据。

- **“头部”**如何被使用：芯片内部的boot rom程序会读取解析imx格式的文件头部来初始化外设，然后将imx文件从Flash中拷贝到外部RAM中运行。也正因为在**imx6ull**中，**u-boot**程序一开始就是在外部**RAM**中运行，所以它的**“代码重定位”**是RAM->RAM的代码**“搬运”**，而不是Flash->RAM。

2、先编译一遍u-boot

2.1 为什么要先编译

编译后会在u-boot根目录生成一些比较重要的文件。比如我们就可以通过`u-boot.lds`文件获取程序各个段的分布、通过`u-boot.map`文件查看各个段的内容分布（尤其是text代码段，可以查看某个函数是在哪个目录下编译）、更有各个子目录下编译生成的.o中间文件、甚至还可以通过命令`xxx-objdump -D -m arm u-boot > u-boot.dis`生成反汇编文件来定位运行异常的问题等等。所以，先编译一遍u-boot就显得很有必要。

2.2 编译流程

查看configs目录下关于i.mx6ull的默认配置，发现有2个：

```
mx6ull_14x14_evk_defconfig
mx6ull_14x14_evk_plugin_defconfig
```

- 1
- 2

我们选择前者就行，因为后者通过《IMX6ULL参考手册.pdf》可以知道这是为了支持以太网等方式启动，原文如下：

8.8 Plugin image

> The ROM supports a limited number of boot devices. When using other devices as a boot source (for example, Ethernet, CDROM, or USB), the supported boot device must be used (typically serial ROM) as a firmware to provide the missing boot drivers.

故编译步骤如下：

```
tar xjf u-boot-2021.01.tar.bz2
cd u-boot-2021.01/
make mx6ull_14x14_evk_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

- 1
- 2
- 3
- 4

编译后生成以下u-boot相关文件：

```
-rwxrwxr-x   1 book book 3571160 2月  14 11:52 u-boot
-rw-rw-r--   1 book book  371855 2月  14 11:52 u-boot.bin
-rw-rw-r--   1 book book   12569 2月  14 11:51 u-boot.cfg
-rw-rw-r--   1 book book    6288 2月  14 11:52 u-boot.cfg.configs
-rw-rw-r--   1 book book   28571 2月  14 11:52 u-boot.dtb
-rw-rw-r--   1 book book  371855 2月  14 11:52 u-boot-dtb.bin
-rw-rw-r--   1 book book    3244 2月  14 11:52 u-boot-dtb.cfgout
-rw-rw-r--   1 book book  375808 2月  14 11:52 u-boot-dtb.imx
-rw-rw-r--   1 book book     194 2月  14 11:52 u-boot-dtb.imx.log
-rw-rw-r--   1 book book    1719 2月  14 11:52 u-boot.lds
-rw-rw-r--   1 book book  609654 2月  14 11:52 u-boot.map
-rwxrwxr-x   1 book book  343284 2月  14 11:52 u-boot-nodtb.bin
-rwxrwxr-x   1 book book 1029978 2月  14 11:52 u-boot.srec
-rw-rw-r--   1 book book  154428 2月  14 11:52 u-boot.sym
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14

编译后生成的可以直接烧录的`u-boot-dtb.imx`，从名字里可以知道imx6ull的u-boot程序里使用了dts设备树。直接执行命令`grep "CONFIG_OF_" .config | grep "=y"`搜索一下：

```
CONFIG_OF_CONTROL=y
CONFIG_OF_SEPARATE=y
CONFIG_OF_TRANSLATE=y
CONFIG_OF_LIBFDT=y
```

- 1
- 2
- 3
- 4

除了表示支持linux使用设备树的`CONFIG_OF_LIBFDT`外，还有几个定义。在根目录下的README文件就有关于它们的说明：

README (~/work/imx6ull/system/u-boot-2021.01) - VIM

```
699
700 - Device tree:
701     CONFIG_OF_CONTROL
702     If this variable is defined, U-Boot will use a device tree
703     to configure its devices, instead of relying on statically
704     compiled #defines in the board file. This option is
705     experimental and only available on a few boards. The device
706     tree is available in the global data as gd->fdt_blob.
707
708     U-Boot needs to get its device tree from somewhere. This can
709     be done using one of the three options below:
710
711     CONFIG_OF_EMBED
712     If this variable is defined, U-Boot will embed a device tree
713     binary in its image. This device tree file should be in the
714     board directory and called <soc>-<board>.dts. The binary file
715     is then picked up in board_init_f() and made available through
716     the global data structure as gd->fdt_blob.
717
718     CONFIG_OF_SEPARATE
719     If this variable is defined, U-Boot will build a device tree
720     binary. It will be called u-boot.dtb. Architecture-specific
721     code will locate it at run-time. Generally this works by:
722
723       cat u-boot.bin u-boot.dtb >image.bin
724
725     and in fact, U-Boot does this for you, creating a file called
726     u-boot-dtb.bin which is useful in the common case. You can
727     still use the individual files if you need something more
728     exotic.
729
730     CONFIG_OF_BOARD
731     If this variable is defined, U-Boot will use the device tree
732     provided by the board at runtime instead of embedding one with
733     the image. Only boards defining board_fdt_blob_setup() support
734     this option (see include/fdtdec.h file).
735
```


由于imx6ull默认配置中定义了`CONFIG_OF_SEPARATE`，所以实际上u-boot-dtb.bin就是u-boot.bin+u-boot.dtb，由于上面是直接使用cat命令将它们合并在一起，所以u-boot.dtb的存储位置其实是

紧挨着u-boot.bin尾部的。而对于imx6ull的u-boot程序来说，用于烧录的镜像文件就是u-boot-dtb.imx=IVT+BootData+DCD+u-boot.bin+u-boot.dtb。

注意，u-boot程序中的设备树和用于启动linux内核时用的设备树并不冲突，它们只是同时都使用了设备树来代替大量的板级细节信息描述而已。

此外，并非所有的SoC都支持u-boot程序中使用设备树，但它们的启动流程基本差不多，只是获取一些硬件信息描述时有些差别，整体启动流程还是一样的。

未完待续...