

原

[uboot]（第四章）uboot流程——uboot编译流程

2016年11月01日 21:23:36

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)从存储加载代码到DDR
  - [uboot]（第一章）uboot流程——uboot-spl
  - [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]（番外篇）global\_data介绍
  - [uboot]（番外篇）uboot relocation介绍

建议先看《[project X] tiny210(s5pv210)上电启动流程（BL0-BL2）》，根据例子了解一下上电之后的BL0\BL1\BL2阶段，以及各个阶段的运行位置，建议先看《[uboot]（第二章）uboot流程——uboot-spl编译流程》，其编译流程基本上是类似的。最大区别在于dtb的编译。

=====

## 一、uboot编译和生成文件

### 0、说明

现在的uboot已经做得和kernel很像，最主要的一点是，uboot也使用了dtb的方法，将设备树和代码分离开来（当然可以通过宏来控制）。  
project-x/u-boot/configs/tiny210\_defconfig

```
1 CONFIG_OF_CONTROL=y
2 // 用于表示是否使用了dtb的方式
3
4 CONFIG_OF_SEPARATE=y
5 // 是否将dtb和uboot分离表一
```

所以在uboot的编译中，和spl的最大区别是还要编译dtb。（前面我们将的spl是没有使用dtb的，当然好像也可以使用dtb，只是我没有试过）。

### 1、编译方法

在project X项目中，所有镜像，包括uboot、kernel、rootfs都是放在build目录下进行编译的。具体去参考该项目build的Makefile的实现。  
假设config已经配置完成，在build编译命令如下：

```
1 make uboot
```

Makefile中对应的命令如下：  
project-x/build/Makefile

```
1 BUILD_DIR=$(shell pwd)
2 OUT_DIR=$(BUILD_DIR)/out
3 UBOOT_OUT_DIR=$(OUT_DIR)/u-boot
4 UBOOT_DIR=$(BUILD_DIR)/../u-boot
```

```
7      make -C $(UBOOT_DIR) CROSS_COMPILE=$(CROSS_COMPILE) KBUILD_OUTPUT=$(UBOOT_OUT_DIR) $(BOARD_NAME)_defconfig
8      make -C $(UBOOT_DIR) CROSS_COMPILE=$(CROSS_COMPILE) KBUILD_OUTPUT=$(UBOOT_OUT_DIR)
9  ## -C $(UBOOT_DIR) 指定了编译目录是uboot的源码目录，也就是uboot的代码根目录下执行make
10 ## CROSS_COMPILE=$(CROSS_COMPILE) 指定了交叉编译器
11 ## KBUILD_OUTPUT=$(UBOOT_OUT_DIR) 指定了最终编译的输出目录是build/out/u-boot.
```

最终，相当于进入了uboot目录执行了make动作。

2、生成文件

最终编译完成之后，会在project-x/build/out/u-boot下生成如下文件：

- 1 arch common dts include net tools u-boot.cfg u-boot.lds u-boot.srec
- 2 board disk example scripts System.map u-boot u-boot.dtb u-boot.map u-boot.sym
- 3 cmd drivers fs u-boot.bin u-boot-dtb.bin u-boot-nodtb.bin

其中，arch、common、dts、include、board、drivers、fs等等目录是对应代码的编译目录，各个目录下都会生成相应的built.o，是由同目录下的目标文件编译生成的。重点说一下以下几个文件：

文件
u-boot
u-boot-nodtb.bin
u-boot.dtb
u-boot-dtb.bin
u-boot.bin
u-boot.lds
System.map
u-boot.cfg

二、uboot编译流程

1、编译整体流程

根据一、2生成的文件说明可知简单流程如下：

- ( 1 ) 各目录下built-in.o的生成
- ( 2 ) 由所有built-in.o以u-boot.lds为连接脚本通过连接来生成u-boot
- ( 3 ) 由u-boot生成u-boot-nodtb.bin
- ( 4 ) 由生成uboot的dtb文件
- ( 5 ) 由u-boot-nodtb.bin和u-boot.dtb生成u-boot-dtb.bin

( 6 ) 由u-boot-dtb.bin复制生成u-boot.bin

写评论

## 2、具体编译流程分析

我们直接从make uboot命令分析，是从uboot下的Makefile的依赖关系来分析整个编译流程。  
注意，这个分析顺序和上述的整体流程的顺序是反着的。

• ( 1 ) 入口分析

在project-x/u-boot/Makefile中

1 all: \$(ALL-y)

2 ALL-y += u-boot.srec u-boot.sym System.map u-boot.cfg binary\_size\_check

u-boot.bin就是我们的目标，所以以后需要主要研究u-boot.bin的依赖关系。

• ( 2 ) u-boot.bin的依赖关系

在project-x/u-boot/Makefile中

1 ifeq (\$(CONFIG\_OF\_SEPARATE),y)

2 ## CONFIG\_OF\_SEPARATE用于定义是否有DTB并且是否是和uboot分开编译的。

3 ## tiny210是有定义这个宏的，所以走的是上面这路

4

5 u-boot-dtb.bin: u-boot-nodtb.bin dts/dt.dtb FORCE

6 \$(call if\_changed,cat)

7 ## 由u-boot-nodtb.bin和dts/dt.dtb连接在一起，先生成u-boot-dtb.bin

8 ## \$(call if\_changed,cat)会调用到cmd\_cat函数，具体实现我们不分析了

9

10 u-boot.bin: u-boot-dtb.bin FORCE

11 \$(call if\_changed,copy)

12 ## 直接将u-boot-dtb.bin复制为u-boot.bin

13 ## \$(call if\_changed,copy)会调用到cmd\_copy函数，具体实现我们不分析了

14

15 else

16 u-boot.bin: u-boot-nodtb.bin FORCE

17 \$(call if\_changed,copy)

18 endif

对应于上述二、1 ( 5 ) 流程和上述二、1 ( 6 ) 流程。  
后续有两个依赖关系要分析，分别是u-boot-nodtb.bin和dts/dt.dtb。  
u-boot-nodtb.bin依赖关系参考下述二、2 ( 3 ) -2 ( 6 ) 。  
dts/dt.dtb依赖关系参考下述二、2 ( 7 )  
其中u-boot-nodtb.bin的依赖关系和SPL的相当类似，可以先参考一下《[uboot]（第二章）uboot流程——uboot-spl编译流程》。

• ( 3 ) u-boot-nodtb.bin的依赖关系

在project-x/u-boot/Makefile中

1 u-boot-nodtb.bin: u-boot FORCE

2 \$(call if\_changed,objcopy)

3 \$(call DO\_STATIC\_RELA,\$<,\$@,\$(CONFIG\_SYS\_TEXT\_BASE))

4 \$(BOARD\_SIZE\_CHECK)

5 ## \$(call if\_changed,objcopy)表示当依赖文件发生变化时，将依赖文件经过objcopy处理之后得到目标文件。

6 ## 也就是通过objcopy把u-boot的符号信息以及一些无用信息去掉之后，得到了u-boot-nodtb.bin。

如上述Makefile代码u-boot-nodtb.bin依赖于u-boot, 并且由u-boot经过objcopy操作之后得到。

对应于上述二、1 (3) 流程。

#### • (4) u-boot的依赖关系

在project-x/u-boot/Makefile中

```
1 u-boot: $(u-boot-init) $(u-boot-main) u-boot.lds FORCE
2     $(call if_changed, u-boot__)
3 ## $(call if_changed,u-boot__)来生成目标
4 ## $(call if_changed,u-boot__)对应cmd_u-boot__命令
```

如上, u-boot依赖于\$(u-boot-init)、\$(u-boot-main)和u-boot.lds, 并且最终会调用cmd\_u-boot\_\_来生成u-boot。

cmd\_u-boot\_\_实现如下

project-x/u-boot/Makefile

```
1 cmd_u-boot__ := $(LD) $(LDFLAGS) $(LDFLAGS_u-boot) -o $@ \
2 -T u-boot.lds $(u-boot-init) \
3 --start-group $(u-boot-main) --end-group \
4 $(PLATFORM_LIBS) -boot.map
```

将cmd\_u-boot\_\_通过echo命令打印出来之后得到如下 ( 拆分出来看的 ) :

project-x/u-boot/Makefile

```
1 /project-x/build/arm-none-linux-gnueabi-4.8/bin/arm-none-linux-gnueabi-ld
2 -pie --gc-sections -Bstatic -Ttext 0x23E00000
3 -o u-boot
4 -T u-boot.lds
5 arch/arm/cpu/armv7/start.o
6 --start-group
7 arch/arm/cpu/built-in.o arch/arm/cpu/armv7/built-in.o arch/arm/lib/built-in.o arch/arm/mach-s5pc1xx/built-in.o board/samsung/common
8 --end-group
9 arch/arm/lib/eabi_compat.o
10 -L /project-x/build/arm-none-linux-gnueabi-4.8/bin/../lib/gcc/arm-none-linux-gnueabi/4.8.3 -lgcc
11 -Map u-boot.map
```

可以看出上述是一条连接命令, 以u-boot.lds为链接脚本, 把\$(u-boot-init)、\$(u-boot-main)的指定的目标文件连接到u-boot中。

**并且已经指定输出文件为u-boot, 连接脚本为u-boot.lds。**

连接很重要的东西就是连接标识, 也就是 \$(LD) \$(LDFLAGS) \$(LDFLAGS\_u-boot)的定义。

尝试把\$(LD) \\$(LDFLAGS) \\$(LDFLAGS\_u-boot) 打印出来, 结果如下 :

```
1 LD=/project-x/build/arm-none-linux-gnueabi-4.8/bin/arm-none-linux-gnueabi-ld
2 LDFLAGS=
3 LDFLAGS_u-boot=-pie --gc-sections -Bstatic -Ttext 0x23E00000
```

LDFLAGS\_u-boot定义如下

```
1 LDFLAGS_u-boot += -pie
2 LDFLAGS_u-boot += $(LDFLAGS_FINAL)
3 ifneq ($(CONFIG_SYS_TEXT_BASE),)
4 LDFLAGS_u-boot += -Ttext $(CONFIG_SYS_TEXT_BASE)
5 endif
6
7 ## 当指定CONFIG_SYS_TEXT_BASE时, 会配置连接地址。在tiny210项目中, 定义如下:
8 ## ./include/configs/tiny210.h:52:#define CONFIG_SYS_TEXT_BASE 0x23E00000
9
10 ## $(LDFLAGS_FINAL)在如下几个地方定义了
11 ## ./config.mk:19:LDFLAGS_FINAL :=
12 ## ./config.mk:80:LDFLAGS_FINAL += -Bstatic
13 ## ./arch/arm/config.mk:16:LDFLAGS_FINAL += --gc-sections
14 ## 通过上述LDFLAGS_u-boot=-pie --gc-sections -Bstatic -Ttext 0x23E00000也就可以理解了
```

对应于上述二、1（2）流程。  
关于u-boot依赖的说明在（5）、（6）中继续介绍

1

•（5）u-boot-init & u-boot-main 系（代码是如何被编译的）

先看一下这两个值打印出来的

1 u-boot-init=arch/arm/cpu/armv7/start.o

2 u-boot-main= arch/arm/cpu/armv7/built-in.o arch/arm/lib/built-in.o arch/arm/mach-s5pc1xx/built-in.o board/s

可以观察到是一堆目标文件的路径，这些目标文件最终都要被连接到u-boot中。

u-boot-init & u-boot-main的定义如马：

project-x/u-boot/Makefile

写评论

目录

收藏

微信

微博

QQ

```
1 u-boot-init := $(head-y)
2 ## head-y定义在如下位置
3 ## ./arch/arm/Makefile:7 d-y := arch/arm/cpu/$(CPU)/start.o
4
5 libs-y += lib/
6 libs-y += fs/
7 libs-y += net/
8 libs-y += disk/
9 libs-y += drivers/
10 libs-y += drivers/dma/
11 libs-y += drivers/gpio/
12 libs-y += drivers/i2c/
13 ...
14 u-boot-dirs := $(patsubst %/,%, $(filter %/, $(libs-y))) tools examples
15 ## 过滤出路径之后，加上tools目录和example目录
16
17 libs-y := $(patsubst %/, %/built-in.o, $(libs-y))
18 ## 先加上后缀built-in.o
19
20 u-boot-main := $(libs-y)
```

那么u-boot-init & u-boot-main是如何生成的呢？  
需要看一下对应的依赖如下：

```
1 $(sort $(u-boot-init) $(u-boot-main)): $(u-boot-dirs) ;
2 ## 也就是说$(u-boot-init) $(u-boot--main)依赖于$(u-boot-dirs)
3 ## sort函数根据首字母进行排序并去除掉重复的。
4 ##u-boot-dirs := $(patsubst %/,%, $(filter %/, $(libs-y))) tools examples
5 ## $(filter %/, $(libs-y)过滤出'/'结尾的字符串，注意，此时$(libs-y)的内容还没有加上built-in.o文件后缀
6 ## patsubst去掉字符串中最后的'/'的字符。
7 ## 最后u-boot-dirs打印出来如下：
8 ## u-boot-dirs=arch/arm/cpu arch/arm/cpu/armv7 arch/arm/lib arch/arm/mach-s5pc1xx board/samsung/common board/samsung/tiny210 cmd co
```

u-boot-dirs依赖规则如下：

```
1 PHONY += $(u-boot-dirs)
2 $(u-boot-dirs): prepare scripts
3 $(Q)$(MAKE) $(build)=$@
4 ## 依赖于prepare scripts
5 ## prepare会导致prepare0、prepare1、prepare2、prepare3目标被执行，最终编译了tools目录下的东西，生成了一些工具
6 ## 然后执行$(Q)$(MAKE) $(build)=$@
7 ## 也就是会对每一个目标文件依次执行make \$(build)=目标文件
```

对每一个目标文件依次执行make \$(build)=目标文件  
\$(build)定义如下：  
project-x/u-boot/scripts/Kbuild.include

以arch/arm/mach-s5pc1xx为例

“(MAKE) \$(build)=\$@"展开后格式如下

make -f project-x/u-boot/scripts/Makefile.build obj=arch/arm/mach-s5pc1xx。

Makefile.build定义built-in.o、.lib.o等目标文件的生成规则。这个Makefile文件生成了子目录的.lib、built-in.o以及目标文件.o。

Makefile.build第一个编译目标是\_\_build，如下

```

1 PHONY := __build
2 __build:
3 ## 所以会直接编译执行__build这个目标，其依赖如下
4 __build: $(if $(KBUILD_EXTRA_TARGETS),$(builtin-target) $(lib-target) $(extra-y)) \
5           $(if $(KBUILD_EXTRA_TARGETS),$(obj-m) $(modorder-target)) \
6           $(subdir-ym) $(srcs)
7 @:
8 ## 和builtin.o相关的是依赖builtin-target。下面来看这个依赖。
9 builtin-target := $(obj)-t-in.o
10 ## 以obj=arch/arm/mach-s5pc1xx为例，那么builtin-target就是arch/arm/mach-s5pc1xx/built-in.o。
11
12 ## 依赖关系如下：
13 $(builtin-target): $(obj)-ORCE
14 $(call if_changed,link_o_target)
15 ## $(call if_changed,link_o_target)将所有依赖连接到$(builtin-target)，也就是相应的built-in.o中了。
16 ## 具体实现可以查看cmd_link_o_target的实现，这里不详细说明了。
17
18 ## 那么$(obj-y)是从哪里来的呢？是从相应目录下的Makefile中include得到的。
19 # The filename Kbuild has precedence over Makefile
20 kbuild-dir := $(if $(filter /%, $(src)), $(src), $(srctree)/$(src))
21 kbuild-file := $(if $(wildcard $(kbuild-dir)/Kbuild), $(kbuild-dir)/Kbuild, $(kbuild-dir)/Makefile)
22 include $(kbuild-file)
23 ## 当obj=arch/arm/mach-s5pc1xx时，得到对应的kbuild-file=u-boot/arch/arm/mach-s5pc1xx/Makefile
24 ## 而在u-boot/arch/arm/mach-s5pc1xx/Makefile中定义了obj-y如下：
25 ## obj-y = cache.o
26 ## obj-y += reset.o
27 ## obj-y += clock.o
28 ## 对应obj-y对应一些目标文件，由C文件编译而来，这里就不说明了。

```

后面来看目标文件的编译流程

./scripts/Makefile.build/scripts/Makefile.build

```

1 # Built-in and composite module parts
2 $(obj)/%.o: $(src)/%.c $(recordmcount_source) FORCE
3 $(call cmd,force_checksrc)
4 $(call if_changed_rule,cc_o_c)
5 ## 调用cmd_cc_o_c对.c文件进行编译
6
7 ## cmd_cc_o_c格式如下：
8 cmd_cc_o_c = $(CC) $(c_flags) -c -o $@ $<
9 ## $(CC) $(c_flags)打印出来如下：
10 ## CC=/home/disk3/xys/temp/project-x/build/arm-none-linux-gnueabi-4.8/bin/arm-none-linux-gnueabi-gcc
11 ## c_flags=-Wp,-MD,arch/arm/mach-s5pc1xx/.clock.o.d -nostdinc -isystem /home/disk3/xys/temp/project-x/build/arm-none-linux-gnueabi-

```

对应于上述二、1 ( 1 ) 流程。

#### • ( 6 ) u-boot.lds依赖关系

这里主要是为了找到一个匹配的连接文件。

```

1 u-boot.lds: $(LDSCRIPT) prepare FORCE
2 $(call if_changed_dep,cpp_lds)
3 ifndef LDSCRIPT
4 ifeq ($(wildcard $(LDSCRIPT)),)
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

8          LDSCRIPT := $(srctree)/$(CPUDIR)/u-boot.lds
9      endif
10     ifeq ($(wildcard $(LDSCRIPT)),)
11         LDSCRIPT = $(srctree)/arch/$(ARCH)/cpu/u-boot.lds
12     endif
13 endif
14 ## 也就是说依次从board/板 写评论、cpudir目录、arch/架构/cpu/目录下去搜索u-boot.lds文件。
15 ## 例如，tiny210(s5pv210)最终会在./arch/arm/cpu/下搜索到u-boot.lds

```

综合，最终指定了project-X/u-boot/arch/arm/cpu/u-boot.lds作为连接脚本。

目录

收藏

#### • (7) dts/dt.dtb依赖关系

该依赖关系的主要目的是生成dtb。

首先了解dts文件被放在了arch/arm/目录下，并通过dts下的Makefile进行选择。

Makefile如下（剪切出一部分）

project-X/u-boot/arch/arm/dts/Makefile

微信

微博

```

1 dtb=$(CONFIG_S5PC110) += 1xx-goni.dtb
2 dtb=$(CONFIG_EXYNOS5) += exynos5250-arndale.dtb \
3     exynos5250-snow.dtb \
4     exynos5250-spring.dtb \
5     exynos5250-smdk5250.dtb \
6     exynos5420-smdk5420.dtb \
7     exynos5420-peach-pit.dtb \
8     exynos5800-peach-pi.dtb \
9     exynos5422-odroidxu3.dtb
10 dtb=$(CONFIG_TARGET_TINY210) += \
11     s5pv210-tiny210.dtb
12 ## 填充选择dtb-y
13
14 targets += $(dtb-y)
15
16 # Add any required device tree compiler flags here
17 DTC_FLAGS +=
18 ## 用于添加DTC编译选项
19
20 PHONY += dtbs
21 dtbs: $(addprefix $(obj)/, $(dtb-y))
22     @:
23 ## 伪目标，其依赖为$(dtb-y)加上了源路径，如下
24 ## arch/arm/dts/s5pc1xx-goni.dtb
25 ## arch/arm/dts/s5pv210-tiny210.dtb
26 ## 后续会使用到这个伪目标

```

接下来看一下dts/dt.dtb的依赖关系

```

1 dtbs dts/dt.dtb: checkdtc u-boot
2     $(Q)$(MAKE) $(build)=dts dtbs
3 ## checkdtc依赖用于检查dtc的版本
4 ## u-boot一旦发生变化那么就重新编译一遍dtb
5 ## 重点关注命令 $(Q)$(MAKE) $(build)=dts dtbs
6 ## 展开来就是make -f ~/project-x/u-boot/scripts/Makefile.build obj=dts dtbs
7 ## 我们相当于值在/scripts/Makefile.build下执行了目标dtbs

```

在scripts/Makefile.build中dtbs的目标定义在哪里呢

project-X/u-boot/scripts/Makefile.build

```

1 kbuild-file := $(if $(wildcard $(kbuild-dir)/Kbuild),$(kbuild-dir)/Kbuild,$(kbuild-dir)/Makefile)
2 include $(kbuild-file)
3 ## 把对应的Makefile路径包含了进去，也就是arch/arm/dts/Makefile
4 ## 如前面所说 arch/arm/dts/Makefile中定义了dtbs的目标

```

登录

注册

```
7  ## 这里我们就找到对应的依赖关系了，依赖就是$(obj)/, $(dtb-y)，举个例子就是arch/arm/dts/s5pv210-tiny210.dtb
8
9  include scripts/Makefile
10 ## 包含了scripts/Makefile 1 在编译dts的时候会用到
```

接下来就是\$(obj)/, \$(dtb-y)的依赖  
project-X/u-boot/scripts/Makefile.lds

```
1 $(obj)/%.dtb: $(src)/%.c 目录 RCE
2     $(call if_changed,dtc)
3 ## 使用了通配符的方式
4 ## 这样就通过dtc对dts编译 dtb文件
```

对应于上述二、1（4）流程。

### 三、一些重点定义

- 1、连接标志  
在二、2（4）中说明。  
连接命令在cmd\_u-boot\_\_中，如下

```
1 cmd_u-boot__ ?= $(LD) $(LDFLAGS) $(LDFLAGS_u-boot) -o $@ \
2 -T u-boot.lds $(u-boot-init) \
3 --start-group $(u-boot-main) --end-group \
4 $(PLATFORM_LIBS) -Map u-boot.map
```

连接标识如下：

```
1 LD=~/.project-x/build/arm-none-linux-gnueabi-4.8/bin/arm-none-linux-gnueabi-ld
2 LDFLAGS=
3 LDFLAGS_u-boot=-pie --gc-sections -Bstatic -Ttext 0x23E00000
```

LDFLAGS\_u-boot定义如下

```
1 LDFLAGS_u-boot += -pie
2 LDFLAGS_u-boot += $(LDFLAGS_FINAL)
3 ifneq ($(CONFIG_SYS_TEXT_BASE),)
4 LDFLAGS_u-boot += -Ttext $(CONFIG_SYS_TEXT_BASE)
5 endif
```

‘-o’指定了输出文件是u-boot，‘-T’是指定了连接脚本是当前目录下的u-boot.lds，-Ttext指定了连接地址是CONFIG\_SYS\_TEXT\_BASE。

- 2、连接地址  
在二、2（4）中说明。  
CONFIG\_SYS\_TEXT\_BASE指定了u-boot.bin的连接地址。这个地址也就是uboot的起始运行地址。  
对于tiny210，其定义如下（可以进行修改）  
/include/configs/tiny210.h

```
1 #define CONFIG_SYS_TEXT_BASE 0x23E00000
```

- 3、连接脚本  
在二、2（6）中说明。  
u-boot/arch/arm/cpu/u-boot.lds

```
1 u-boot.lds: $(LDSCRIPT) prepare FORCE
2     $(call if_changed_dep,cpp_lds)
3 ifndef LDSCRIPT
4     ifeq ($(wildcard $(LDSCRIPT)),)
5         LDSCRIPT = $(src)/arch/arm/cpu/u-boot.lds
```



```

8         LDSCRIPT := $(srctree)/$(CPUDIR)/u-boot.lds
9     endif
10    ifeq ($(wildcard $(LDSCRIPT)),)
11        LDSCRIPT = $(srctree)/arch/$(ARCH)/cpu/u-boot.lds
12    endif
13 endif

```

写评论

综上，最终指定了project-X/u-boot/arch/arm/cpu/u-boot.lds作为连接脚本。

## 四、uboot链接脚本说

### 1、连接脚本整体分析

相对比较简单，直接看连接脚本的 `project-x/u-boot/arch/arm/cpu/u-boot.lds` 前面有一篇分析连接脚本的文章了 [\[wingsnel 启动流程\] 前篇——vmlinux.lds分析](#)，可以参考一下。参考如下，只提取了一部分：

```

1  ENTRY(_start)
2  //定义了地址为_start的地址，我们分析代码就是从这个函数开始分析的!!!
3      . = 0x00000000;
4
5  //以下定义文本段
6      . = ALIGN(4);
7      .text :
8      {
9          __image_copy_start = .;
10 //定义__image_copy_start这个标号地址为当前地址
11      *(.vectors)
12 //所有目标文件的vectors段，也就是中断向量表连接到这里来
13      CPUDIR/start.o (.text*)
14 //start.o文件的.text段链接到这里来
15      *(.text*)
16 //所有目标文件的.text段链接到这里来
17      }
18
19 //以下定义只读数据段
20      . = ALIGN(4);
21      .rodata : { *(SORT_BY_ALIGNMENT(SORT_BY_NAME(.rodata*))) }
22
23 //以下定义数据段
24      . = ALIGN(4);
25      .data : {
26          *(.data*)
27 //所有目标文件的.data段链接到这里来
28      }
29
30      . = ALIGN(4);
31
32 //以下定义u_boot_list段，具体功能未知
33      . = ALIGN(4);
34      .u_boot_list : {
35          KEEP(*(SORT(.u_boot_list*)));
36      }
37
38      . = ALIGN(4);
39
40      .image_copy_end :
41      {
42          *(.__image_copy_end)
43      }
44 //定义__image_copy_end符号的地址为当前地址
45 //从__image_copy_start 到__image_copy_end的区间，包含了代码段和数据段。
46

```

目录

收藏

微信

微博

QQ

```

49     *(__rel_dyn_start)
50 }
51 //定义__rel_dyn_start 符号的地址为当前地址，后续在代码中会使用到
52     .rel_dyn : {
53         *(__rel*)
54     }
55     .rel_dyn_end :
56     {
57         *(__rel_dyn_end)
58     }
59 //定义__rel_dyn_end 符号的地址为当前地址，后续在代码中会使用到
60 //从__rel_dyn_start 到__rel_dyn_end 的区间，应该是在代码重定向的过程中会使用到，后续遇到再说明。
61     .end :
62     {
63         *(__end)
64     }
65     _image_binary_end =
66 //定义__image_binary_end 符号的地址为当前地址
67 // 以下定义堆栈段
68     .bss_start __rel_dyn_start (OVERLAY) : {
69         KEEP(*(__bss_start));
70         __bss_base = .;
71     }
72     .bss __bss_base (OVERLAY) : {
73         *(__bss*)
74         . = ALIGN(4);
75         __bss_limit = .;
76     }
77     .bss_end __bss_limit (OVERLAY) : {
78         KEEP(*(__bss_end));
79     }
80 }

```

## 2、以下以.vectors段做说明，

.vectors是uboot链接脚本第一个链接的段，也就是\_start被链接进来的部分，也负责链接异常中断向量表  
先看一下代码project-x/u-boot/arch/arm/lib/vectors.S

```

1  .globl _start
2  .section ".vectors", "ax"
3  @@ 定义在.vectors段中
4
5  _start:
6
7      b    reset
8      ldr pc, _undefined_instruction
9      ldr pc, _software_interrupt
10     ldr pc, _prefetch_abort
11     ldr pc, _data_abort
12     ldr pc, _not_used
13     ldr pc, _irq
14     ldr pc, _fiq
15
16     .globl _undefined_instruction
17     .globl _software_interrupt
18     .globl _prefetch_abort

```

```
22     .globl _fiq
23 @@ 定义了异常中断向量表
```

通过“arm-none-linux-gnueabi-objdump -D u-boot > uboot\_objdump.txt”进行反编译之后，得到了如下指令

```
1  23e00000 <__image_copy_start>:
2  23e00000: ea0000be <reset>
3  23e00004: e59ff014 c, [pc, #20] ; 23e00020 <undefined_instruction>
4  23e00008: e59ff014 c, [pc, #20] ; 23e00024 <software_interrupt>
5  23e0000c: e59ff014 c, [pc, #20] ; 23e00028 <prefetch_abort>
6  23e00010: e59ff014 c, [pc, #20] ; 23e0002c <data_abort>
7  23e00014: e59ff014 c, [pc, #20] ; 23e00030 <not_used>
8  23e00018: e59ff014 c, [pc, #20] ; 23e00034 <irq>
9  23e0001c: e59ff014 c, [pc, #20] ; 23e00038 <fiq>
10
11 // 可以看出以下是异常终端
12 23e00020 <undefined_instruction>:
13 23e00020: 23e00060 mvnccs r0, #96 ; 0x60
14 // 其中，23e00020存放的是 指令处理函数的地址，也就是23e00060
15 // 以下以此类推
16
17 23e00024 <software_interrupt>:
18 23e00024: 23e000c0 mvnccs r0, #192 ; 0xc0
19
20 23e00028 <prefetch_abort>:
21 23e00028: 23e00120 mvnccs r0, #8
22
23 23e0002c <data_abort>:
24 23e0002c: 23e00180 mvnccs r0, #32
25
26 23e00030 <not_used>:
27 23e00030: 23e001e0 mvnccs r0, #56 ; 0x38
28
29 23e00034 <irq>:
30 23e00034: 23e00240 mvnccs r0, #4
31
32 23e00038 <fiq>:
33 23e00038: 23e002a0 mvnccs r0, #10
34 23e0003c: deadbeef cdple 14, 10, cr11, cr13, cr15, {7}
```

### 3、符号表中需要注意的符号

前面我们说过了在tiny210中把连接地址设置为0x23e00000。

project-x/build/out/u-boot/spl/u-boot.map

```
1  Linker script and memory map
2
3  Address of section .text set to 0x23e00000
4  .text          0x23e00000      0x29b28
5  *(.__image_copy_start)
6  __image_copy_start
7          0x23e00000      0x0 arch/arm/lib/built-in.o
8          0x23e00000      __image_copy_start
9  *(.vectors)
10 .vectors      0x23e00000      0x300 arch/arm/lib/built-in.o
11          0x23e00000      _start
12          0x23e00020      _undefined_instruction
13          0x23e00024      _software_interrupt
14          0x23e00028      _prefetch_abort
15          0x23e0002c      _data_abort
16          0x23e00030      _not_used
17          0x23e00034      _irq
18          0x23e00038      _fiq
```

```
21  __image_copy_end
22      0x23e36b78      0x0 arch/arm/lib/built-in.o
23  *(.__rel_dyn_start)
24  __rel_dyn_start      1
25      0x23e36b78      0x0 arch/arm/lib/built-in.o
26  *(.__rel_dyn_end)
27  __rel_dyn_end      写评论
28      0x23e36b78      0x0 arch/arm/lib/built-in.o
29      0x23e36b78      __image_binary_end = .
30  *(.__bss_start)      目录
31  __bss_start      0x23e36b78      0x0 arch/arm/lib/built-in.o
32      0x23e36b78      __bss_start
33  __bss_end      0x23e6b78      0x0 arch/arm/lib/built-in.o
34      0x23e6b78      __bss_end
```

重点关注

\* \_\_image\_copy\_start & \_\_image\_binary\_end

界定了代码空间的位置，用于重定位代码的时候使用，在uboot relocate的过程中，需要把这部分拷贝到uboot的新的地址空间中，后续在新地址空间中具体可以参考《[uboot] （番外篇）uboot relocation介绍》。

\* \_start

在u-boot-spl.lds中ENTRY(\_start)，也就规定了代码的入口函数是\_start。所以后续分析代码的时候就是从这里开始分析。

\* \_\_rel\_dyn\_start & \_\_rel\_dyn\_end

由链接器生成，存放了绝对地址符号的label的地址，用于修改uboot relocate过程中修改绝对地址符号的label的值。具体可以参考《[uboot] （番外篇）uboot relocation介绍》。

\* \_\_image\_binary\_end

综上，u-boot的编译就完成了。

文章标签：

u-boot

编译

个人分类：

uboot

相关热词：

uboot的功能

uboot下命令

uboot命令行

uboot剪裁

uboot组成

- 上一篇

[project X] tiny210(s5pv210)从存储设备加载代码到DDR
- 下一篇

[uboot] （番外篇）global\_data介绍

2018年Python全栈平均薪资是多少？

转型学Python如何从8K提升至20K月薪，多数高薪Python全栈需要掌握Django框架、网络爬虫Scrapy框架、Xpath、PhantomJS、BeautifulSoup、Redis存储和Docker容器技术

 Orangehaswing 2017-03-22 17:22:10 #1楼

spl可以使用DTB。如果用mkimage时候提示size过大，就要生成u-boot-spl-dtb.bin。其中CONFIG\_SPL\_OF\_CONTROL和CONFIG\_OF\_SEPARATE要加，CONFIG\_S

uboot之Makefile编译过程详解

1.主Makefile分析：uboot的version（版本信息）：VERSION = 1 PATCHLEVEL = 3 SUBLEVEL = 4 EXTRAV...

 qq\_25827755 2016-12-15 16:30:24 阅读量：294

u-boot-2016.09 make编译过程分析（一）

### 区块链以太坊与APP开发为什么人才稀少？薪资到底有多高？

区块链以太坊与APP开发为什么人才稀少？薪资到底有多高？区块链以太坊与APP开发为什么人才稀少？他们都在学些什么？区块链的日益火爆和备受追捧，使得区块链开发人员成为稀缺人才，同时更加伴随着高薪

写评论

#### uboot配置和编译过程详解

uboot主Makefile分析1 1、uboot version 目录（ Makefile的24-29行 ）(1)uboot的版本号分3个级别： VERSION：主版本号 PATCHLEVEL：次版本号...

czg13548930186 2016-12-03 11:36:02 阅读量：7628

收藏

#### [IMX6Q]uboot\_v2015.04编译流程分析

执行生成.config文件 #make mx6qecov 微信 roid\_configMakefile: %config: scripts\_basic outputmakefile FORCE ...

kris\_fei 2016-01-18 11:36:02 阅读量：3889

微博

#### Uboot 2017.01 启动流程分析

前言2017.01 UBoot包含两个阶段的启动，一个是SPL启动，一个是正常的启动我们称为第二阶段Uboot。当然，我们也可以选择使用SPL和不使用。 在编译的过程中，是先编译...

kl1125290220 2017-12-01 10:29:30 阅读量：6810

#### [uboot] （第一章）uboot流程——概述

[uboot] uboot流程系列： [project X] tiny210(s5pv210)上电启动流程（ BL0-BL2 ）建议先看 《[project X] tiny210(s5pv210)上电启动...

ooonebook 2016-10-26 22:30:45 阅读量：2946

#### 三、uboot的编译链接过程 (2011-03-10 20:36)

分类： uboot2010.09移植 配置完之后，执行make即可编译，从makefile中可以了解uboot使用了哪些文件、哪个文件先执行，可执行文件占用内存的情况。 下...

mirkerson 2012-08-06 11:38:15 阅读量：3479

#### uboot 2015-01版本启动linux简易流程

uboot 2015-01版

litao31415 2016-10-27 00:39:05 阅读量：493

#### uboot启动流程和架构

概述： 本文将从两个方面来阐述uboot: 1、启动流程 2、架构一、uboot流程图： 从上图中看到红色1,2,3,4,5,7,8,9的标号，下面分别说明每个过程： ...

eZiMu 2017-02-01 15:52:04 阅读量：911

#### u-boot配置和编译过程详解

备注：分析的是OK210开发板自带的uboot\_smdkv210，可能有些部分和其他版本不太一样，但是原理都类似。编译u-boot的步骤make forlinux\_linux\_config make首先...

Kevin\_Mr 2016-05-16 21:43:21 阅读量：13836

#### uboot - 启动内核过程分析

我们都知道u-boot被缔造出来的使命是 启动内核。那么，他是如何完成他的使命的涅！请看下面↓↓↓...（1）我们先来分析下Linux内核镜像这个概念吧。我们编译内核完（编译...

KayChanGEEK 2015-11-29 22:57:28 阅读量：2380

#### uboot学习心得（uboot流程分析）max32590芯片

1、代码框架源码解压以后，我们可以看到以下的文件和文件夹： cpu与处理器相关的文件。每个子目录中都包括cpu.c和interrupt.c、start.S、u-boot.lids。cpu.c：初始化CP...

guoyiyan1987 2018-05-08 10:56:33 阅读量：28

### 移植uboot-2015-10(一)

移植uboot-2015.10.rc1(一) 开发板：y arm 2440 工具：Win7 + VMware + ubuntu U-boot版本：u-...

lee244868149 2015-09-30 14:0 写评论 阅读数：10406

### openwrt的uboot

http://wiki.openwrt.org/doc/techref/boo /uboot

xiaoxiaozhu2010 2014-05-08 1' 收藏 阅读数：4563

### UBoot命令解析与执行流程

本文为CP根据网上内容以及U-Boot 20 本进行整理而成。原文地址为http://blog.chinaunix.net/uid-8867796-id-358806.html ——...

jiujiaobusiniaoo 2016-11-25 16:0 微博 阅读数：1539

### UBOOT 学习心得（UBOO QQ 群分析）

网上找到的UBOOT研究文章，结合自己这几天看的。目前是明白了UBOOT主干程序流程了。开始分析细节部分了。下面是别人写的UBOOT分析。参考了fzb和赵春江两位大

windsun0800 2013-12-14 01:25:34 阅读数：5661

### #嵌入式Linux最小系统移植# 对uboot移植和裁剪的一点点个人思考和总结

思路: 1.分析启动流程 2.移植config文件(smdk440\_config) 3.移植包含控制条件编译宏的.h文件(configs/s3c2440.h) 4.移植板级初始化.c文件(s3c24...

sinat\_26551021 2018-02-09 20:55:06 阅读数：174

### IMX6Solo启动流程-从Uboot到kernel 中

Uboot的C函数入口以及命令的处理流程

baicaiaichibaicai 2015-08-28 10:26:43 阅读数：2102

### uboot编译架构

2015年12月04日 31KB 下载

### 新版UBOOT启动流程

转载请注明地址：http://blog.csdn.net/zsy2020314/article/details/9824035 1.关于启动流程 1.1 启动阶段分为3个，bl0，bl1，...

gujintong1110 2015-10-04 22:49:15 阅读数：2395

### 【u-boot】u-boot-2017.05启动过程分析（一）

u-boot发展至今，版本已经很多，随着版本的升级，框架越来越复杂，不过其启动流程的核心过程都是一样的，本博文以当前最新u-boot-2017.05为例分析其启动过程，主要

qq\_38144425 2017-06-19 12:00:54 阅读数：2464

### uboot启动流程详解(2)-reset

1、cpsr寄存器介绍 通过向模式位M[4:0]里写入相应的数据切换到不同的模式，在对CPSR，SPSR寄存器进行操作不能使用mov，ldr等通用指令，只能使用特权指令msr

silent123go 2016-11-12 18:58:01 阅读数：925

### Uboot启动流程和Kernel启动流程

/\*\*\*\*\*Uboot启动流程（分为两部分）\*\*\*\*\*/ 第一部分（放在start.s中，汇编）1).定义入口（通过链接器脚本...

zhangsan\_3 2016-11-27 16:31:47 阅读数：827

当U-boot完成重定位和初始化外设后，它将正式进入工作状态，可以加载内核镜像到DDR的链接地址中了...

 qq\_28992301

2016-07-10 20:37:46

阅读数：3222

### uboot学习笔记之uboot1.3移植

自己学习嵌入式学习已经有一段时间了，不懂的太多，在这里想把每一步学习的只是一步一步记录下来，以供自己以后查看使用，也希望看到这篇文章的朋友多给点意见

 dghfjj

2016-06-05 00:50:11

评论数：640

### uboot的编译及配置浅析 和 U-boot启动，内核启动详细讲解

uboot的编译及配置浅析 网址：http://blog.csdn.net/evenness/article/details/7395535 U-Boot的源码是通过Git

 a746742897

2016-11-12 16:30:46

阅读数：1340

### uboot移植-从uboot官方源码开始移植过程总结

1 选取源码 下载源码 解压源码 自行登录官网，下载uboot源码，我选取的是2013.10的，因为之后的源码采用类似新的配置模式（能用即可）。我的源码是uboot-2013.

 KayChanGEEK

2016-01-16 16:30:46

阅读数：7072

### uboot的eMMC初始化代码流程分析

源码参考九鼎科技移植的X210开发板捆绑BSP中的uboot，版本为1.3.4mmc初始化函数int mmc\_initialize(bd\_t \*bis)在uboot/lib\_arm/board.c中...

 harrydotter\_wh

2017-09-14 14:36:44

阅读数：955

### uboot启动流程分析之一

最开始的就是start.S 一个可执行的Image 必须有一个入口点并且只能有一个唯一的全局入口，通常这个入口放在Rom(flash)的0x0地址。start.S \_st...

 Stars\_Moon\_Sky

2015-04-22 11:33:38

阅读数：1312

### Uboot，内核，设备树编译步骤

一 . Uboot编译 1.生成配置信息 2.编译 Make O=dir 注意：环境变量导出： 二 . 内核及设备树 编译 1 .根据自己的板级信息，修改设备树 对于本实验板，需要进入i...

 vertor11

2017-05-04 15:43:26

阅读数：1699

### 深入理解uboot 2016 - 基础篇（处理器启动流程分析）

最近一段时间一直在做uboot相关的移植的工作，需要将uboot-2016-7移植到ARMv7的处理器上。正好元旦放假三天闲来无事，有段完整的时间来整理下最近的工作成果。之i

 kernel\_yx

2016-11-05 14:52:35

阅读数：8436

### uboot 2016.05编译uboot.bin和spl

1： Makefile中的all目标编译出相应的文件. 我们来看看这个all目标 all: \$(ALL-y) 2： # Always append ALL so that...


 michaelcao1980

2016-10-25 10:54:40

阅读数：1059

### uboot编译全过程

u-boot的Makefile分析 U - BOOT是一个LINUX下的工程，在编译之前必须已经安装对应体系结构的交叉编译环境，这里只针对ARM，编译器系列软件为arm-linux-\*

 w8713015050275

2014-07-19 15:32:12

阅读数：1743

### [RK3288][Android6.0] U-boot显示模块部分流程小结

Platform: RK3288 OS: Android 6.0 Version: v2014.10 //mipi dsi接口为例: drv\_lcd\_init -> lcd.c lc...

 kris\_fei

2016-10-09 17:36:39


阅读数：1889

### LCD 在uboot和Kernel中的基本流程

转自http://blog.csdn.net/haoyang41/article/details/6766494 LCD显示的基本原理：通过帧缓冲设备（Framebuffer）将应用程序中的图形数据...

### u-boot显示logo

经过个人实践一下两种方法都可以实现，lcd驱动必须能正常运行，想确定是不是正常，就看uboot启动时能不能显示默认的logo。#define CONFIG\_LCD 其次想在u-boc


 sddsighhz

2015-03-12 18:37:35

1 阅读数：1959

### 2017.1.7 \_u-boot的初步认识

今天结束了C语言部分和裸机部分，现在学习UBOOT，学习完毕UBOOT后以后就是在操作系统的基础上来进行操作裸机了。UBOOT是用来启动操作系统的。当我们学


 u012204121

2017-01-18 09:53:10

目录 阅读数：1070

### u-boot-2016.09 make编译流程分析（二）

上一篇文章详尽分析了u-boot执行配置脚本make xxx\_defconfig的整个流程，本文着眼于编译流程，即配置完成后执行make命令生成二进制文件的过程。由于涉及的依赖和命


 guyongqiangx

2016-10-08 23:28

微信 阅读数：2863

### uboot开发笔记一之ralink 20编译

ralink mt7620 uboot开发笔记之一 参考：http://www.ralink.com.tw/TK\_Ralink\_ApSoC\_SDK\_4200\_20131106.tar.bz2,这个ralink的sdk网上有，自行搜索; ...


 openswc

2016-11-19 01:39:08

QQ 阅读数：3767

### u-boot编译学习--uboot编译链接过程

参考博客：http://blog.chinaunix.net/uid-18921523-id-165078.html U - BOOT是一个Linux下的工程，在编译之前必须已经安装...

 zhaolushandong

2016-01-16 11:12:16

阅读数：1986

### U-boot配置及启动流程，RK3288可参考

2017年07月26日 328KB 下载

### IMX6Q Uboot 从零开始编译

Imx6Q Uboot 从零开始编译 作为一个比较新的平台，freescale的这个四核的平台IMX6Q在很多方面的开发都很难下手，国内资源稀少，转载转发较多，且大多数的开发都是


 zhihuihuan

2014-11-18 20:45:52

阅读数：3258

### 关于uboot fastboot 的一些原理分析

1、分区实现 fastboot.c 2、如何用fastboot 烧写wince 内核NK.nb0 文件？


 lqxandroid2012

2013-03-20 09:46:16

阅读数：4278

### uboot之CONFIG\_SPL\_BUILD

首先进行第一步，下载工作：输入 U-Boot 下载的地址，找到自己要下载的 U-Boot 版本， 点击开始下载，下载完成之后开始解压。 U-Boot 下载之后压缩包的压缩方式是.tar。

 michaelcao1980

2015-03-18 10:29:12

阅读数：4443

### TQ2440使用uboot下的命令行进行ftp下载程序到nand flash

看了韦东山第一期视频的第12集，由于手头没有open-jtag，所以着重看了下ftp下载程序的方法。准备工作： 1.下载安装ftp-server工具 http://pan.baidu.com/s...


 yanlutian

2016-08-30 19:23:40

阅读数：1609

### 展讯平台的u-boot驱动流程解析

U-Boot启动内核的过程可以分为两个阶段，两个阶段的功能如下： （1）第一阶段的功能 Ø 硬件设备初始化 Ø 加载U-Boot第二阶段代码到RAM空间 Ø 设置好栈 Ø 跳

 canjianfantasy

2013-08-04 22:57:05

阅读数：2292

### [SPRD1]uboot展讯平台启动流程介绍



 dearsq

2016-04-05 11:08:38

阅读数：1452

1

从零移植uboot 2017 到nu-277(第十一天)

今天开始了，第11天，通过这个我学习，也为将来学习内核打基础吧 首先我昨天加了串口 之后打印出来一些信息 LD lib/built-in.o LD u-boot...

 bushipeien

2017-03-08 17:42:4

阅读数：1656

目录

[uboot]（第三章）uboot流程——uboot-spl代码流程

以下例子都以project X项目tiny210（stm32mp151平台,armv7架构）为例。[uboot] uboot流程系列：[project X] tiny210(s5pv210)上电启动流程...

 ooonebook

2016-10-28 16:24:1

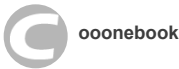
阅读数：2506

微信

微博

QQ

个人资料



关注

原创	粉丝	喜欢	评论
48	145	26	17

等级：

博客 4

访问：11万+

积分：1644排名：3万+

勋章：

恒

最新文章

- [sd card] mmc硬件总线扫描流程（以sd card为例）
- [emmc] emmc总线设置
- [sd card] mmc\_blk层为sd card创建块设备流程
- [sd card] sd card块设备(mmc\_blk)读写流程学习笔记
- [sd card] sd card初始化流程

个人分类

console	2篇
kernel启动流程	8篇
project-X	5篇
uboot	15篇
mmc	17篇

归档

2017年3月	6篇
2017年2月	12篇
2016年12月	3篇
2016年11月	9篇

登录

注册

1

写评论

目录

收藏

微信

微博

QQ

展开

热门文章

[uboot] uboot启动kernel篇（二）——bootm跳转到kernel的流程

阅读量：6901

[uboot] （番外篇）uboot 驱动模型

阅读量：6830

[uboot] （第六章）uboot流程——命令行模式以及命令处理介绍

阅读量：4720

[emmc] emmc总线设置

阅读量：4600

[uboot] uboot启动kernel篇（一）——Legacy-ulmage & FIT-ulmage

阅读量：4433

最新评论

[project X] tiny2...

baidu\_26866585：[reply]woshidahuaidan2011[reply] CPU0运行，其他核心睡眠等待...

[project X] tiny2...

woshidahuaidan2011：写的不错，有没有想过：“s5pv210上电之后，CPU会直接从0x0地址取指令，也就是直接执行...

[emmc] emmc总线设置

weixin\_41711181：楼主，您好，请问你的EMMC数据资料是从哪下载的？

[emmc] emmc总线设置

kingman\_0717：你好，请问在52MHz的DDR模式下，能否把数据总线电压设置为3v呢？我的VCCQ是3.3v，使用...

[uboot] （番外篇）uboo...

chenxiaozhen22：感谢博主的讲解，作为一个应届生都能大概跟上博主的讲解节奏。学到很多东西了。