WTF is Uboot

一个BootLoader，支持多种架构，类似于裸机编程，启动linux的

一般有zimage+dtb(设备树)

Uboot获取

1. 官网

2. SOC厂商比如nxp。st，他们有自己的开发板

3. 做开发板的厂商，理智派

Uboot编译

记得设置好ARCH和CORSS_COMPILE

建议在Makefile里就设置好

Uboot Makefile讲解

编译后的文件

类型	名字	描述	备注
文件夹	api	与硬件无关的 API 函数。	uboot 自带
	arch	与架构体系有关的代码。	
	board	不同板子(开发板)的定制代码。	
	cmd	命令相关代码。	
	common	通用代码。	
	configs	配置文件。	
	disk	磁盘分区相关代码	
	doc	文档。	
	drivers	驱动代码。	
	dts	设备树。	
	examples	示例代码。	
	fs	文件系统。	
	include	头文件。	
	lib	库文件。	
	Licenses	许可证相关文件。	
	net	网络相关代码。	
	post	上电自检程序。	
	scripts	脚本文件。	
	test	测试代码。	
	tools	工具文件夹。	
文件	.config	配置文件，重要的文件。	编译生成的文件
	.gitignore	git 工具相关文件。	uboot 自带
	.mailmap	邮件列表。	
	.u-boot.xxx.cmd (一系列)	这是一系列的文件，用于保存着一些命令。	编译生成的文件
	config.mk	某个 Makefile 会调用此文件。	uboot 自带
	imxdownload	正点原子编写的 SD 卡烧写软件。	正点原子提供
	Kbuild	用于生成一些和汇编有关的文件。	uboot 自带
	Kconfig	图形配置界面描述文件。	
	MAINTAINERS	维护者联系方式文件。	
	MAKEALL	一个 shell 脚本文件，帮助编译 uboot 的。	
	Makefile	主 Makefile，重要文件！	
	mx6ull_alientek_emmc.sh	上一章编写的编译脚本文件	上一章编写的。
	mx6ull_alientek_nand.sh	上一章编写的编译脚本文件	
	README	相当于帮助文档。	uboot 自带
	snapshot.commint	？？？	
	System.map	系统映射文件	编译出来的文件
	u-boot	编译出来的 u-boot 文件。	
	u-boot.xxx (一系列)	生成的一些 u-boot 相关文件，包括 u-boot.bin、u-boot.imx.等	

1. `arm` 文件夹->cpu架构有关

   ->board->freescale-> mx6ullevk 这个文件夹来定义我们的板子

2. `config` 文件夹 ： mx6ull_14x14_ddr512_emmc_defconfig等等这些默认配置

3. `.u-boot.xxx.cmd` 文件

   这玩意有点玄机啊，基本上都是存命令的。比如说.u-boot.bin.cmd （bin）

```
cmd_u-boot.bin := cp u-boot-nodtb.bin u-boot.bin
```

可以看到cp u-boot-nodtb.bin过来并改名对吧

那u-boot-nodtb.bin怎么来的？可以看他的_cmd文件`.u-boot-nodtb.bin.cmd`

```
cmd_u-boot-nodtb.bin := arm-linux-gnueabihf-objcopy --gap-fill=0xff  -j .text -j .secure_text -j .rodata -j .hash -j .data -j .got -j .got.plt -j .u_boot_list -j .rel.dyn -O binary  u-boot u-boot-nodtb.bin
```

使用 objcopy 将 ELF 格式的 u-boot 文件转换为二进制的 u-boot-nodtb.bin 文件

之后就是.u-boot.cmd

```
cmd_u-boot := arm-linux-gnueabihf-ld.bfd   -pie  --gc-sections -Bstatic -Ttext 0x87800000 -o u-boot -T u-boot.lds arch/arm/cpu/armv7/start.o --start-group  arch/arm/cpu/built-in.o  arch/arm/cpu/armv7/built-in.o  arch/arm/imx-common/built-in.o  arch/arm/lib/built-in.o  board/freescale/common/built-in.o  board/freescale/mx6ullevk/built-in.o  cmd/built-in.o  common/built-in.o  disk/built-in.o  drivers/built-in.o  drivers/dma/built-in.o  drivers/gpio/built-in.o  drivers/i2c/built-in.o  drivers/mmc/built-in.o  drivers/mtd/built-in.o  drivers/mtd/onenand/built-in.o  drivers/mtd/spi/built-in.o  drivers/net/built-in.o  drivers/net/phy/built-in.o  drivers/pci/built-in.o  drivers/power/built-in.o  drivers/power/battery/built-in.o  drivers/power/fuel_gauge/built-in.o  drivers/power/mfd/built-in.o  drivers/power/pmic/built-in.o  drivers/power/regulator/built-in.o  drivers/serial/built-in.o  drivers/spi/built-in.o  drivers/usb/dwc3/built-in.o  drivers/usb/emul/built-in.o  drivers/usb/eth/built-in.o  drivers/usb/gadget/built-in.o  drivers/usb/gadget/udc/built-in.o  drivers/usb/host/built-in.o  drivers/usb/musb-new/built-in.o  drivers/usb/musb/built-in.o  drivers/usb/phy/built-in.o  drivers/usb/ulpi/built-in.o  fs/built-in.o  lib/built-in.o  net/built-in.o  test/built-in.o  test/dm/built-in.o --end-group arch/arm/lib/eabi_compat.o  -L /usr/local/arm/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/../lib/gcc/arm-linux-gnueabihf/7.5.0 -lgcc -Map u-boot.map
```

.u-boot.cmd 使用到了 arm-linux-gnueabihf-ld.bfd，也就是链接工具，使用 ld.bfd 将各个 built-in.o 文件链接在一起就形成了 u-boot 文件。uboot 在编译的时候会将同一个目录中的所有.c 文件都编译在一起，并命名为 built-in.o，相当于将众多的.c 文件对应的.o 文件集合在一起，这个就是 u-boot 文件的来源。

然后需要转成Imx

```
cmd_u-boot.imx := ./tools/mkimage -n board/freescale/mx6ull_alientek_emmc/imximage.cfg.cfgtmp -T imximage -e 0x87800000 -d u-boot.bin u-boot.imx
```

用/tools/mkimage将保存在board/freescale/mx6ullevk/imximage-ddr512.cfg.cfgtmp的 IVT 、 DCD 等数据加到bin的头部合成

4. `Makefile`文件

   顶层调用子模块链接

5. `u-boot.xxx`文件

   `u-boot`：编译出来的 ELF 格式的 uboot 镜像文件。

   `u-boot.bin`：编译出来的二进制格式的 uboot 可执行镜像文件。

   `u-boot.cfg`：uboot 的另外一种配置文件。

`u-boot.imx`：u-boot.bin 添加头部信息以后的文件，NXP 的 CPU 专用文件。

`u-boot.lds`：链接脚本。

`u-boot.map`：uboot 映射文件，通过查看此文件可以知道某个函数被链接到了哪个地址上。

`u-boot.srec`：S-Record 格式的镜像文件。

`u-boot.sym`：uboot 符号文件。

`u-boot-nodtb.bin`：和 u-boot.bin 一样，u-boot.bin 就是 u-boot-nodtb.bin 的复制文件。

6. `config` 文件

uboot 配置文件，使用命令“make xxx_defconfig”配置 uboot 以后就会自动生成

CONFIG_CMD_BOOTM=y

如果使能了 bootd 这个命令的话，CONFIG_CMD_BOOTM 就为‘y’。在 cmd/Makefile 中有如下代码：

示例代码 31.1.6 cmd/Makefile 代码

```
1 ifndef CONFIG_SPL_BUILD
2 # core command
3 obj-y += boot.o
4 obj-$(CONFIG_CMD_BOOTM) += bootm.o
5 obj-y += help.o
6 obj-y += version.o
```

在示例代码 31.1.6 中，有如下所示一行代码：

obj-$(CONFIG_CMD_BOOTM) += bootm.o

CONFIG_CMD_BOOTM=y，将其展开就是：

obj-y += bootm.o

也就是给 obj-y 追加了一个“bootm.o”，obj-y 包含着所有要编译的文件对应的.o 文件，这里表示需要编译文件 cmd/bootm.c。相当于通过“CONFIG_CMD_BOOTD=y”来使能 bootm 这个命令，进而编译 cmd/bootm.c 这个文件，这个文件实现了命令 bootm。在 uboot 和 Linux 内核中都是采用这种方法来选择使能某个功能，编译对应的源码文件。

U-Boot顶层Makefile分析

First of all , is the version number .

```
VERSION = 2016
PATCHLEVEL = 03
SUBLEVEL =
EXTRAVERSION =
NAME =
```

1. 主版本号

2. 补丁版本号

3. 次版本号

4. 附加版本信息

5. 名字有关

MAKEFLAGS变量

make可以递归make子目录然后链接，比如subdir这个子目录，可以通过下面的命令编译子目录

```
$(MAKE) -C subdir
```

`$(MAKE)`即调用make命令， -C指定子目录

:::tip

有时候需要向子make传递变量，需要用`export`来导出，不需要就unexport

但是有两个特殊的变量：`SHELL` `MAKEFLAGS`，这两个变量除非使用 “unexport” 声明，否则的话在整个 make 的执行过程中，它们的值始终自动的传递给子 make。在 uboot 的主 Makefile中有如下代码：

```
MAKEFLAGS += -rR --include-dir=$(CURDIR)
```

`-rR`表示禁止使用内置的隐含规则和变量定义，`--include-dir`指明搜索路径，`$(CURDIR)`表示当前目录。

:::

命令输出

编译过程一般都是短命令，需要加`V=1`来输出完整的命令输出，方便调试

控制输出的代码：

```
ifeq ("$(origin V)", "command line")
  KBUILD_VERBOSE = $(V)
endif
ifndef KBUILD_VERBOSE
  KBUILD_VERBOSE = 0
endif

ifeq ($(KBUILD_VERBOSE),1)
  quiet =
  Q =
else
  quiet=quiet_
  Q = @
endif
```

1. 用`ifeq`判断$(origin V)和command line时不时一样

   origin是Makefile的函数，用于告诉变量的来源，相当于判断这个V是不是来源于命令行

2. V=1相当于KBUILD_VERBOSE=1

Makefile 中会用到变量 quiet 和 Q 来控制编译的时候是否在终端输出完整的命令，在顶层Makefile 中有很多如下所示的命令：

```
$(Q)$(MAKE) $(build)=tools
```

按照上面来展开，不输出就是@make ...=tools，输出就是make...=tools

静默输出

使用`make -s`静默输出

```
# If the user is running make -s (silent mode), suppress echoing of
# commands
ifneq ($(filter 4.%,$(MAKE_VERSION)),)  # make-4
ifneq ($(filter %s ,$(firstword x$(MAKEFLAGS))),)
  quiet=silent_
endif
else                    # make-3.8x
ifneq ($(filter s% -s%,$(MAKEFLAGS)),)
  quiet=silent_
endif
endif

export quiet Q KBUILD_VERBOSE
```

用到了`filter <parttern...> , <text>`过滤单词，查询版本对不对

获取了MAKEFLAGS的第一个单词，加入-s的编译选项后，`firstword=xrRs`

展开就是$(filter %s, xrRs)，而$(filter %s, xrRs)的返回值肯定不为空，条件成立，quiet=silent_

设置编译结果输出目录

> kbuild supports saving output files in a separate directory.
>
> To locate output files in a separate directory two syntaxes are supporte
>
> In both cases the working directory must be the root of the kernel src.
>
> 1. O=
>
>   Use "make O=dir/to/store/output/files/"
>
> 2. Set KBUILD_OUTPUT
>
>   Set the environment variable KBUILD_OUTPUT to point to the directory
>
>   where the output files shall be placed.
>
>   export KBUILD_OUTPUT=dir/to/store/output/files/
>
>   make
>
>   The O= assignment takes precedence over the KBUILD_OUTPUT environment
>
>   variable.

使用`O=out`来指定输出目录到out

代码检查

uboot 支持代码检查，使用命令 “make C=1” 使能代码检查，检查那些需要**重新编译**的文件。“make C=2” 用于检查所有的源码文件，顶层 Makefile 中的代码如下：

```
# Call a source code checker (by default, "sparse") as part of the
# C compilation.
#
# Use 'make C=1' to enable checking of only re-compiled files.
# Use 'make C=2' to enable checking of *all* source files, regardless
# of whether they are re-compiled or not.
#
# See the file "Documentation/sparse.txt" for more details, including
# where to get the "sparse" utility.

ifeq ("$(origin C)", "command line")
  KBUILD_CHECKSRC = $(C)
endif
```

```
ifndef KBUILD_CHECKSRC
  KBUILD_CHECKSRC = 0
endif
```

模块编译

`make M=dir`

```
# Use make M=dir to specify directory of external module to build
# Old syntax make ... SUBDIRS=$PWD is still supported
# Setting the environment variable KBUILD_EXTMOD take precedence
ifdef SUBDIRS
  KBUILD_EXTMOD ?= $(SUBDIRS)
endif

ifeq ("$(origin M)", "command line")
  KBUILD_EXTMOD := $(M)
endif


# If building an external module we do not care about the all: rule
# but instead _all depend on modules
PHONY += all
ifeq ($(KBUILD_EXTMOD),)
_all: all
else
_all: modules
endif

ifeq ($(KBUILD_SRC),)
        # building in the source tree
        srctree := .
else
        ifeq ($(KBUILD_SRC)/,$(dir $(CURDIR)))
                # building in a subdirectory of the source tree
                srctree := ..
        else
                srctree := $(KBUILD_SRC)
        endif
endif
objtree     := .
src     := $(srctree)
obj     := $(objtree)

VPATH       := $(srctree)$(if $(KBUILD_EXTMOD),:$(KBUILD_EXTMOD))

export srctree objtree VPATH
```

ifeq ($(KBUILD_EXTMOD),)处判断KBUILD_EXTMOD是不是空，为空，让_all=all

因此要先编译出all。否则的话默认目标_all 依赖 modules，要先编译出 modules，也就是编译模块。一般情况下我们不会在 uboot 中编译模块，所以此处会编译 all 这个目标。

最后导出变量

获取主机架构和系统

```
HOSTARCH := $(shell uname -m | \
    sed -e s/i.86/x86/ \
        -e s/sun4u/sparc64/ \
        -e s/arm.*/arm/ \
        -e s/sa110/arm/ \
        -e s/ppc64/powerpc/ \
        -e s/ppc/powerpc/ \
```

```
        -e s/macppc/powerpc/\
        -e s/sh.*/sh/)

HOSTOS := $(shell uname -s | tr '[:upper:]' '[:lower:]' | \
        sed -e 's/\(cygwin\).*/cygwin/')

export  HOSTARCH HOSTOS
```

调用SHELL获取当前系统架构与主机OS（会将大写的Linux换成小写的）

设置目标架构、交叉编译器、配置文件

```
# set default to nothing for native builds
ifeq ($(HOSTARCH),$(ARCH))
CROSS_COMPILE ?=
endif

KCONFIG_CONFIG  ?= .config
export KCONFIG_CONFIG
```

判断是不是架构一致

可以直接修改顶层 Makefile，在里面加入 `ARCH` 和 `CROSS_COMPILE` 的定义

可以看到调用了.config，是kconfig生成的，基本上是从defconfig搬过来

调用scripts/Kbuild.include