

U-Boot 配置和编译

刘通平 Homepage: <http://www.cs.umass.edu/~tonyliu/>

因为 U-Boot 涉及 160 多种目标板和多种 CPU，如何生成针对具体 CPU 的目标代码？配置就是实现这个目的，通过配置你可以指定具体的 CPU 和目标板，让编译程序在编译时能编译指定的文件。为了描述配置过程，这里我们取一个现存的目标板做分析，比如 OMAP1610 目标板。

4.3.1 U-Boot 的配置

- 1) 首先进入 u-boot 目录，执行 `make omap1610h2_cs0boot_config`。因为 `make` 命令将首先执行顶级目录下的 `Makefile` 文件。
- 2) 在 `Makefile` 文件中，对 `make omap1610h2_cs0boot_config` 的行为进行了如下定义：

```
omap1610inn_config \  
omap1610inn_cs0boot_config \  
omap1610inn_cs3boot_config \  
omap1610inn_cs_autoboot_config \  
omap1610h2_config \  
omap1610h2_cs0boot_config \  
omap1610h2_cs3boot_config \  
omap1610h2_cs_autoboot_config:unconfig  
    @if [ "$(findstring _cs0boot_, $@)" ]; then \  
        echo "#define CONFIG_CS0_BOOT" >> ./include/config.h ; \  
        echo "... configured for CS0 boot"; \  
    elif [ "$(findstring _cs_autoboot_, $@)" ]; then \  
        echo "#define CONFIG_CS_AUTOBOOT" >> ./include/config.h ; \  
        echo "... configured for CS_AUTO boot"; \  
    else \  
        echo "#define CONFIG_CS3_BOOT" >> ./include/config.h ; \  
        echo "... configured for CS3 boot"; \  
    fi;  
    @./mkconfig -a $(call xtract_omap1610xxx,$@) arm arm926ejs omap1610inn
```

当用户进行 `make omap1610h2_cs0boot_config` 时，将首先执行 `unconfig`，然后从命令中查找是否包含“_cs0boot_”、“_cs_autoboot_”或者其它字符串，我们这个命令包含“_cs0boot_”字符串，因此将把语句“`#define CONFIG_CS0_BOOT`”输出到文件 `./include/config.h` 中，然后在屏幕上打印一条相应的信息。

执行 `@./mkconfig -a $(call xtract_omap1610xxx,$@) arm arm926ejs omap1610inn`，即调用相同目录下的脚本文件 `mkconfig`，并把 `arm arm926ejs, omap1610inn` 作为参数传递过去。注意，和 `main()` 函数的传递参数一样，此处 `mkconfig` 属于第一个参数，`arm` 属于第 2 个参数，`arm926ejs` 属于第 3 个参数，`omap1610inn` 属于第 4 个参数。

而脚本文件 mkconfig 的主要内容如下所示：

```
cd ./include
#
# Create link to architecture specific headers
#
rm -f asm
ln -s asm-$2 asm
rm -f asm-$2/arch
ln -s arch-$3 asm-$2/arch

if [ "$2" = "arm" ] ; then
    rm -f asm-$2/proc
    ln -s proc-armv asm-$2/proc
fi

#
# Create include file for Make
#
echo "ARCH    = $2" > config.mk
echo "CPU     = $3" >> config.mk
echo "BOARD   = $4" >> config.mk

[ "$5" ] && [ "$5" != "NULL" ] && echo "VENDOR = $5" >> config.mk

[ "$6" ] && [ "$6" != "NULL" ] && echo "SOC     = $6" >> config.mk

#
# Create board specific header file
#
if [ "$APPEND" = "yes" ] # Append to existing config file
then
    echo >> config.h
else
    > config.h          # Create new config file
fi
echo "/* Automatically generated - do not edit */" >>config.h
echo "#include <configs/$1.h>" >>config.h
```

(1) 进入 include 目录。

(2) 执行

```
rm -f asm
ln -s asm-$2 asm
```

删除目录下的 `asm` 符号链接并执行 `ln -s asm-$2 asm`，因为调用脚本 `mkconfig` 时的第二个参数为 `arm`，因此此句相当于执行了 `ln -s asm-arm asm`，执行的结果是产生一个软符号链接 `asm`，指向 `asm-arm`。

(3) 执行下列语句：

```
rm -f asm-$2/arch
ln -s arch-$3 asm-$2/arch
```

相当于执行 `ln -s arch-arm926ejs asm-arm/arch`，即在 `/include/asm-arm/arch` 创建了一个链接指向 `/include/asm-arm/arch-926ejs`。

3) 通过

```
rm -f asm-$2/proc
ln -s proc-armv asm-$2/proc
```

在 `/include/asm-arm` 下创建了一个软符号连接 `proc` 指向 `/include/asm-arm/proc-armv`。

4) 通过执行

```
echo "ARCH = $2" > config.mk
echo "CPU = $3" >> config.mk
echo "BOARD = $4" >> config.mk
```

把 `ARCH = arm`, `CPU = arm926ejs`, `BOARD = omap1610inn` 写入 `config.mk` 文件。

5) 然后，把这所有的配置写入一个新的 `config.h`。至此，配置工作顺利完成了。

这次编译过程将会产生 `config.mk` 文件和 `config.h` 文件，这两个文件后面将用于编译。

4.3.2 U-Boot 的编译

当配置过程完成后，将涉及到编译过程。编译时一般执行 `make` 或者 `make all` 命令进行编译。一般在 `Makefile` 中定义了相应的编译命令过程，比如 U-Boot 的顶层的 `Makefile` 中有如下定义：

```
ALL = u-boot.srec u-boot.bin System.map
```

```
all: $(ALL)
```

```
u-boot.srec: u-boot
$(OBJCOPY) ${OBJCFLAGS} -O srec $< $@
```

```
u-boot.bin: u-boot
$(OBJCOPY) ${OBJCFLAGS} -O binary $< $@
```

```
u-boot: depend $(SUBDIRS) $(OBJS) $(LIBS) $(LDSCRIPT)
UNDEF_SYM=`$(OBJDUMP) -x $(LIBS) |
sed -n -e 's/.*\(__u_boot_cmd_.*\)/-u\1/p|sort|uniq`;
$(LD) $(LD_FLAGS) $$UNDEF_SYM $(OBJS) \
--start-group $(LIBS) $(PLATFORM_LIBS) --end-group \
-Map u-boot.map -o u-boot
```

```
System.map: u-boot
    @$(NM) $< | \
    grep -v `(compiled)\(\.o$$\)\( [aUw] \)\(\.ng$$\)\(LASH[RL]DI\)' | \
    sort > System.map
```

当用户输入 `make` 命令时，首先会在 `Makefile` 文件中查找首个编译选项，一般而言 `Makefile` 的首个编译选项都是 `all` 所在的编译选项，因此执行 `make` 命令就相当于执行 `make all` 命令。当执行编译命令 `make/make all` 时，将会执行相应的 `all` 程序段。

当执行 `make all` 时，根据

```
all: $(ALL)
```

此处说明 `all` 依赖于宏 `ALL`，而“`ALL = u-boot.srec u-boot.bin System.map`”，因此当执行 `make all` 编译命令时，将会检查 `u-boot.srec`、`u-boot.bin` 和 `System.map` 三个文件是否需要重新生成，而这三个文件也取决于其各自的依赖文件是否发生了改变。

`u-boot.srec` 是 Motorola 的 S-Record 格式的 U-Boot 映像，该映像来自于 ELF 格式的 U-Boot 映像 `u-boot` 文件。通过以下命令生成：

```
u-boot.srec: u-boot
    $(OBJCOPY) ${OBJCFLAGS} -O srec $< $@
```

因此，实际上是调用 `objcopy` 命令生成的，

```
arm-linux-objcopy --gap-fill=0xff -O srec u-boot u-boot.srec。
```

`objcopy` 将进行映像格式的转换（该工具的详细介绍参见第二章“GNU 工具链的编译”），其中，其中 `-O srec` 说明输出映像的格式为 `srec` 的格式。

`u-boot.bin` 是原始二进制映像，该映像也是通过 `objcopy` 命令对 ELF 映像 `u-boot` 转换而成，只是转换的参数有所不同而已：

```
u-boot.bin: u-boot
    $(OBJCOPY) ${OBJCFLAGS} -O binary $< $@
```

即 `-O binary` 让其生成二进制格式的映像。

而 `System.map` 是 `u-boot` 的符号表，即该文件中给出内存地址和相应的符号的映射关系。通过 `System.map` 文件可以知道函数、全局变量和静态变量的地址，当然，由地址也可以查出对应的符号。和 Linux 内核中 `System.map` 生成一样，`System.map` 是通过 `nm` 工具从目标文件 `u-boot` 中提取对应的符号表：

```
System.map: u-boot
    @$(NM) $< | \
    grep -v `(compiled)\(\.o$$\)\( [aUw] \)\(\.ng$$\)\(LASH[RL]DI\)' | \
    sort > System.map
```

而实际上调用 `nm` 工具从 `u-boot` 中提取符号表，然后调用 `grep` 打印相匹配的行，而 `sort` 则是对这些符号排序后输出到 `System.map` 中。

当然，其中这些映像和符号表都取决于 ELF 映像的 `u-boot`，这是这节的重点。`u-boot` 映像的生成取决于以下语句：

```
u-boot: depend $(SUBDIRS) $(OBJS) $(LIBS) $(LDSCRIPT)
```

```

UNDEF_SYM=`$(OBJDUMP) -x $(LIBS) |
            sed -n -e 's/.*\(__u_boot_cmd_.*\)/-u\1/p|sort|uniq`;
$(LD) $(LDFLAGS) $$UNDEF_SYM $(OBJS) \
      --start-group $(LIBS) $(PLATFORM_LIBS) --end-group \
      -Map u-boot.map -o u-boot

```

即 u-boot 依赖于 depend, \$(SUBDIRS), \$(OBJS), \$(LIBS), \$(LDSCRIPT)。

其中, depend 定义了文件的依赖关系, 而这些依赖关系最终将被包含进同级的 Makefile 中, 以便在编译时使用, 比如当执行 make omap1510inn_config 进行配置后, 执行 Make 命令将首先生成对应的.depend 文件, 比如对于 tools 目录下的.depend 文件如下:

```

environment.o: environment.c ../include/config.h \
    ../include/configs/omap1510inn.h ../include/cmd_confdefs.h \
    ../include/configs/omap1510.h ../include/asm/arch/sizes.h \
    ../include/environment.h

```

通过这种方式, 把相应的配置结果给用起来了, 即把相应的 config.h 使用起来了。

而宏 SUBDIRS 的命令如下:

```

SUBDIRS = tools \
    examples \
    post \
    post/cpu
$(SUBDIRS):
    $(MAKE) -C $@ all

```

即将进入到各个列举出来的子目录 (比如 tools, examples, post 和 post/cpu 目录), 并执行 make all 操作。

\$(OBJS)的定义如下:

```
OBJS = cpu/$(CPU)/start.o
```

而 CPU 的类型取决于配置的结果, 前一节 “U-Boot 的配置” 中提到, 最终把 ARCH, CPU, BOARD 的配置写入了 config.mk 文件, 比如对于 OMAP1610 而言, ARM 为 arm926ejs, 因此

```
OBJS =cpu/arm926ejs/start.o.
```

这也是整个 u-boot 映像的第一个文件。

而\$(LIBS)将会生成 LIBS 定义的所有文件, 具体可参看 U-Boot 顶层目录的 Makefile 文件, 此处不再赘述。

\$(LDSCRIPT)的定义如下:

```
LDSCRIPT := $(TOPDIR)/board/$(BOARD)/u-boot.lds
```

因此根据 config.mk 的定义, 将包含具体目标板所在目录的 u-boot.lds 链接脚本文件。

当这些依赖的文件都生成后或者没有改变时, 将利用以下语句生成 u-boot 映像:

```

u-boot:    depend $(SUBDIRS) $(OBJS) $(LIBS) $(LDSCRIPT)
UNDEF_SYM=`$(OBJDUMP) -x $(LIBS) |
            sed -n -e 's/.*\(__u_boot_cmd_.*\)/-u\1/p|sort|uniq`;

```

```
$(LD) $(LDFLAGS) $$UNDEF_SYM $(OBJS) \  
  --start-group $(LIBS) $(PLATFORM_LIBS) --end-group \  
  -Map u-boot.map -o u-boot
```