U-BOOT配置编译过程

一、主Makefile结构

在深入分析uboot的配置编译过程之前先看一下uboot主Makefile结构，然后为配置编译过程提供有效信息。

1、24-34行，配置uboot版本信息

```
VERSION = 2012
PATCHLEVEL = 10
SUBLEVEL =
EXTRAVERSION =
ifneq "$(SUBLEVEL)" ""
U_BOOT_VERSION = $(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
else
U_BOOT_VERSION = $(VERSION).$(PATCHLEVEL)$(EXTRAVERSION)
endif
TIMESTAMP_FILE = $(obj)include/generated/timestamp_autogenerated.h
VERSION_FILE = $(obj)include/generated/version_autogenerated.h
```

2、36-54行，主机开发环境的架构、操作系统和shell

```
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/')

# Set shell to bash if possible, otherwise fall back to sh
SHELL := $(shell if [ -x "$$BASH" ]; then echo $$BASH; \
    else if [ -x /bin/bash ]; then echo /bin/bash; \
    else echo sh; fi; fi)

export  HOSTARCH HOSTOS SHELL
```

3、89-104行，配置是否将目标文件编译到指定文件夹或者当前源文件文件夹

```
ifdef O
ifeq ("$(origin O)", "command line")
BUILD_DIR := $(O)
endif
```

```
endif

ifneq ($(BUILD_DIR),)
saved-output := $(BUILD_DIR)

# Attempt to create a output directory.
$(shell [ -d ${BUILD_DIR} ] || mkdir -p ${BUILD_DIR})

# Verify if it was successful.
BUILD_DIR := $(shell cd $(BUILD_DIR) && /bin/pwd)
$(if $(BUILD_DIR),,$(error output directory "$(saved-output)" does not exist))
endif # ifneq ($(BUILD_DIR),)
```

使用方法：make -o=编译目录

4、106-111行，设置目标编译目录、源文件目录、顶层目录、链接目录，导出环境变量

```
OBJTREE      := $(if $(BUILD_DIR),$(BUILD_DIR),$(CURDIR))
SPLTREE      := $(OBJTREE)/spl
SRCTREE      := $(CURDIR)
TOPDIR       := $(SRCTREE)
LNDIR        := $(OBJTREE)
export  TOPDIR SRCTREE OBJTREE SPLTREE

MKCONFIG     := $(SRCTREE)/mkconfig
export MKCONFIG
```

说明：如果编译到指定目录A，则OBJTREE是目录A，SPLTREE是A/spl目录，SRCTREE是uboot根目录，TOPDIR是uboot根目录，LNDIR是指定目录A；如果不编译到指定目录，则编译到源文件所在目录，此时OBJTREE是uboot根目录，SPLTREE是uboot根目录下的spl目录，SRCTREE是uboot根目录，TOPDIR是uboot根目录，LNDIR是uboot根目录。

5、113-114行，配置脚本，后续调用

```
MKCONFIG     := $(SRCTREE)/mkconfig
export MKCONFIG
```

6、124-131行，设置前缀变量obj和src

```
ifneq ($(OBJTREE),$(SRCTREE))
obj := $(OBJTREE)/
src := $(SRCTREE)/
else
obj :=
src :=
endif
export obj src
```

7、162-163行，加载调用include目录下的config.mk文件，里面的信息就是ARCH CPU BOARD VENDOR SOC，它是在配置过程产生的（make XXX_config）

```
include $(obj)include/config.mk
export  ARCH CPU BOARD VENDOR SOC
```

8、加载调用顶层目录的config.mk

```
include $(TOPDIR)/config.mk
```

9、剩下的主要是各种编译规则和编译目标

二、uboot配置过程

配置过程是uboot根目录下的mkconfig文件和参数来决定的。

主要作用：

(1) 创建到平台架构和开发板的符号链接

(2) 创建顶层Makefile包含的config.mk（在根目录的include目录下）

(3) 创建开发板相关的头文件config.h（在根目录的include目录下）

1、主Makefile中的配置入口：

```
%_config::  unconfig
    @$(MKCONFIG) -A $(@:_config=)
```

说明：以make socfpga_cyclone5_config 为例，以上代码展开后为 ./mkconfig -A socfpga_cyclone5。

2、利用boards.cfg重新设置mkconfig参数

```
if [ \( $# -eq 2 \) -a \( "$1" = "-A" \) ] ; then
    # Automatic mode
    line=`egrep -i "^[[:space:]]*${2}[[:space:]]" boards.cfg` || {
        echo "make: *** No rule to make target \`$2_config'.  Stop." >&2
        exit 1
    }

    set ${line}
```

说明：以上最终目的是利用boards.cfg中的socfpga_cyclone5 arm armv7 socfpga_cyclone5 altera socfpga这一行重新将 ./mkconfig -A socfpga_cyclone5 变为 ./mkconfig socfpga_cyclone5 arm armv7 socfpga_cyclone5 altera socfpga。

3、创建符号链接

```
if [ "$SRCTREE" != "$OBJTREE" ] ; then
    mkdir -p ${OBJTREE}/include
    mkdir -p ${OBJTREE}/include2
    cd ${OBJTREE}/include2
    rm -f asm
    ln -s ${SRCTREE}/arch/${arch}/include/asm asm
    LNPREFIX=${SRCTREE}/arch/${arch}/include/asm/
    cd ../include
    mkdir -p asm
else
    cd ./include
#删除上次配置产生的连接文件
    rm -f asm
#建立新的连接文件，asm -> ../arch/arm/include/asm
    ln -s ../arch/${arch}/include/asm asm
fi

#删除/include/asm/arch文件夹
rm -f asm/arch

if [ -z "${soc}" ] ; then
    ln -s ${LNPREFIX}arch-${cpu} asm/arch
else
#创建软连接asm/arch -> arch-socfpga
    ln -s ${LNPREFIX}arch-${soc} asm/arch
fi

if [ "${arch}" = "arm" ] ; then
#删除/include/asm/proc文件夹
    rm -f asm/proc
#创建软连接asm/proc -> proc-armv
    ln -s ${LNPREFIX}proc-armv asm/proc
fi
```

说明：假如uboot根目录是/A，第一个软连接在根目录的include目录下，最终效果是/A/asm指向/A/arch/arm/include/asm，第二个软连接在/A/asm下和/A/arch/arm/include/asm下，最终效果是/A/asm/arch和/A/arch/arm/include/asm/arch都指向/A/arch/arm/include/asm/arch-socfpga，第三个软连接也在/A/asm下和/A/arch/arm/include/asm下，最终效果是/A/asm/proc和/A/arch/arm/include/asm/proc都指向/A/arch/arm/include/asm/proc-armv

4、生成include目录下的config.mk

```
( echo "ARCH   = ${arch}"
    if [ ! -z "$spl_cpu" ] ; then
    echo 'ifeq ($(CONFIG_SPL_BUILD),y)'
    echo "CPU    = ${spl_cpu}"
    echo "else"
    echo "CPU    = ${cpu}"
    echo "endif"
    else
    echo "CPU    = ${cpu}"
    fi
    echo "BOARD  = ${board}"

    [ "${vendor}" ] && echo "VENDOR = ${vendor}"
    [ "${soc}"    ] && echo "SOC    = ${soc}"
    exit 0 ) > config.mk
```

说明：最终在config.mk中生成的信息如下：

ARCH = arm

CPU = armv7

BOARD = socfpga_cyclone5

VENDOR = altera

SOC = socfpga

5、生成include目录下的config.h

```
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

for i in ${TARGETS} ; do
    i="`echo ${i} | sed '/=/ {s/=/  /;q; } ; { s/$/ 1/; }'`"
    echo "#define CONFIG_${i}" >>config.h ;
done

echo "#define CONFIG_SYS_ARCH  \"${arch}\""  >> config.h
echo "#define CONFIG_SYS_CPU   \"${cpu}\""   >> config.h
echo "#define CONFIG_SYS_BOARD \"${board}\"" >> config.h

[ "${vendor}" ] && echo "#define CONFIG_SYS_VENDOR \"${vendor}\"" >> config.h

[ "${soc}"    ] && echo "#define CONFIG_SYS_SOC    \"${soc}\""    >> config.h

cat << EOF >> config.h
#define CONFIG_BOARDDIR board/$BOARDDIR
#include <config_cmd_defaults.h>
#include <config_defaults.h>
#include <configs/${CONFIG_NAME}.h>
#include <asm/config.h>
#include <config_fallbacks.h>
#include <config_uncmd_spl.h>
EOF
```

说明：上述代码主要定义了arch cpu board vendor soc 5个相关的宏输出到头文件config.h中，另外还在config.h中包含了几个头文件，特别要注意的是#include <configs/${CONFIG_NAME}.h>，其中${CONFIG_NAME}由脚本上下文得出是socfpga_cyclone5，因此该包含的头文件是configs/socfpga_cyclone5.h，这个头文件里定义了开发板的配置信息，我们可以通过修改该文件来设置、裁减uboot。

三、uboot编译链接过程

1、先调用配置生成的include目录下的config.mk，导出ARCH CPU BOARD VENDOR SOC，确定交叉编译工具链CROSS_COMPILE，调用顶层config.mk

```
include $(obj)include/config.mk
export  ARCH CPU BOARD VENDOR SOC

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

# load other configuration
include $(TOPDIR)/config.mk
```

说明：include目录下的config.mk在第二大节uboot配置过程中讲清楚了；CROSS_COMPILE通常是命令行传入，比如编译的时候输入make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all，然后CROSS_COMPILE就是arm-linux-gnueabihf-；顶层目录的config.mk做的东西比较多，下面分析config.mk。

2、顶层config.mk

主要工作：

(1)设置交叉编译工具链

(2)设置编译和链接参数

2.1 设置交叉编译工具链

```
AS  = $(CROSS_COMPILE)as

# Always use GNU ld
LD  = $(shell if $(CROSS_COMPILE)ld.bfd -v > /dev/null 2>&1; \
        then echo "$(CROSS_COMPILE)ld.bfd"; else echo "$(CROSS_COMPILE)ld"; fi;)

CC  = $(CROSS_COMPILE)gcc
CPP = $(CC) -E
AR  = $(CROSS_COMPILE)ar
NM  = $(CROSS_COMPILE)nm
LDR = $(CROSS_COMPILE)ldr
STRIP   = $(CROSS_COMPILE)strip
OBJCOPY = $(CROSS_COMPILE)objcopy
OBJDUMP = $(CROSS_COMPILE)objdump
RANLIB  = $(CROSS_COMPILE)RANLIB
DTC = dtc
```

2.2 设置编译和链接参数

```
...

CPPFLAGS += -I$(TOPDIR)/include
```

```
...

LDFLAGS_u-boot += -T $(obj)u-boot.lds $(LDFLAGS_FINAL)
ifneq ($(CONFIG_SYS_TEXT_BASE),)
LDFLAGS_u-boot += -Ttext $(CONFIG_SYS_TEXT_BASE)
endif

LDFLAGS_u-boot-spl += -T $(obj)u-boot-spl.lds $(LDFLAGS_FINAL)
ifneq ($(CONFIG_SPL_TEXT_BASE),)
LDFLAGS_u-boot-spl += -Ttext $(CONFIG_SPL_TEXT_BASE)
endif
```

说明：CPPFLAGS += -I$(TOPDIR)/include说明在源文件中包含$(TOPDIR)/include中的头文件时不需要再加上$(TOPDIR)/include前缀；LDFLAGS_u-boot是连接uboot的连接参数，如果CONFIG_SYS_TEXT_BASE定义了则会增加-Ttext $(CONFIG_SYS_TEXT_BASE)到LDFLAGS_u-boot；LDFLAGS_u-boot-spl是连接spl的连接参数，如果CONFIG_SPL_TEXT_BASE定义了则会增加-Ttext $(CONFIG_SPL_TEXT_BASE)到LDFLAGS_u-boot-spl。

3、从主Makefile编译目标推断编译链接顺序

```
ALL-y += $(obj)u-boot.srec $(obj)u-boot.bin $(obj)System.map
...
ALL-$(CONFIG_SPL) += $(obj)spl/u-boot-spl.bin
...


all:        $(ALL-y) $(SUBDIR_EXAMPLES)
```

最终目标依赖ALL-y，而ALL-y中最重要的是spl的二进制镜像u-boot-spl.bin和uboot的二进制镜像u-boot.bin。

```
$(obj)spl/u-boot-spl.bin:   $(SUBDIR_TOOLS) depend
        $(MAKE) -C spl all
```

先来看编译spl，进入spl目录执行spl的makefile编译all目标，然后会在spl目录下得到u-boot-spl.bin。

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

然后看编译u-boot.bin，它依赖于u-boot，因此需要先编译u-boot。

```
$(obj)u-boot:   depend \
        $(SUBDIR_TOOLS) $(OBJS) $(LIBBOARD) $(LIBS) $(LDSCRIPT) $(obj)u-boot.lds
        $(GEN_UBOOT)
```

编译u-boot也依赖$(OBJS) $(LIBS)等平台/开发板相关的目标文件、通用的目标文件库文件等等。

```
#########################################################################
# U-Boot objects....order is important (i.e. start must be first)

```

```
OBJS  = $(CPUDIR)/start.o
ifeq ($(CPU),x86)
OBJS += $(CPUDIR)/start16.o
OBJS += $(CPUDIR)/resetvec.o
endif
ifeq ($(CPU),ppc4xx)
OBJS += $(CPUDIR)/resetvec.o
endif
ifeq ($(CPU),mpc85xx)
OBJS += $(CPUDIR)/resetvec.o
endif

OBJS := $(addprefix $(obj),$(OBJS))

HAVE_VENDOR_COMMON_LIB = $(if $(wildcard board/$(VENDOR)/common/Makefile),y,n)

LIBS-y += lib/libgeneric.o
LIBS-y += lib/lzma/liblzma.o
LIBS-y += lib/lzo/liblzo.o
LIBS-y += lib/zlib/libz.o

...

$(OBJS):    depend
        $(MAKE) -C $(CPUDIR) $(if $(REMOTE_BUILD),$@,$(notdir $@))

$(LIBS):    depend $(SUBDIR_TOOLS)
        $(MAKE) -C $(dir $(subst $(obj),,$@))
```

以上是一些编译u-boot依赖的目标文件以及对应的编译规则，编译规则都是进入目标文件对应目录执行对应的makefile。

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
 . = 0x00000000;
 . = ALIGN(4);
 .text :
 {
  __image_copy_start = .;
  arch/arm/cpu/armv7/start.o (.text)
  *(.text)
 }
 . = ALIGN(4);
 .rodata : { *(SORT_BY_ALIGNMENT(SORT_BY_NAME(.rodata*))) }
 . = ALIGN(4);
 .data : {
  *(.data)
 }
 . = ALIGN(4);
 . = .;
 __u_boot_cmd_start = .;
 .u_boot_cmd : { *(.u_boot_cmd) }
 __u_boot_cmd_end = .;
 . = ALIGN(4);
 __image_copy_end = .;
 .rel.dyn : {
  __rel_dyn_start = .;
  *(.rel*)
  __rel_dyn_end = .;
 }
 .dynsym : {
  __dynsym_start = .;
  *(.dynsym)
 }
 _end = .;
 . = ALIGN(4096);
 .mmutable : {
  *(.mmutable)
 }
 .bss __rel_dyn_start (OVERLAY) : {
  __bss_start = .;
  *(.bss)
   . = ALIGN(4);
  __bss_end__ = .;
 }
 /DISCARD/ : { *(.dynstr*) }
 /DISCARD/ : { *(.dynamic*) }
 /DISCARD/ : { *(.plt*) }
 /DISCARD/ : { *(.interp*) }
 /DISCARD/ : { *(.gnu*) }
}
```

编译完u-boot依赖的相关目标文件后根据链接参数$(LDSCRIPT)和链接脚本$(obj)u-boot.lds，对这些目标文件进行组装链接，链接参数和链接脚本确定了文件的布局，其中$(LDSCRIPT)最重要的参数是-Ttext $(CONFIG_SYS_TEXT_BASE)在前文中讲到，如果在配置文件中定义CONFIG_SYS_TEXT_BASE则代码段从CONFIG_SYS_TEXT_BASE开始排布，然后接下来的段往后排布，如果没有定义CONFIG_SYS_TEXT_BASE，则按连接脚本中的地址排布，即代码段排布到0x0。从连接脚本中还发现第一个连接的目标文件为arch/arm/cpu/armv7/start.o，因此uboot的第一句执行指令在该文件中。