< 2015年1月 >

日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	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
1	2	3	4	5	6	7

公告

昵称：深海的小鱼儿
园龄：4年
粉丝：112
关注：0
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

ARM(37)
AVR_Proteus(17)
c/c++(24)
chapts(1)
HTML5(1)
javascript(83)
Linux(104)
Linux Driver for Embedded(26)
linux kernel 0.11
Live(30)
Lwip(4)
Multisim10(1)
New os(1)
Products(9)
QT(15)
Redhat开发环境配置(2)
Sqlite3(4)
Uboot(1)
UCOS(4)
汇编

随笔档案

2014年10月 (7)
2014年9月 (5)
2014年8月 (2)
2014年7月 (1)
2014年6月 (15)
2014年5月 (2)
2014年4月 (20)
2014年3月 (9)
2014年1月 (3)
2013年12月 (1)
2013年10月 (1)
2013年9月 (5)
2013年8月 (1)
2012年7月 (2)
2012年6月 (1)
2012年5月 (1)
2012年4月 (10)
2012年3月 (12)
2012年2月 (9)
2012年1月 (3)
2011年12月 (4)
2011年11月 (44)
2011年10月 (23)
2011年9月 (11)
2011年8月 (24)
2011年7月 (20)

u-boot(Makefile)

当我们编译U - BOOT的时候，大家键入make smdk2410_config,make 的时候都作了那些动作呢，这里我先大概介绍一下Makefile的内容，然后在大概理解一下命令执行的流程。如果有错的地方，希望大家指正，谢谢。

1.u-boot顶层目录的Makefile分析:

```
HOSTARCH := $(shell uname -m | \
        sed -e s/i.86/i386/ \
            -e s/sun4u/sparc64/ \
            -e s/arm.*/arm/ \
            -e s/sa110/arm/ \
            -e s/powerpc/ppc/ \
            -e s/macppc/ppc/)
```

首先执行uname -m得到I686,通过管道传送给sed命令,然后sed命令将执行sed -e s/i.86/i386/,将I686替换成i386,最后的结果是HOSTARCH=i386.

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

首先执行uname -s 查看开发平台的系统,结果为Linux,然后通过管道传送给tr命令,tr命令利用字符类[:lower:]和[:upper:]将LInux字符串转化为linux,然后再利用sed命令.最后的结果是HOSTOS=linux

```
export HOSTARCH HOSTOS
```

export 是Makefile的语法关键词,将这些变量传递给下一层的Makefile.总控Makefile的变量可以传递到下级的Makefile中（如果你显示的声明），但是不会覆盖下层的Makefile中所定义的变量，除非指定了“-e”参数。
如果你要传递变量到下级Makefile中，那么你可以使用这样的声明：
    export <variable ...>;
如果你不想让某些变量传递到下级Makefile中，那么你可以这样声明：
    unexport <variable ...>;

```
TOPDIR := $(shell if [ "$$PWD" != "" ]; then echo $$PWD; else pwd; fi)
export TOPDIR
```

得到U-BOOT的绝对路径为TOPDIR.

```
ifeq (,$(findstring s,$(MAKEFLAGS)))
XECHO = echo
  else
  XECHO = :
endif
```

通过findstring函数来找MAKEFLAGS是否有匹配s的关键词，如果没有则ifeq就为真。那么变量XECHO就等于echo 反之亦然。

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

这里主要说明origin的语法：

origin函数不像其它的函数，他并不操作变量的值，他只是告诉你你的这个变量是哪里来的？其语法是：
    $(origin <variable>;)
注意，<variable>;是变量的名字，不应该是引用。所以你最好不要在<variable>;中使用“$”字符。Origin函数会以其返回值来告诉你这个变量的“出生情况”，下面，是origin函数的返回值:

“undefined”
    如果<variable>;从来没有定义过，origin函数返回这个值“undefined”。
“default”
    如果<variable>;是一个默认的定义
“environment”
    如果<variable>;是一个环境变量，并且当Makefile被执行时，“-e”参数没有被打开。
“file”
    如果<variable>;这个变量被定义在Makefile中。
“command line”
    如果<variable>;这个变量是被命令行定义的。
“override”
    如果<variable>;是被override指示符重新定义的。
“automatic”
    如果<variable>;是一个命令运行中的自动化变量。

$(shell [-d ${BUILD_DIR}] || mkdir -p ${BUILD_DIR}) //判断当前是否有个 { BUILD_DIR } 目录，如果没有执行mkdir -p ${BUILD_DIR}，创建 { BUILD_DIR } 目录,这个变量为空。

```
# ifneq ($(BUILD_DIR),)
```

2011年6月 (19)
2011年5月 (24)
2011年4月 (43)
2011年3月 (37)
2011年2月 (4)
2011年1月 (4)


```
OBJTREE     := $(if $(BUILD_DIR),$(BUILD_DIR),$(CURDIR))
  SRCTREE     := $(CURDIR)
  TOPDIR      := $(SRCTREE)
LNDIR       := $(OBJTREE)
export TOPDIR SRCTREE OBJTREE
MKCONFIG    := $(SRCTREE)/mkconfig
export MKCONFIG
//最后 TOPDIR SRCTREE OBJTREE这三个变量一样，都是u-boot源码目录的根目录路径。然后设置MKCONFIG变量，代表一个脚本，这个脚本以后用。
ifneq ($(OBJTREE),$(SRCTREE))
obj := $(OBJTREE)/
src := $(SRCTREE)/
else
obj :=
src :=
endif
export obj src
//由以上可知obj,src都为空

ifeq (include/config.mk,$(wildcard include/config.mk)) //通过wildcard文件名函数判断是否有include/config.mk文件,也就是执行make smdk2410_config以后产生的文件.

$(wildcard pattern)
参数pattern是一个文件名格式，包含有通配符。函数wildcard的结果是一列和格式匹配且真实存在的文件的名称，文件名之间用一个空格隔开。
比如当前目录下有文件1.c,2.c,1.h,2.h 则
c_src := $(wildcard *.c)
结果为：1.c 2.c

# load ARCH, BOARD, and CPU configuration
include $(obj)include/config.mk //包含这个文件.这里obj为空

export ARCH CPU BOARD VENDOR SOC //将include/config.mk里的变量申明给其他的Makefile使用.

# load other configuration
include $(TOPDIR)/config.mk //然后包含根目录的config.mk文件.

这些config.mk将在以后介绍

ifndef CROSS_COMPILE   //确实没有定义CROSS_COMPILE变量
ifeq ($(HOSTARCH),ppc) //HOSTARCH为i386,CROSS_COMPILE所以不为空
CROSS_COMPILE =
else
ifeq ($(ARCH),ppc)
CROSS_COMPILE = powerpc-linux-
endif
ifeq ($(ARCH),arm)
CROSS_COMPILE = arm-linux-
endif
.....

首先没有定义CROSS_COMPILE,然后我们的HOSTARCH=i386,然后在判断ARCH,由于在前面已经指定ARCH=arm.所以CROSS_COMPILE=arm-linux-.通过这个可以选择不同平台下的交叉编译器.

include $(TOPDIR)/config.mk //包含根目录下的config.mk文件，这个文件以后会分析到。

OBJS = cpu/$(CPU)/start.o
ifeq ($(CPU),i386)
OBJS += cpu/$(CPU)/start16.o
OBJS += cpu/$(CPU)/reset.o
endif .....

OBJS := $(addprefix $(obj),$(OBJS)) //将OBJS赋值给OBJ
$(addprefix src/,foo bar)
结果：src/foo src/bar

由于start.S是我们启动代码,所以首先编译.OBJ=cpu/arm920t/start.o

LIBS = lib_generic/libgeneric.a
LIBS += board/$(BOARDDIR)/lib$(BOARD).a
LIBS += cpu/$(CPU)/lib$(CPU).a
ifdef SOC
LIBS += cpu/$(CPU)/$(SOC)/lib$(SOC).a
endif
LIBS += lib_$(ARCH)/lib$(ARCH).a
.....

.PHONY : $(LIBS)

添加相应的静态库.
```

```
__OBJS := $(subst $(obj),,$(OBJS)
__LIBS := $(subst $(obj),,$(LIBS)) $(subst $(obj),,$(LIBBOARD))
```

(1) $(subst from,to,text).
在文本“text”中使用to替换每一处的from。
比如：
$(subst ee,EE,feet on the street)
结果为：fEET on the strEET
ALL += $(obj)u-boot.srec $(obj)u-boot.bin $(obj)System.map $(U_BOOT_NAND) $(U_BOOT_ONENAND) //这个是最后要生成的文件。$(U_BOOT_NAND) $(U_BOOT_ONENAND) 要添加相应的宏定义即可。

```
$(obj)u-boot.hex:   $(obj)u-boot
        $(OBJCOPY) ${OBJCFLAGS} -O ihex $< $@ 将u-boot ELF格式文件生成16进制格式的文件
$(obj)u-boot.srec: $(obj)u-boot
        $(OBJCOPY) ${OBJCFLAGS} -O srec $< $@ 将u-bootELF格式文件生成另一种S-Record格式的文件
unconfig:
    @rm -f $(obj)include/config.h $(obj)include/config.mk \
        $(obj)board/*/config.tmp $(obj)board/*/*/config.tmp \                   $(obj)include/autoconf.mk $(obj)include/autoconf.mk.dep
//删除以前的配置文件
```

以上是一些Makefile的大概信息，这里就说到这里。感兴趣的可以再深入了解。

```
//当我们执行make smdk2410_config的时候，要作的事情如下：
```

Makefile文件里面可以看出支持好多种体系结构，并有相应开发板的配置信息。这里主要研究的是ARM，开发板是smdk2410.
当我们执行：make smdk2410_config的时候，首先执行：

```
smdk2410_config :   unconfig
    @$(MKCONFIG) $(@:_config=) arm arm920t smdk2410 NULL s3c24x0
```

可以看出。现执行unconfig这个标签，以上可以看出主要是删除以前的配置信息。
然后执行$(MKCONFIG),也就是mkconfig脚本，并传递6个参数。
$(@:_config=)他的作用就是将smdk2410_config中的_config设置为空，结果为smdk2410.
这个命令也就是：./mkconfig smdk2410 arm arm920t smdk2410 NULL s3c24x0.
接下来看看mkconfig的源代码：
1.确定开发板的名称

```
APPEND=no   # Default: Create new config file
BOARD_NAME=""   # Name to print in make output
while [ $# -gt 0 ] ; do
    case "$1" in
    --) shift ; break ;;
    -a) shift ; APPEND=yes ;;
    -n) shift ; BOARD_NAME="${1%%_config}" ; shift ;;
    *) break ;;
    esac
done
```

由于参数里没有-- -a -n等参数，所以这个while没有执行。然后APPEND BOARD_NAME没有改变。

```
[ "${BOARD_NAME}" ] || BOARD_NAME="$1" //这个时候BOARD_NAME的值就等于"smdk2410".
[ $# -lt 4 ] && exit 1 //参数的个数小于4退出
[ $# -gt 6 ] && exit 1//参数的个数大于6退出
```

2.创建开发板相关的头文件的连接

```
//判断源代码目录和目标文件目录是否一样，由于直接我们都是在源代码目录编译，所以将执行else分之的代码。
if [ "$SRCTREE" != "$OBJTREE" ] ; then
    mkdir -p ${OBJTREE}/include
    mkdir -p ${OBJTREE}/include2
    cd ${OBJTREE}/include2
    rm -f asm
    ln -s ${SRCTREE}/include/asm-$2 asm
    LNPREFIX="../../include2/asm/"
    cd ../include
    rm -rf asm-$2
    rm -f asm
    mkdir asm-$2
    ln -s asm-$2 asm
else
    cd ./include
    rm -f asm
    ln -s asm-$2 asm
fi
```

//进入include目录，删除asm文件（这是上一次的配置时建立的连接文件），然后再次建立asm文件，并令它连接向asm-$2目录，也就是asm-arm目录。

```
rm -f asm-$2/arch //删除asm-$2即asm-arm目录
if [ -z "$6" -o "$6" = "NULL" ] ; then //-z表示：[ -z STRING ] “STRING” 的长度为零则为真。
    ln -s ${LNPREFIX}arch-$3 asm-$2/arch
else
    ln -s ${LNPREFIX}arch-$6 asm-$2/arch
fi
```

//对于$6就是s3c24x0,不为空，也不是NULL，所以将执行else分之。LNPREFIX为空，所以连接的命令就是ln -s arch-$6 asm-$2/arch,也就是ln -s arch-s3c24x0 asm-arm/arch

```
if [ "$2" = "arm" ] ; then
```

```
    rm -f asm-$2/proc
    ln -s ${LNPREFIX}proc-armv asm-$2/proc
fi
```

重新建立asm-arm/proc文件，并让它连接向proc-armv目录。

3.创建顶层Makefile包含的文件include/config.mk

echo "ARCH   = $2" > config.mk //“>”，“>>”如果有config.mk文件，并将ARCH输入到config.mk文件里。如果没有首先创建然后将ARCH输入。

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

//将ARCH，CPU，BOARD变量重定向到include/config.mk文件里

```
[ "$5" ] && [ "$5" != "NULL" ] && echo "VENDOR = $5" >> config.mk
[ "$6" ] && [ "$6" != "NULL" ] && echo "SOC    = $6" >> config.mk
```

//将VENDOR，SOC变量重定向到include/config.mk文件里

这样include/config.mk文件里的内容如下：

```
ARCH   = arm
CPU    = arm920t
BOARD = smdk2410
SOC    = s3c24x0
#
# Create board specific header file
#
if [ "$APPEND" = "yes" ]    # Append to existing config file
then
    echo >> config.h
else
    > config.h      # Create new config file //创建include/config.h文件
fi
echo "/* Automatically generated - do not edit */" >>config.h
echo "#include <configs/$1.h>" >>config.h //将#include <configs/$1.h重定向到include/config.h文件里。
exit 0
```

这样include/config.h里的内容如下：

```
/* Automatically generated - do not edit */
#include <configs/smdk2410.h>
```

3.u-boot的编译和连接过程

首先在Makefile里包含了include/config.mk和根目录的config.mk两个文件。第一个主要是那6个参数。第二个config.mk文件的内容如下：

```
BOARDDIR = $(BOARD)
endif
ifdef   BOARD
sinclude $(TOPDIR)/board/$(BOARDDIR)/config.mk # include board specific rules
endif //包含board/smdk2410/config.mk，里面主要定义了TEXT_BASE=0x33f80000
.....
LDSCRIPT := $(TOPDIR)/board/$(BOARDDIR)/u-boot.lds
。。。。
```

LDFLAGS += -Bstatic -T $(LDSCRIPT) $(PLATFORM_LDFLAGS)//加入连接文件为以后使用。LDFLAGS有“-T board/smdk2410/u-boot.lds -Ttext 0x33f80000”字样。首先我们的u-boot.lds告诉我们的代码的分布状况，而 -Ttext 0x33f80000 告诉我们text段放在0x33f80000.待会会讲到u-boot.lds的内容。对于OBJS，LIBS的每个成员，都将进入相应的子目录执行make命令。当所有的OBJS，LIBS所表示的.o,.a文件生成后，就剩下最后的连接了，这对应Makefile的如下几行：

```
$(obj)u-boot.srec: $(obj)u-boot
        $(OBJCOPY) ${OBJCFLAGS} -O srec $< $@
$(obj)u-boot.bin:   $(obj)u-boot
        $(OBJCOPY) ${OBJCFLAGS} -O binary $< $@
$(obj)u-boot:       depend $(SUBDIRS) $(OBJS) $(LIBBOARD) $(LIBS) $(LDSCRIPT)
        UNDEF_SYM=`$(OBJDUMP) -x $(LIBBOARD) $(LIBS) | \
        sed -n -e 's/.*\($(SYM_PREFIX)__u_boot_cmd_.*\)/-u\1/p'|sort|uniq`;\
        cd $(LNDIR) && $(LD) $(LDFLAGS) $$UNDEF_SYM $(__OBJS) \
            --start-group $(__LIBS) --end-group $(PLATFORM_LIBS) \
            -Map u-boot.map -o u-boot
```

首先使用下面的语句连接得到ELF格式的u-boot.最后转化为二进制格式的u-boot.bin,S-Record格式的u-boot.srec。LDFLAGS确定了连接的方式，其中“-T board/smdk2410/u-boot.lds -Ttext 0x33f80000”字样指定了程序的布局和地址。u-boot.lds的文件如下：

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
/*OUTPUT_FORMAT("elf32-arm", "elf32-arm", "elf32-arm")*/
/*指定输出可执行文件是elf格式,32位ARM指令,小端*/
OUTPUT_ARCH(arm)
/*指定输出可执行文件的平台为ARM*/
ENTRY(_start)
/*指定输出可执行文件的起始代码段为_start*/
(.globl _start _start: b       start_code//cpu/arm920t/start.S)
SECTIONS
{
/*指定可执行image文件的全局入口点，通常这个地址都放在ROM(flash)0x0位置。必须使编译器知道这个地址，通常都是修改此处来完成*/
    . = 0x00000000; /*;从0x0位置开始*/
```

```
  . = ALIGN(4);/*代码以4字节对齐*/
  .text      :
  {
    cpu/arm920t/start.o   (.text) /*代码的第一个代码部分*/
    *(.text) /*其它代码部分*/
  }
  . = ALIGN(4);
  .rodata : { *(.rodata) } /*指定只读数据段*/
  . = ALIGN(4);
  .data : { *(.data) }/*指定读/写数据段*/
  . = ALIGN(4);
  .got : { *(.got) } /*指定got段, got段是uboot自定义的一个段, 非标准段*/
  . = .;
                        /*把__u_boot_cmd_start赋值为当前位置, 即起始位置*/
  __u_boot_cmd_start = .;
                        /*指定u_boot_cmd段, uboot把所有的uboot命令放在该段.*/
  .u_boot_cmd : { *(.u_boot_cmd) }
                        /*把__u_boot_cmd_end赋值为当前位置,即结束位置*/
  __u_boot_cmd_end = .;
  . = ALIGN(4);
  __bss_start = .;    /*把__bss_start赋值为当前位置,即bss段的开始位置*/
  .bss (NOLOAD) : { *(.bss) } /*指定bss段,告诉加载器不要加载这个段*/
  _end = .;                 /*把_end赋值为当前位置,即bss段的结束位置*/
}
```

这样代码的都是以0x33f80000+0x0为基准开始，如果你从nandflash启动，测试前4K的代码的地址都是在0x0，那么4K的代码的实现可以通过位置无关指令b来实现。b指令的程序不依赖代码存储的位置 - 即不管这条代码放在什么位置，B指令都可以跳转到正确的位置。
bootloader,内核等程序刚开始运行时。他们所处的地址通常不等于运行地址，在程序的开头，先使用b,bl.mov等位置无关的指令将代码从flash等设备中复制到内存的运行地址处，然后跳转到运行地址去执行。

分类: Linux Driver for Embedded

« 上一篇：U-Boot启动过程完全分析(转)
» 下一篇：U-Boot Makefile文件分析

posted on 2012-04-02 13:34 深海的小鱼儿 阅读(1365) 评论(0) 编辑 收藏