**配置过程**

#CURDIR 当前目录，这个就是源码的要目录下

SRCTREE := $(CURDIR)

MKCONFIG := $(SRCTREE)/mkconfig

100ask24x0\_config : unconfig

@$(MKCONFIG) $(@:\_config=) arm arm920t 100ask24x0 NULL s3c24x0

$(MKCONFIG)是使用变量(Makefile中的变量相当于C语言中的宏)，最前面的@表示在make时不输出make的信息（类似windows下的echo off），(@:\_config=) 是将\_config替换为空，即：

100ask24x0\_config : unconfig

$(SRCTREE)/mkconfig 100ask24x0 arm arm920t 100ask24x0 NULL s3c24x0

$0(参数0) $1 $2 $3 $4 $5 $6

在根目录下可以查找到mkconfig



图 1

打开mkconfig的文件

APPEND**=**no # Default: Create new config file

BOARD\_NAME**=**"" # Name to print in make output

# $#表示参数的个数，如果参数个数超过0，则执行while中的语句

**while** **[** **$#** **-**gt 0 **]** **;** **do**

**case** "$1" **in**

**--)** **shift** **;** **break** **;;**

-a**)** **shift** **;** APPEND**=**yes **;;**

-n**)** **shift** **;** BOARD\_NAME**=**"${1%%\_config}" **;** **shift** **;;**

**\*)** **break** **;;**

**esac**

**done**

# BOARD\_NAME为空，则执行BOARD\_NAME = 100ask24x0

**[** "${BOARD\_NAME}" **]** **||** BOARD\_NAME**=**"$1"

#参数个数小于4则执行exit 1

#$#表示参数的个数

**[** **$#** **-**lt 4 **]** **&&** **exit** 1

#参数个数超过6则执行exit 1

**[** **$#** **-**gt 6 **]** **&&** **exit** 1

#打印Configuring for arm board…

**echo** "Configuring for ${BOARD\_NAME} board..."

#

# Create link to architecture specific headers

#

**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

#建立软链接，将asm-arm 链接为asm

**ln** -s asm-**$2** asm

**fi**

#删除asm-arm/arch

**rm** -f asm-**$2/**arch

#第6个参数为空或者等于NULL就执行then中的分支

**if** **[** **-z** "$6" **-o** "$6" **=** "NULL" **]** **;** **then**

**ln** -s ${LNPREFIX}arch-**$3** asm-**$2/**arch

**else**

#没有定义LNPREFIX则为空，相当于ln –s arch-s3c24x0 asm-arm/arch 如图2

**ln** -s ${LNPREFIX}arch-**$6** asm-**$2/**arch

**fi**

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

**rm** -f asm-**$2/**proc

**ln** -s ${LNPREFIX}proc-armv asm-**$2/**proc

**fi**

#

# Create include file for Make

# 创建config.mk，config.mk中的内容为:

# ARCH = arm

# CPU = arm920t

# BOARD = 100ask24x0

# SOC = s3c24x0

# 如图 3

**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

**exit** 0



图 2

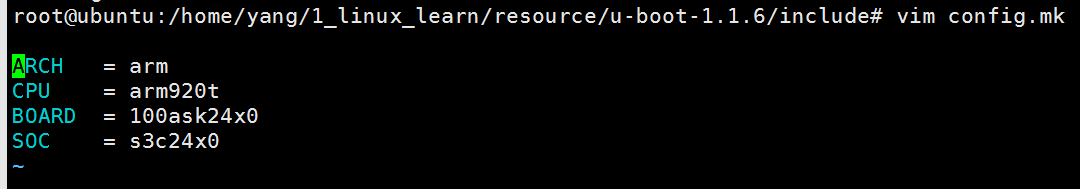


图 3

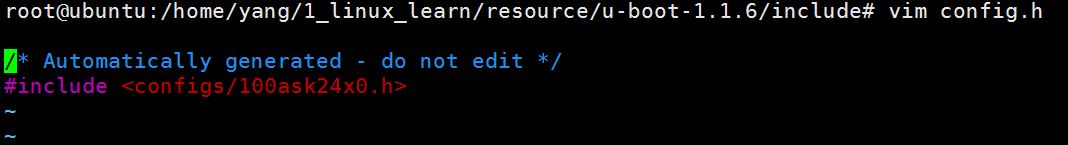


图 4

Makefile中

#$(if CONDITION,THEN-PART[,ELSE-PART]),如果CONDITION变量不为空，则将THEN-PART的作为返回值，否则将ELSE-PART的作为返回值。当条件CONDITION为空，且ELSE-PART为空时，则返回的变量为空。

OBJTREE **:=** $(if $(BUILD\_DIR),$(BUILD\_DIR),$(CURDIR))

SRCTREE **:=** $(CURDIR)

TOPDIR **:=** $(SRCTREE)

LNDIR **:=** $(OBJTREE)

**编译过程**

Makefile中

# load ARCH, BOARD, and CPU configuration

include $(OBJTREE)/include/config.mk

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

即：OBJS = cpu/arm920t/start.o

ALL = $(obj)u-boot.srec $(obj)u-boot.bin $(obj)System.map $(U\_BOOT\_NAND)

all: $(ALL)

在Makefile中如果不指定，则按照第1个指令生成，all 依赖ALL

$(obj)u-boot.bin: $(obj)u-boot

$(OBJCOPY) ${OBJCFLAGS} -O binary $< $@

要的是u-boot.bin 则b-boot.bin 依赖u-boot(这是个elf文件)

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

UNDEF\_SYM=`$(OBJDUMP) -x $(LIBS) |sed -n -e 's/.\*\(\_\_u\_boot\_cmd\_.\*\)/-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

可以一一展开

UNDEF\_SYM=`arm-linux-objdump -x lib\_generic/libgeneric.a board/100ask24x0/lib100ask24x0.a cpu/arm920t/libarm920t.a cpu/arm920t/s3c24x0/libs3c24x0.a lib\_arm/libarm.a fs/cramfs/libcramfs.a fs/fat/libfat.a fs/fdos/libfdos.a fs/jffs2/libjffs2.a fs/reiserfs/libreiserfs.a fs/ext2/libext2fs.a net/libnet.a disk/libdisk.a rtc/librtc.a dtt/libdtt.a drivers/libdrivers.a drivers/nand/libnand.a drivers/nand\_legacy/libnand\_legacy.a drivers/usb/libusb.a drivers/sk98lin/libsk98lin.a common/libcommon.a |sed -n -e 's/.\*\(\_\_u\_boot\_cmd\_.\*\)/-u\1/p'|sort|uniq`;\

cd /home/yang/1\_linux\_learn/resource/u-boot-1.1.6 &&

arm-linux-ld -Bstatic -T /home/yang/1\_linux\_learn/resource/u-boot-1.1.6/board/100ask24x0/u-boot.lds -Ttext 0x33F80000 $UNDEF\_SYM cpu/arm920t/start.o \

--start-group lib\_generic/libgeneric.a board/100ask24x0/lib100ask24x0.a cpu/arm920t/libarm920t.a cpu/arm920t/s3c24x0/libs3c24x0.a lib\_arm/libarm.a fs/cramfs/libcramfs.a fs/fat/libfat.a fs/fdos/libfdos.a fs/jffs2/libjffs2.a fs/reiserfs/libreiserfs.a fs/ext2/libext2fs.a net/libnet.a disk/libdisk.a rtc/librtc.a dtt/libdtt.a drivers/libdrivers.a drivers/nand/libnand.a drivers/nand\_legacy/libnand\_legacy.a drivers/usb/libusb.a drivers/sk98lin/libsk98lin.a common/libcommon.a --end-group \

-Map u-boot.map -o u-boot

原材料是.o和.a文件，看谁放在最前面，就看链接脚本

OUTPUT\_ARCH(arm)

ENTRY(\_start)

SECTIONS

{

//这个0会加上命令行中传入的0x33F80000,也就是说从0x33F80000开始排放代码

. = 0x00000000;

. = ALIGN(4);

.text :

{

//最先排的代码是start.o的代码段，接下来是boot\_init.o的代码段，最后是其他的代码段。Start.o是最先开始运行的代码，分析uboot就是从start.S文件入手，从头读到尾

cpu/arm920t/start.o (.text)

board/100ask24x0/boot\_init.o (.text)

\*(.text)

}

. = ALIGN(4);

.rodata : { \*(.rodata) }

. = ALIGN(4);

.data : { \*(.data) }

. = ALIGN(4);

.got : { \*(.got) }

. = .;

\_\_u\_boot\_cmd\_start = .;

// 这个是uboot自己定义的段

.u\_boot\_cmd : { \*(.u\_boot\_cmd) }

\_\_u\_boot\_cmd\_end = .;

. = ALIGN(4);

\_\_bss\_start = .;

.bss : { \*(.bss) }

\_end = .;

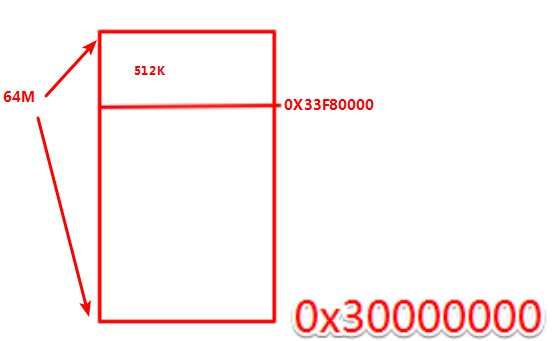
}

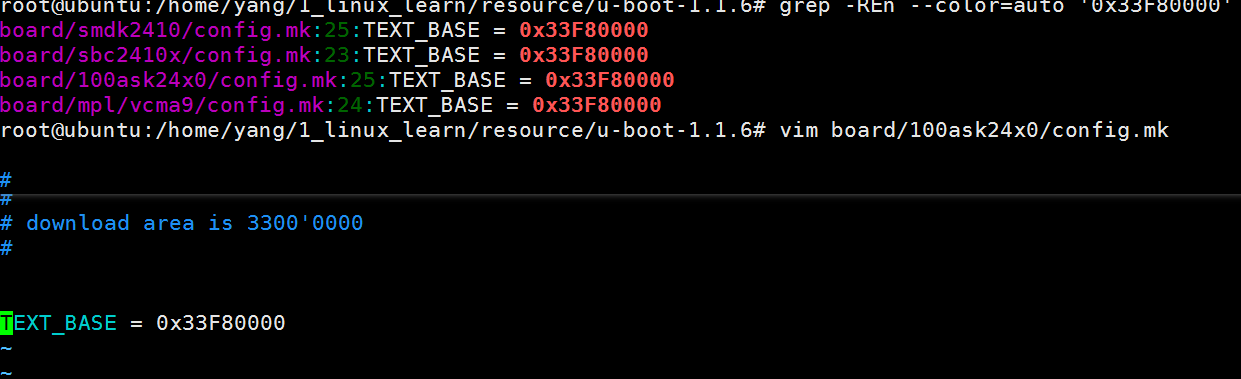
分析Makefile可以道：

1、第1个文件cpu/arm920t/start.S

2、链接地址 board/100ask24x0/u-boot.lds -Ttext 0x33F80000

如果想将uboot放在另一个地址，只需要修改TEXT\_BASE的值就OK了。根据uboot的大小，或者内存的大小作相应的修改。





顶层目录下的config.mk中定义了

LDFLAGS += -Bstatic -T $(LDSCRIPT) -Ttext $(TEXT\_BASE) $(PLATFORM\_LDFLAGS)

**根据start.S分析uboot的功能**

硬件实验

1. 初始化

关闭看门狗

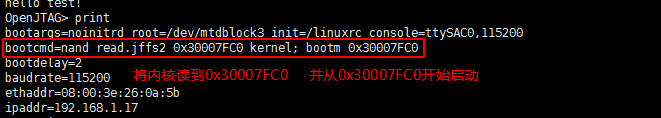
初始化时钟

初始化SDRAM

2. 把程序从Nand拷贝到SDRAM

3. 设置SP

#define Struct\_Section \_\_attribute\_\_ ((unused,section (".u\_boot\_cmd")))



bootargs=noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0,115200

从NAND读出内核：从哪里读，从kernel分区

放到哪里去？----0x30007FC0

bootcmd=nand read.jffs2 0x30007FC0 kernel; bootm 0x30007FC0

在100ask24x0.h中有配制分区



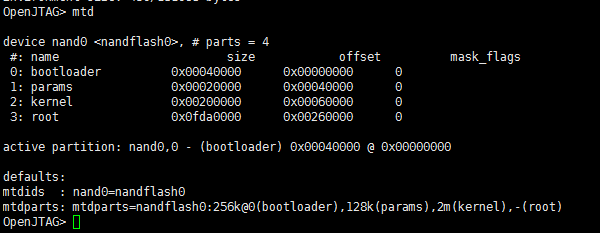
#define MTDPARTS\_DEFAULT "mtdparts=nandflash0:256k@0(bootloader)," \

"128k(params)," \

"2m(kernel)," \

"-(root)"

在nandflash的0地址开始256K是bootloader分区，接下来的128K是params分区，接下来的2M是kernel分区，剩下的是root分区。



bootcmd=nand read.jffs2 0x30007FC0 kernel的效果和

bootcmd=nand read.jffs2 0x30007FC0 0x00060000 0x00200000