基础4-U-boot根目录下的mkconfig分析

此文件位于uboot源码的根目录下，是源码自带的shell脚本文件，主要功能是创建符号链接以及一些头文件（拥有符号链接的功能）。
输入make x210_sd_config时，本脚本将会被主Makefile调用执行。
其内容主要结构为：
1.解析输入参数
2.创建符号链接及一些头文件
需要注意的是，结构顺序不代表代码执行顺序，关于代码的执行顺序以及推荐阅读顺序请移步 [U-boot配置及编译阶段流程宏观分析]

1.解析输入参数

这个脚本在主makefile中被执行前，会被传入6个参数，分别是：x210_sd，arm，s5pc11x，x210，samsung，s5pc110。所以和参数有关的变量为$#=6（$#的值是输入参数个数），$1=x210_sd $2=arm $3=s5pc11x $4=x210 $5=samsung $6=s5pc110

- 本段代码的功能为解析传入本脚本的参数，首先利用了一个while循环，判断$#的值（即参数个数），是否大于0，如果大于0则进入循环
- shell的switch case语法中是不需要break的，故此处的switch case是为了跳出外面的while循环
- 此外每个case的结束都需要加';'执行语句结束也要加';'，所以在每行case的最后都会有两个分号
- 这段代码上来先判断$1的值（即第一个参数），如果为一些特定的值，那么会进行shift操作，即向左移动参数列表一次，将第一个参数移出参数列表
- 但由于我们第一个参数为x210_sd，故只符合最后一个case即*，也就是通配符，直接break跳出了while循环。其实这整段代码对我们没有产生什么作用。

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

```
done
```

- 本句是缩写的if判断语句，判断变量BOARD_NAME是否不为空，由于在12行处对其赋为空值，所以此处将它的值赋为第一个参数的值，即x210_sd

```
[ "${BOARD_NAME}" ] || BOARD_NAME="$1"
```

- 开头两句是缩写的if判断语句，如果参数个数小于4，则这个脚本程序将退出，并返回1（表示出错）；同样，如果参数个数大于6，则这个脚本程序将退出，并返回1（表示出错）
- 最后一句为打印配置信息

```
[ $# -lt 4 ] && exit1
[ $# -gt 6 ] && exit1echo"Configuring for ${BOARD_NAME} board..."
```

2.创建符号链接及一些头文件

从这里开始的代码便是创建符号链接及一些头文件，其根本目的是使uboot具有可移植性。uboot中有很多功能平行的代码文件，各自属于各自不同的平台/开发板/cpu。通过传入此shell脚本的参数，来创建指向我们需要的文件的符号链接。但uboot并不直接与这些功能平行的代码建立联系，符号链接帮助uboot屏蔽了许多用不到的代码。比如某源文件中包含了一个 #include “asm/xx.h”，其中asm为本脚本文件创建的符号链接，它指向的地址为根目录下include/asm-arm，由此便可以定位到真正的路径，即”根目录下include/asm-arm/xx.h”。综上所述，创建符号链接的本质是通过不同的参数，令源代码文件包含特定的文件

- 本段代码功能是确定和汇编指令集相关文件夹
- 首先判断是否使用了“外部输出文件夹编译”功能，如果未使用则进入源码目录下的include目录，删除原本存在的符号链接asm
- 最后创建符号链接asm，使其链接到asm-arm文件夹

```
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
elsecd ./include
    rm -f asm
    ln -s asm-$2 asm
fi
```

- 删除指令集文件夹下原本存在的符号链接arch
- 然后确定指令集文件夹下的架构文件夹，但是这段代码其实是历史遗留代码，创建的是有问题的，真正的确定架构文件夹是在后面

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

if [ -z "$6" -o "$6" = "NULL" ] ; then
    ln -s ${LNPREFIX}arch-$3 asm-$2/arch
else
    ln -s ${LNPREFIX}arch-$6 asm-$2/arch
fi
```

- 本段代码旨在确定cpu的寄存器头文件，和确定指令集文件夹下的架构文件夹
- 首先判断第三个参数是不是s5pc11x，然后删除原本存在的符号链接regs.h，再创建regs.h链接到s5pc110.h
- 最后删除原本./include/asm-arm目录下的存在的符号链接arch，再在./include/asm-arm目录下创建arch链接到arch-s5pc110这个架构文件夹

```
这里省略了一堆和其他soc有关的代码，只保留了我们的s5pv210
if [ "$3" = "s5pc11x" ] ; then
        rm -f regs.h
        ln -s $6.h regs.h
        rm -f asm-$2/arch
        ln -s arch-$3 asm-$2/arch
fi
```

- 本段代码旨在确定指令集文件夹下的proc文件夹。在./include/asm-arm目录下创建proc，指向proc-armv文件夹

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

- 本段是创建include/config.mk，由于之前所停留的目录一直是include目录下，故config.mk将被创建在include目录下
- bash语法中的>功能是创建文件并填充内容，>>功能是额外添加内容，所以第2,3,4个参数都被添加到这个文件中去了，当5，6存在且不为0时，第5,6个参数才被添加

```
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
```

- 本段的功能主要是创建根目录下include/config.h这个文件，这个.h文件内只有一句话，即#include “configs/x210_sd.h”
- 从某种意义上来说，./include/config.h这个文件的功能就是符号链接，它指向了真正的配置头文件./include/configs/x210_sd.h
- APPEND变量是在本文件一开始的地方定义的，值默认是no，当本脚本文件被传参数-a时，它会被赋值yes（具体代码大概在25行左右）
- 创建完config.h之后，再向其填充入内容，最后shell结束，返回值0（表示正常）

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

exit 0
```