U-boot 的配置编译的命令：

（1）.make　unsp210_config　　//源码筛选的过程（取其精华去其糟粕）

（2）.make　　//编译的过程，最终生成 u-boot.bin

注：这些命令都是在虚拟机的 linux 命令行的 u-boot 顶级目录下执行的。（如下图 0）所示：

```
huibo@sz:~/hbb/boot/unsp210_u-boot$ pwd
/home/huibo/hbb/boot/unsp210_u-boot
huibo@sz:~/hbb/boot/unsp210_u-boot$ make unsp210_config
```

U-boot 的配置过程：

（1）.**目的**：寻找配置命令的目标：unsp210_config

路径+文件：TOPDIR/Makefile 的 2581 行（如下图 1）：

```
2580
2581 unsp210_config :    unconfig
2582     @$(MKCONFIG) $(@:_config=) arm s5pc11x unsp210 samsung s5pc110
2583     @echo "TEXT_BASE = 0xc3e00000" > $(obj)board/samsung/unsp210/config.mk
```

（2）.**目的**：寻找配置命令的目标的依赖：unconfig

路径+文件：TOPDIR/Makefile 的 473 行（如下图 2）

```
473 unconfig:
474     @rm -f $(obj)include/config.h $(obj)include/config.mk \
475         $(obj)board/*/config.tmp $(obj)board/*/*/config.tmp \
476         $(obj)include/autoconf.mk $(obj)include/autoconf.mk.dep \
477         $(obj)board/$(VENDOR)/$(BOARD)/config.mk
478
```

功能：删除之前的一些旧的配置信息。

（3）.**目的**：解析上图 1 中执行的第一个命令

路径+文件：TOPDIR/Makefile　2582 行；给脚本传递六个参数。（如下图 3）

```
@$(MKCONFIG)  1
topdir/mkconfig

$(@:_config=)  2
      $1

arm   s5pc11x   unsp210   samsung   s5pc110
 $2     $3        $4        $5        $6
```

红色方框 1 部分内容作用是：

通过 Makefile 变量“MKCONFIG”指定执行脚本：TOPDIR/mkconfig

文件+路径：TOPDIR/Makefile　101 行

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

并且可以继续追踪 $(SRCTREE) 就是 TOPDIR。

红色方框 2 部分内容作用是：

确定第 1 个参数$1

$(@:_config=)　　中冒号作用是过滤，把 unsp210_config 中的_config 过滤掉，得到结果是　unsp210。

最终上图 3 整理后得到的结果是：（下图 4）

@$(MKCONFIG)	$(@:_config=)	arm	s5pc11x	unsp210	samsung	s5pc110
topdir/mkconfig	$1	$2	$3	$4	$5	$6
mkconfig	unsp210	arm	s5pc11x	unsp210	samsung	s5pc110

功能：主要完成了一下 4 个功能：

A：打印一句话：（下图 5）

```
huibo@sz:~/hbb/boot/unsp210_u-boot$ make unsp210_config
Configuring for unsp210 board...
huibo@sz:~/hbb/boot/unsp210_u-boot$
```

路径+文件：TOPDIR/mkconfig　脚本的 23～28 行实现。（下图 6）

```
33
34 [ "${BOARD_NAME}" ] || BOARD_NAME="$1"
35
36 [ $# -lt 4 ] && exit 1
37 [ $# -gt 6 ] && exit 1
38
39 echo "Configuring for ${BOARD_NAME} board..."
```

B：创建新的链接文件：TOPDIR/include/asm

路径+文件：TOPDIR/mkconfig　脚本的 33～49 行实现。（下图 7）

```
41 #
42 # Create link to architecture specific headers
43 #
44 if [ "$SRCTREE" != "$OBJTREE" ] ; then
45     mkdir -p ${OBJTREE}/include
46     mkdir -p ${OBJTREE}/include2
47     cd ${OBJTREE}/include2
48     rm -f asm
49     ln -s ${SRCTREE}/include/asm-$2 asm
50     LNPREFIX="../../include2/asm/"
51     cd ../include
52     rm -rf asm-$2
53     rm -f asm
54     mkdir asm-$2
55     ln -s asm-$2 asm
56 else
57     cd ./include
58     rm -f asm
59     ln -s asm-$2 asm
60 fi
```

C：创建一个配置脚本：TOPDIR/include/config.mk

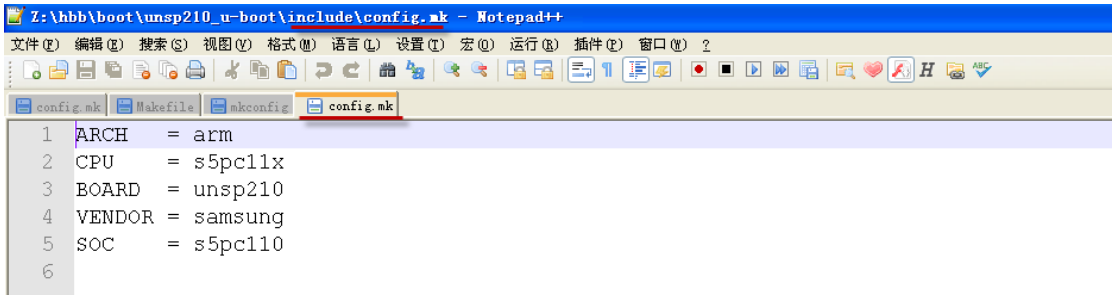
（三个 config.mk 中第 1 个）

路径+文件：TOPDIR/mkconfig　脚本的 121～129 行实现。（下图 8）

```
131 #
132 # Create include file for Make
133 #
134 echo "ARCH   = $2" >  config.mk
135 echo "CPU    = $3" >> config.mk
136 echo "BOARD  = $4" >> config.mk
137
138 [ "$5" ] && [ "$5" != "NULL" ] && echo "VENDOR = $5" >> config.mk
139
140 [ "$6" ] && [ "$6" != "NULL" ] && echo "SOC    = $6" >> config.mk
141
```

结果是如下图（下图 9）

路径：**TOPDIR/include/config.mk**



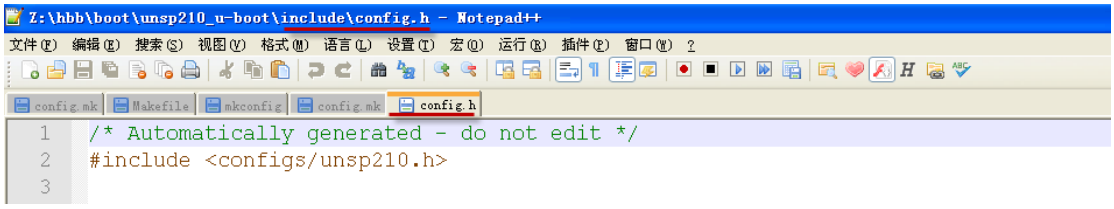
D：创建一个公共头文件：TOPDIR/include/config.h

路径+文件：TOPDIR/mkconfig　脚本的 142～152 行实现。（下图 10）

```
142 #
143 # Create board specific header file
144 #
145 if [ "$APPEND" = "yes" ]    # Append to existing config file
146 then
147     echo >> config.h
148 else
149     > config.h      # Create new config file
150 fi
151 echo "/* Automatically generated - do not edit */" >>config.h
152 echo "#include <configs/$1.h>" >>config.h
```

结果是（下图 11）

路径：**TOPDIR/include/config.h**



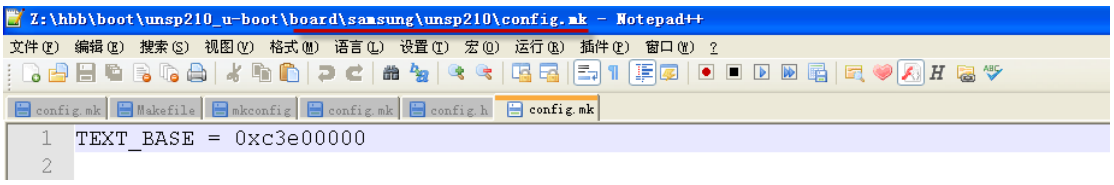
（4）.**目的**：解析上图 **1** 中执行的第二个命令

路径+文件：TOPDIR/Makefile　2583 行实现（如下图 **12**）

```
2583     @echo "TEXT_BASE = 0xc3e00000" > $(obj)board/samsung/unsp210/config.mk
```

结果是（下图 **13**）

路径：**TOPDIR/board/samsung/unsp210/config.mk**　**(**第 2 个 config.mk**)**



配置过程的总结：

（1）.首先是通过向 mkconfig 这个脚本传递 6 个参数实现对 u-boot 的配置。

（2）.上面只是介绍了执行脚本的时候实现的 4 个功能，并没有介绍为什么要这样做；

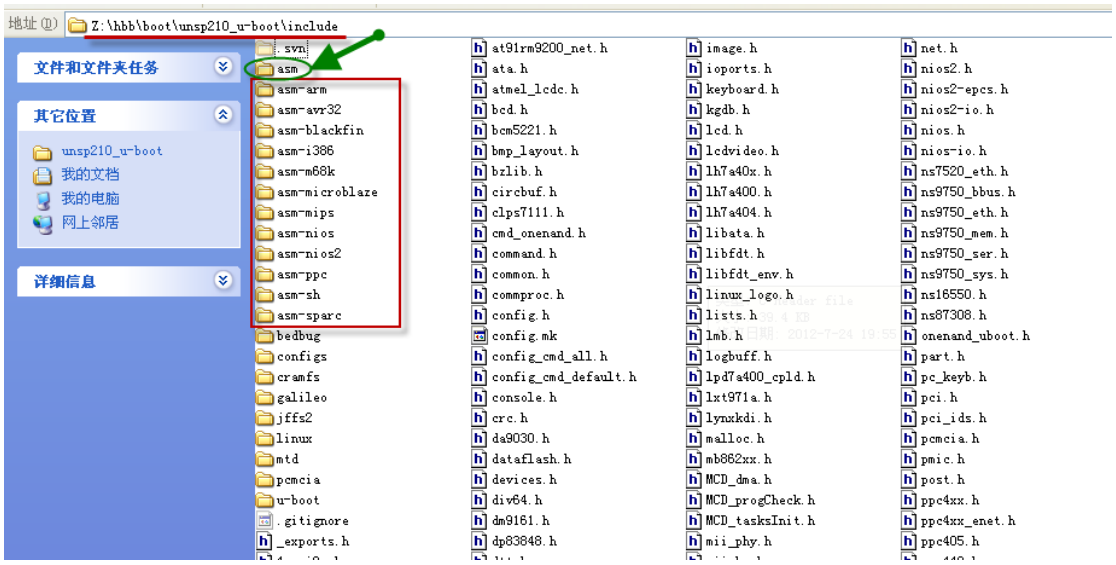
下面将一一介绍为什么要做这四件事情：

A：为什么要打印 Configuring for unsp210 board...这句话？

提示用户 u-boot 已经编译完成了。

B：为什么创建一个新的链接？

可以回到上图 7 查看实现过程，（下图 14）是实现结果。



如果绿色箭头指示的内容，是刚开始删除后有新建的链接文件，最终实现的结果就是把红色方框中的第一个文件夹的内容“asm-arm”拷贝到“asm”中，其实就是实现了一个筛选的过程，也就是把 arm 相关的头文件或者叫做板级的头文件筛选出来供后面的编译过程使用。

C：为什么要在 **TOPDIR/include/**路径下面创建 **config.mk** 这个配置文件？

从这个文件的内容看我们就知道有什么作用，如上图 9 所示：初始化几个变量。

这些变量在很多地方都使用到了；比如：

（1）.TOPDIR/Makefile 的 133 行，143 行，146 行：（下图 15）

```
133  include $(obj)include/config.mk
134  export  ARCH CPU BOARD VENDOR SOC
135
136  ifndef CROSS_COMPILE
137  ifeq ($(HOSTARCH),$(ARCH))
138  CROSS_COMPILE =
139  else
140  ifeq ($(ARCH),ppc)
141  CROSS_COMPILE = ppc_8xx-
142  endif
143  ifeq ($(ARCH),arm)
144  #CROSS_COMPILE = arm-linux-
145  #CROSS_COMPILE = /usr/local/arm/4.4.1-eabi-cortex-a8/usr/bin/arm-linux-
146  CROSS_COMPILE = /usr/local/arm/4.3.2/bin/arm-linux-
147  #CROSS_COMPILE = /usr/local/arm/arm-2009q3/bin/arm-none-linux-gnueabi-
148  endif
```

133：makefile 包含这个配置文件，134 导出符号后，143 判断“ARCH”是否为 arm，146 指定整个工程的交叉编译器。

（2）.TOPDIR/config.mk 的 113~130 行（下图 16）

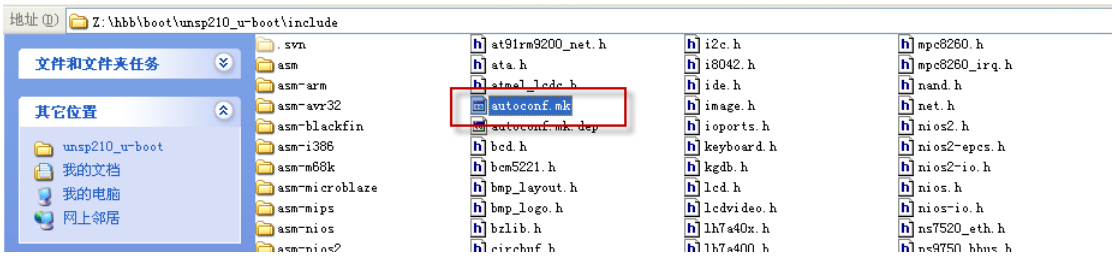
```
113
114 ifdef   ARCH
115 sinclude $(TOPDIR)/$(ARCH)_config.mk    # include architecture dependend rules
116 endif
117 ifdef   CPU
118 sinclude $(TOPDIR)/cpu/$(CPU)/config.mk # include  CPU  specific rules
119 endif
120 ifdef   SOC
121 sinclude $(TOPDIR)/cpu/$(CPU)/$(SOC)/config.mk  # include  SoC  specific rules
122 endif
123 ifdef   VENDOR
124 BOARDDIR = $(VENDOR)/$(BOARD)
125 else
126 BOARDDIR = $(BOARD)
127 endif
128 ifdef   BOARD
129 sinclude $(TOPDIR)/board/$(BOARDDIR)/config.mk  # include board specific rules
130 endif
```

这其中有很多地方都使用了这些变量。

D：为什么要在 **TOPDIR/include/**路径下面定义 **config.h**？

因为 config.h 相当于一个公共的头文件，在编译代码的时候，我们真实的目的是想要 **TOPDIR/include/configs/unsp210.h**，但是编译时只认识 config.h,不认识 unsp210.h，所以创建了一个 config.h 并包含 unsp210.h。为什么需要 unsp210.h 呢？因为 unsp210.h 的内容都是一堆的宏开关可以决定源码是否编译到可执行文件 u-boot.bin 中，换句话说，修改 unsp210.h 可以实现对 u-boot 的裁剪。

unsp210.h 还有其他的作用吗？答案是有！在编译源码的时候，在 **TOPDIR/include** 下会生成一个配置文件 **autoconf.mk**（注：这个文件只是在 make 编译的时候生成，配置时不生成），这个文件非常重要：因为它将在下面编译的过程中发挥作用。（下图 17）



这个文件中的内容如下图所示（下图 18）

```
CONFIG_CMD_FAT=y
CONFIG_USB_OHCI=y
CONFIG_SYS_CLK_FREQ=24000000
CONFIG_CMD_ITEST=y
CONFIG_S3C_HSMMC=y
CONFIG_DISPLAY_BOARDINFO=y
CONFIG_CMD_XIMG=y
CONFIG_CMD_CACHE=y
CONFIG_STACKSIZE="0x40000"
CONFIG_BOOTDELAY=3
CONFIG_CHECK_MPLL_LOCK=y
CONFIG_NR_DRAM_BANKS=2
CONFIG_ETHADDR="00:40:5c:26:0a:5b"
CONFIG_CMD_CONSOLE=y
CONFIG_SW_WORKAROUND=y
CONFIG_GATEWAYIP="172.20.223.254"
CONFIG_ZIMAGE_BOOT=y
CONFIG_CMD_REGINFO=y
CONFIG_MMC=y
CONFIG_NAND_BL1_8BIT_ECC=y
CONFIG_CMD_MISC=y
CONFIG_ZERO_BOOTDELAY_CHECK=y
CONFIG_ENV_OVERWRITE=y
CONFIG_CMD_NET=y
CONFIG_CMD_NFS=y
```

U-boot 的编译过程：

上面只是 u-boot 的配置过程，也就是筛选的过程，从整个 u-boot 源码中筛选出我们有用的源码文件，接下来要做的事情就是要把这些*.c 或*.S 文件变成*.o 目标文件，然后在使用 arm-linux-ld 链接器通过链接脚本*.lds 进行链接生成 ELF 格式的可执行文件，然后再使用 arm-linux-objcopy 工具进行处理，去掉 ELF 格式文件的头信息，生成二进制文件 u-boot.bin

下面采用倒叙的方法介绍编译的过程：

（1）.u-boot.bin 的生成：

（2）.u-boot 的生成（u-boot 链接的全过程）：

（3）.u-boot 的依赖文件的生成：（执行子目录下面的 Makefile 文件）