1. **Shell脚本语法---在Makefile等文件中常见**

**表1.1. 测试命令**

|  |  |
| --- | --- |
| [ -d DIR ] | 如果DIR存在并且是一个目录则为真 |
| [ -f FILE ] | 如果FILE存在且是一个普通文件则为真 |
| [ -z STRING ] | 如果STRING的长度为零则为真 |
| [ -n STRING ] | 如果STRING的长度非零则为真 |
| [ STRING1 = STRING2 ] | 如果两个字符串相同则为真 |
| [ STRING1 != STRING2 ] | 如果字符串不相同则为真 |
| [ ARG1 OP ARG2 ] | ARG1和ARG2应该是整数或者取值为整数的变量，OP是-eq（等于）-ne（不等于）-lt（小于）-le（小于等于）-gt（大于）-ge（大于等于）之中的一个 |

表 1.2. 带与、或、非的测试命令

|  |  |
| --- | --- |
| [ ! EXPR ] | EXPR可以是上表中的任意一种测试条件，!表示逻辑反 |
| [ EXPR1 -a EXPR2 ] | EXPR1和EXPR2可以是上表中的任意一种测试条件，-a表示逻辑与 |
| [ EXPR1 -o EXPR2 ] | EXPR1和EXPR2可以是上表中的任意一种测试条件，-o表示逻辑或 |

makefile中参数：

$0 这个程式的执行名字

$n 这个程式的第n个参数值，n=1..9

$\* 这个程式的所有参数,此选项参数可超过9个。

$# 这个程式的参数个数

$$ 这个程式的PID(脚本运行的当前进程ID号)

$! 执行上一个背景指令的PID(后台运行的最后一个进程的进程ID号)

$? 执行上一个指令的返回值 (显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误)

$- 显示shell使用的当前选项，与set命令功能相同

$@ 跟$\*类似，但是可以当作数组用

**2.u-boot 配置过程分析**

拿到任何一个工程源码，首先是配置，然后是编译，有的需要安装，有的不需要， *u-boot* 也不例外，  
在 *u-boot-2014.04* 源码顶层目录下的 *README* 文件介绍了 *u-boot* 的历史，版本命名规则，目录组织架构，软件配置，如何添加一个新的板子等。目录组织结构在 *README* 中 *Directory Hierarchy* 一节有详细说明，下面列出一些常用的：  
*arch*：存放 *CPU* 架构相关的文件，里面的每个目录代表一种架构。  
*arch/arm/cpu*：存放 *arm CPU* 相关的文件，里面每个目录代表一种 *arm cpu  
board*：存放单板相关的文件  
*board/samsung*：存放三星公司生产的单板  
*include/configs*：存放每个单板的配置文件

软件配置在*README* 中 *Software Configuration* 一节有详细说明， *README* 告诉我们：要针对某个单  
板进行配置，需要执行： *make NAME\_config*，其中 *NAME* 在 *boards.cfg* 中列出。

*S5PV210*和*S5PC100* 都是 *Cortex-A8* 架构，比较类似，我以 *S5PC100* 为例进行说明。

myroot@myroot:~/TQ210N/u-boot-2014.04$ make smdkc100\_config

Configuring for smdkc100 board...

通过执行 *make smdkc100\_config*，其中*smdkc100* 是*smdkc100* 这个单板的一个目标名称，因为有的  
单板有多个配置目标，比如 *atmel* 公司的 *at91sam9263ek* 单板就有 *5* 个目标， *make* 命令首先会从顶层*Makefile* 中搜索匹配目标 *smdkc100\_config*，找到

%\_config:: outputmakefile

@$(MKCONFIG) -A $(@:\_config=)

*%\_config* 前面的*%*是通配符，会匹配所有以*\_config* 为后缀的目标  
*::*是 *Makefile* 的中的多目标规则，可以同时跟多个目标，举例：

myroot@myroot:~/TQ210N/u-boot-2014.04$ make smdkc100\_config smdk2410\_config

Configuring for smdkc100 board...

Configuring for smdk2410 board...

*$(MKCONFIG)*是顶层目录下的一个可执行 *shell* 脚本文件， *$(@:\_config=)*会将所有目标中的后缀  
*\_config*去掉，得到*smdkc100*，然后 *make* 会执行命令：  
*u-boot-2014.04/mkconfig –A smdkc100*下面分析 *mkconfig*首先定义了几个变量：  
*TARGETS*： *Makefile* 的目标  
*arch*：体系架构，比如 *arm*、 *x86*、 *mips* 等  
*cpu*： *cpu* 类型，比如 *arm920t*、 *arm11* 等  
*board*：单板名称，比如 *smdk2410*、 *smdkc100* 等  
*vendor*：厂商名称，比如 *samsung*、 *freescale* 等  
*soc*：片上系统，比如 *s3c2410*、 *s3c2440*、 *s5pv210* 等

if [ \( $# -eq 2 \) -a \( "$1" = "-A" \) ] ; then

# Automatic mode

line=`awk '($0 !~ /^#/ && $7 ~ /^'"$2"'$/) { print $1, $2, $3, $4, $5, $6, $7, $8 }' $srctree/boards.cfg`

if [ -z "$line" ] ; then

echo "make: \*\*\* No rule to make target \`$2\_config'. Stop." >&2

exit 1

fi

set ${line}

# add default board name if needed

[ $# = 3 ] && set ${line} ${1}

fi

如果参数个数等于*2*，而且第 *1* 个参数等于“*-A*”，则执行：  
*line=`awk '($0 !~ /^#/ && $7 ~ /^'"$2"'$/) { print $1, $2, $3, $4, $5, $6, $7, $8 }' $srctree/boards.cfg`  
awk* 是一个非常强大的文本处理工具， *$srctree* 被替换为 *u-boot-2014.04*，即 *u-boot* 源码树目录， *awk*会读取 *u-boot-2014.04/boards.cfg* 中的每一行，如果和前面的表达式*($0 !~ /^#/ && $7 ~ /^'"$2"'$/)*相匹配，则执行*{ print $1, $2, $3, $4, $5, $6, $7, $8 }*。  
*boards.cfg* 保存了各种单板的相关信息，其格式为：  
*# Status, Arch, CPU:SPLCPU, SoC, Vendor, Board name, Target, Options, Maintainers  
################################################################################  
Active arm armv7 s5pc1xx samsung smdkc100 smdc100 - Minkyu Kang<mk7.kang@samsung.com>  
awk* 在执行过程中， *$0* 代表当前整行， 同时将第一个字段存入*$1*，将第 *2* 个字段存入*$2*， 依此类推，*awk* 缺省按空格分段，可以通过*-F* 指定分隔符。

若果当前行不以*#*开头，而且第 *7* 个字段和 *mkconfig* 传进来的第 *2* 个参数（*smdkc100*） 相等，则分别将字段 *1~*字段 *8* 输出到 *line* 中保存，最终得到：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *line=Active* | *arm* | *armv7* | *s5pc1xx* | *samsung* | *smdkc100* | *smdkc100* | *–* |
| *$#=8 $1=Active* | 状态 |  |  |  |  |  |  |
| *$2=arm* | 架构 |  |  |  |  |  |  |
| *$3=armv7* | *CPU* |  |  |  |  |  |  |
| *$4=s5pc1xx* | *SOC* |  |  |  |  |  |  |
| *$5=samsung* | 厂商 |  |  |  |  |  |  |
| *$6=smdkc100* | 单板名称 |  |  |  |  |  |  |
| *$7=smdkc100* | 配置目标 |  |  |  |  |  |  |
| *$8=-* | 选项 |  |  |  |  |  |  |

# Strip all options and/or \_config suffixes

CONFIG\_NAME="${7%\_config}"

[ "${BOARD\_NAME}" ] || BOARD\_NAME="${7%\_config}"

*${7%\_config}*表示去掉*${7}*的后缀*\_config*，这里*${7}=smdkc100*，得到  
*CONFIG\_NAME=smdkc100  
BOARD\_NAME=smdkc100*

arch="$2"

cpu=`echo $3 | awk 'BEGIN {FS = ":"} ; {print $1}'`

spl\_cpu=`echo $3 | awk 'BEGIN {FS = ":"} ; {print $2}'`

if [ "$6" = "<none>" ] ; then

board=

elif [ "$6" = "-" ] ; then

board=${BOARD\_NAME}

else

board="$6"

fi

[ "$5" != "-" ] && vendor="$5"

[ "$4" != "-" ] && soc="$4"

*arch=arm  
cpu=armv7  
spl\_cpu=*

*board=smdkc100  
vendor=samsung*

*soc=s5pc1xx*

if [ "$options" ] ; then

echo "Configuring for ${BOARD\_NAME} - Board: ${CONFIG\_NAME}, Options: ${options}"

else

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

fi

*$8=- 所以*

*Configuring for smdkc100 board...*

#

# Create link to architecture specific headers

#

if [ -n "$KBUILD\_SRC" ] ; then @@KBUILD\_SRC=

mkdir -p ${objtree}/include

LNPREFIX=${srctree}/arch/${arch}/include/asm/

cd ${objtree}/include

mkdir -p asm

else

cd arch/${arch}/include

fi

rm -f asm/arch

if [ -z "${soc}" ] ; then

ln -s ${LNPREFIX}arch-${cpu} asm/arch

else

ln -s ${LNPREFIX}arch-${soc} asm/arch

fi

if [ "${arch}" = "arm" ] ; then

rm -f asm/proc

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

fi

if [ -z "$KBUILD\_SRC" ] ; then

cd ${srctree}/include

fi

KBUILD\_SRC=

创建架构相关的头文件的链接  
如果 *KBUILD\_SRC* 不为空，则进入到 *arch/arm/include* 目录，删除里面的 *asm/arch* 这个链接文件。  
如果*${soc}*为空，则执行 *ln -s ${LNPREFIX}arch-${cpu} asm/arch*，否则执行*ln -s ${LNPREFIX}arch-${soc} asm/arch*， 这里 *soc=s5pc1xx*， *LNPREFIX* 为空，所以会执行*ln –s arch-s5pc1xx asm/arch*

这里给*u-boot-2014.04/arch/arm/include/asm/arch-s5pc1xx* 创建了一个软连接，如下所示：

myroot@myroot:~/TQ210N/u-boot-2014.04/arch/arm/include/asm$ ll arch

lrwxrwxrwx 1 myroot myroot 12 12月 16 22:54 arch -> arch-s5pc1xx/

紧接着，如果*${arch} = “arm”*， 则删除 *asm/proc*，同时给 *proc-armv* 创建一个软链接，如下所示：

myroot@myroot:~/TQ210N/u-boot-2014.04/arch/arm/include/asm$ ll proc

lrwxrwxrwx 1 myroot myroot 9 12月 16 22:54 proc -> proc-armv/

如果 *KBUILD\_SRC* 为空，则进入到 *u-boot-2014.04/include*，这里 *KBUILD\_SRC* 为空，所以会进入到  
*u-boot-2014.04/include* 目录。

#

# Create include file for Make

#

( 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

# Assign board directory to BOARDIR variable

if [ -z "${vendor}" ] ; then

BOARDDIR=${board}

else

BOARDDIR=${vendor}/${board}

fi

为 *make* 创建头文件  
*ARCH=arm  
CPU=arm  
BOARD=smdkc100  
VENDOR=samsung  
SOC=s5pc1xx*

使用 *echo* 将这些信息一行一行的输出到 *u-boot-2014.04/include/config.mk* 中：

然后给变量 *BOARDDIR* 赋值为 *samsung/smdkc100*

#

# 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

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

[ "${board}" ] && echo "#define CONFIG\_BOARDDIR board/$BOARDDIR" >> config.h

cat << EOF >> config.h

#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

创建单板相关的头文件

*APPEND=no*，所以会创建一个新的 *u-boot-2014.04/include/config.h*

/\* Automatically generated - do not edit \*/

#define CONFIG\_SYS\_ARCH "arm"

#define CONFIG\_SYS\_CPU "armv7"

#define CONFIG\_SYS\_BOARD "smdkc100"

#define CONFIG\_SYS\_VENDOR "samsung"

#define CONFIG\_SYS\_SOC "s5pc1xx"

#define CONFIG\_BOARDDIR board/samsung/smdkc100

#include <config\_cmd\_defaults.h>

#include <config\_defaults.h>

#include <configs/smdkc100.h>

#include <asm/config.h>

#include <config\_fallbacks.h>

#include <config\_uncmd\_spl.h>

在这个头文件中定义了单板相关的宏，包含了单板的配置头文件 *configs/smdkc100.h*，我们需要针对  
自己的单板配置这个头文件。

**总结： 执行 *make smdkc100\_config* 主要做了 *3* 件事：**

1. 创建架构相关的头文件的链接

myroot@myroot:~/TQ210N/u-boot-2014.04/arch/arm/include/asm$ ll arch

lrwxrwxrwx 1 myroot myroot 12 12月 16 22:54 arch -> arch-s5pc1xx/

myroot@myroot:~/TQ210N/u-boot-2014.04/arch/arm/include/asm$ ll proc

lrwxrwxrwx 1 myroot myroot 9 12月 16 22:54 proc -> proc-armv/

1. 为 *make* 创建头文件 *u-boot-2014.04/include/config.mk*

ARCH=arm  
CPU=arm  
BOARD=smdkc100  
VENDOR=samsung  
SOC=s5pc1xx

*3)* 创建单板相关的头文件 *u-boot-2014.04/include/config.h*

**3.u-boot编译过程分析**

**4.SPL**

参考 *doc/README.SPL*

我们知道， *S5PV210*启动过程需要 *BL1*， *BL2*，其中 *BL1* 进行一些基本初始化（时钟、内存），加载*BL2* 到内存。 *U-boot*为此专门实现了*SPL* 框架。  
 *SPL* 的全称为 *secondary program loader*，即第 *2* 阶段程序加载器，即我们要实现的 *BL1*。  
要支持*SPL*，我们需要在单板配置文件 *u-boot-2014.04/include/configs/smdkc100.h* 定义宏 *CONFIG\_SPL  
#define CONFIG\_SPL*这样顶层 *Makefile* 在包含 *u-boot-2014.04/include/configs/smdkc100.h* 时，会得到*ALL-y += spl/u-boot-spl.bin*

这样在执行 *make all* 时， *make* 会编译 *spl/u-boot-spl.bin* 这个目标

spl/u-boot-spl.bin: spl/u-boot-spl

@:

spl/u-boot-spl: tools prepare

$(Q)$(MAKE) obj=spl -f $(srctree)/spl/Makefile all

从这得知， *spl/u-boot-spl.bin* 依赖于 *spl/u-boot-spl*，然后 *make* 找到目标 *spl/u-boot-spl*，然后执行指定的 *Makefile* 文件： *u-boot-2014.04/spl/Makefile*。  
 *u-boot-2014.04/spl/Makefile* 中首先导出环境变量 *CONFIG\_SPL\_BUILD=y*， 这个宏在各个源代码文件中用来控制代码的走向。  
 这样编译后，最终会在 *u-boot-2014.04/spl* 下生成 *u-boot-spl.bin*， 同时在 *u-boot-2014.04* 下生成  
*u-boot.bin  
 u-boot.bin* 使用的链接脚本为 *u-boot-2014.04/arch/arm/cpu/u-boot.lds  
 u-boot-spl.bin* 使用的链接脚本为 *u-boot-2014.04/arch/arm/cpu/u-boot-spl.lds*

1. **添加自己的单板**

在 *u-boot* 的帮助文档 *README* 中有介绍如何添加自己的单板：

If the system board that you have is not listed, then you will need

to port U-Boot to your hardware platform. To do this, follow these

steps:

1. Add a new configuration option for your board to the toplevel

"boards.cfg" file, using the existing entries as examples.

Follow the instructions there to keep the boards in order.

2. Create a new directory to hold your board specific code. Add any

files you need. In your board directory, you will need at least

the "Makefile", a "<board>.c", "flash.c" and "u-boot.lds".

3. Create a new configuration file "include/configs/<board>.h" for

your board

3. If you're porting U-Boot to a new CPU, then also create a new

directory to hold your CPU specific code. Add any files you need.

4. Run "make <board>\_config" with your new name.

5. Type "make", and you should get a working "u-boot.srec" file

to be installed on your target system.

6. Debug and solve any problems that might arise.

[Of course, this last step is much harder than it sounds.]

1. 首先在顶层目录下的 *boards.cfg* 中添加自己的单板的硬件配置信息，我们仿照 *smdkc100*

Active arm armv7 s5pc1xx samsung smdkv210 smdkv210 - cumtzwf [<cumtzwf@163.com>](mailto:<cumtzwf@163.com>)

1. 在顶层目录下的 *board* 目录下创建一个用于存储单板相关代码文件的目录，我们直接拷贝  
   *board/samsung/smdkc100/*目录为 *board/samsung/smdkv210*，然后修改里面的文件名

myroot@myroot:~/TQ210N/u-boot-2014.04/board/samsung/smdkv210$ mv smdkc100.c smdkv210.c

|  |  |
| --- | --- |
| Makefile修改前：  obj-y := smdkc100.o  obj-$(CONFIG\_SAMSUNG\_ONENAND) += onenand.o  obj-y += lowlevel\_init.o | Makefile修改后：  obj-y := smdkv210.o  obj-$(CONFIG\_SAMSUNG\_ONENAND) += onenand.o  obj-y += lowlevel\_init.o |

1. 为自己的单板创建配置文件 *include/configs/<board>.h* ，也是直接拷贝 *smdkc100* 的配置文件

[myroot@myroot:~/TQ210N/u-boot-2014.04$](mailto:myroot@myroot:~/TQ210N/u-boot-2014.04$) cp include/configs/smdkc100.h include/configs/smdkv210.h

现在可以测试一下：

编写编译脚本：build-uboot.sh

myroot@myroot:~/TQ210N/u-boot-2014.04$ vim build-uboot.sh

make smdkv210\_config

make ARCH=arm CROSS\_COMPILE=arm-linux-

一切 *ok* 的话，最终在顶层目录下生成 *u-boot.bin*

***u-boot* 源码包含很多 *CPU* 架构和单板相关的文件，我们只关心 *S5PV210*，其它的都用不到，可以把  
不用的删掉：**

*u-boot-2014.04/arch* 目录下只保留 *arm* 目录

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls arch/

arm

*u-boot-2014.04/arch/arm* 目录下保留内容如下：

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls arch/arm/

config.mk cpu include lib

*u-boot-2014.04/arch/arm/cpu* 目录下保留内容如下：

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls arch/arm/cpu/

armv7 Makefile u-boot.lds u-boot-spl.lds

*u-boot-2014.04/arch/arm/include/asm* 目录下以 *arch-*开头的目录只保留 *arch-s5pc1xx*

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls arch/arm/include/asm/

arch-s5pc1xx byteorder.h errno.h kona-common omap\_gpio.h ptrace.h u-boot-arm.h

armv7.h cache.h gic.h linkage.h omap\_mmc.h sections.h u-boot.h

armv8 config.h global\_data.h mach-types.h omap\_musb.h setup.h unaligned.h

assembler.h davinci\_rtc.h gpio.h macro.h pl310.h spl.h utils.h

atomic.h dma-mapping.h hardware.h memory.h posix\_types.h string.h

bitops.h ehci-omap.h imx-common omap\_boot.h proc-armv system.h

bootm.h emif.h io.h omap\_common.h processor.h types.h

*u-boot-2014.04/board* 目录下只保留 *samsung*

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls board/

samsung

*u-boot-2014.04/board/samsung* 目录下只保留如下内容

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls board/samsung/

common smdkv210

*u-boot-2014.04/include/configs* 目录下只保留 *smdkv210.h*

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls include/configs/

smdkv210.h

把多余的文件删除后，同时执行 *make distclean* 清楚编译产生的目标文件后，经过压缩后，大小只  
有 *4MB*，原来有 *9M* 多

1. **移植 u-boot-spl.bin**

首先大致分析一下 *u-boot* 的代码走向， 通过看链接脚本 *u-boot-2014.04/arch/arm/cpu/u-boot.lds* 和*u-boot-2014.04/arch/arm/cpu/u-boot-spl.lds* 得知，不管是 *u-boot.bin* 还是 *u-boot-spl.bin* 都以  
*u-boot-2014.04/arch/arm/cpu/armv7/start.S* 开始，同时在链接脚本文件中指定了程序的入口函数  
*ENTRY(\_start)*

.globl \_start

\_start: b reset

ldr pc, \_undefined\_instruction

ldr pc, \_software\_interrupt

ldr pc, \_prefetch\_abort

ldr pc, \_data\_abort

ldr pc, \_not\_used

ldr pc, \_irq

ldr pc, \_fiq

*start.S* 首先安装了异常向量表， 执行第 *1* 条指令，然后跳转到这里：

reset:

bl save\_boot\_params

/\*

\* disable interrupts (FIQ and IRQ), also set the cpu to SVC32 mode,

\* except if in HYP mode already

\*/

mrs r0, cpsr

and r1, r0, #0x1f @ mask mode bits

teq r1, #0x1a @ test for HYP mode

bicne r0, r0, #0x1f @ clear all mode bits

orrne r0, r0, #0x13 @ set SVC mode

orr r0, r0, #0xc0 @ disable FIQ and IRQ

msr cpsr,r0

接着往下执行到：

/\* the mask ROM code should have PLL and others stable \*/

#ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT

bl cpu\_init\_cp15

bl cpu\_init\_crit

#endif

这里有个判断，如果没有定义 *CONFIG\_SKIP\_LOWLEVEL\_INIT* 这个宏，则进行一些底层的初始化（不  
跳过底层初始化），其中 *cpu\_init\_crit* 又调用了 *lowlevel\_init*，这个函数在  
*u-boot-2014.04/board/samsung/smdkv210/lowlevel\_init.S* 中定义，我们可以在里面进行时钟、内存等  
底层的初始化。然后跳转到函数*\_main*， 这个函数在 *u-boot-2014.04/arch/arm/lib/crt0.S* 中定义， *crt* 即 *C Run-time*

/\*

\* entry point of crt0 sequence

\*/

ENTRY(\_main)

/\*

\* Set up initial C runtime environment and call board\_init\_f(0).

\*/

#if defined(CONFIG\_SPL\_BUILD) && defined(CONFIG\_SPL\_STACK)

ldr sp, =(CONFIG\_SPL\_STACK)

#else

ldr sp, =(CONFIG\_SYS\_INIT\_SP\_ADDR)

#endif

bic sp, sp, #7 /\* 8-byte alignment for ABI compliance \*/

sub sp, sp, #GD\_SIZE /\* allocate one GD above SP \*/

bic sp, sp, #7 /\* 8-byte alignment for ABI compliance \*/

mov r9, sp /\* GD is above SP \*/

mov r0, #0

bl board\_init\_f

*ENTRY(\_main)*展开即为：  
*.globl \_main;  
ALIGN;  
\_main:*

接着调用函数 *board\_init\_f*，这个函数在*u-boot-2014.04/arch/arm/lib/board.c* 中定义， 进行一些基本  
的硬件初始化，为进入 *DRAM* 内存运行做准备，然后继续往下执行。

/\*

\* Set up intermediate environment (new sp and gd) and call

\* relocate\_code(addr\_moni). Trick here is that we'll return

\* 'here' but relocated.

\*/

ldr sp, [r9, #GD\_START\_ADDR\_SP] /\* sp = gd->start\_addr\_sp \*/

bic sp, sp, #7 /\* 8-byte alignment for ABI compliance \*/

ldr r9, [r9, #GD\_BD] /\* r9 = gd->bd \*/

sub r9, r9, #GD\_SIZE /\* new GD is below bd \*/

adr lr, here

ldr r0, [r9, #GD\_RELOC\_OFF] /\* r0 = gd->reloc\_off \*/

add lr, lr, r0

ldr r0, [r9, #GD\_RELOCADDR] /\* r0 = gd->relocaddr \*/

b relocate\_code

这里调用 *relocate\_code*，重定位，将 *u-boot* 搬运到 *DRAM* 内存

/\* Set up final (full) environment \*/

bl c\_runtime\_cpu\_setup /\* we still call old routine here \*/

ldr r0, =\_\_bss\_start /\* this is auto-relocated! \*/

ldr r1, =\_\_bss\_end /\* this is auto-relocated! \*/

mov r2, #0x00000000 /\* prepare zero to clear BSS \*/

clbss\_l:cmp r0, r1 /\* while not at end of BSS \*/

strlo r2, [r0] /\* clear 32-bit BSS word \*/

addlo r0, r0, #4 /\* move to next \*/

blo clbss\_l

bl coloured\_LED\_init

bl red\_led\_on

/\* call board\_init\_r(gd\_t \*id, ulong dest\_addr) \*/

mov r0, r9 /\* gd\_t \*/

ldr r1, [r9, #GD\_RELOCADDR] /\* dest\_addr \*/

/\* call board\_init\_r \*/

ldr pc, =board\_init\_r /\* this is auto-relocated! \*/

/\* we should not return here. \*/

这里执行清 *BSS* 操作，然后把函数 *board\_init\_r* 的链接地址赋给 *PC* 指针，直接跳转到 *DRAM* 内存运行，在 *board\_init\_r* 继续执行一些硬件初始化，最终执行到这里

/\* main\_loop() can return to retry autoboot, if so just run it again \*/

for (;;)

main\_loop();

*main\_loop* 在*u-boot-2014.04/common/main.c* 中定义，在*u-boot-2014.04/common/board\_r.c中调用*， 在 *main\_loop* 中会检测用户的输入，如果在指定的延时时间内，没有用户按下按钮，则根据环境变量，从 *SD* 卡或 *FLASH* 拷贝内核镜像到 *DRAM*，然后跳转到 *OS* 起始地址执行 *OS*。 这里只是大致分析了一下代码的走向，并没有详细分析。 具体可以根据代码走向一步一步分析代码。

**首先移植 *u-boot-spl.bin***

参照u-boot-2014.04/doc/README.SPL

The building of SPL images can be with:

#define CONFIG\_SPL

我们采用*SPL* 方式，因此需要在单板配置文件 *u-boot-2014.04/include/configs/smdkv210.h* 中添加宏

#define CONFIG\_SPL

#endif /\* \_\_CONFIG\_H \*/

我们可以进行如下操作，只编译*u-boot-spl.bin*

myroot@myroot:~/TQ210N/u-boot-2014.04$ make smdkv210\_config

Configuring for smdkv210 board...

myroot@myroot:~/TQ210N/u-boot-2014.04$ make ARCH=arm CROSS\_COMPILE=arm-linux- spl/u-boot-spl.bin

编译出错：

LD spl/arch/arm/cpu/built-in.o

LDS spl/u-boot-spl.lds

LD spl/u-boot-spl

arch/arm/lib/built-in.o: In function `\_main':

/home/myroot/TQ210N/u-boot-2014.04/arch/arm/lib/crt0.S:74: undefined reference to `board\_init\_f'

/home/myroot/TQ210N/u-boot-2014.04/spl/Makefile:225: recipe for target 'spl/u-boot-spl' failed

make[1]: \*\*\* [spl/u-boot-spl] Error 1

Makefile:1082: recipe for target 'spl/u-boot-spl' failed

make: \*\*\* [spl/u-boot-spl] Error 2

我们前面分析得知在 *u-boot-2014.04/arch/arm/lib/board.c* 中定义了一个函数*board\_init\_f*，这个函数  
进行了非常多的初始化操作，由于 *u-boot-spl.bin* 的最终目的只是把 *BL2* 从外部存储器（*SD* 卡、 *NAND*）拷贝到 *SDRAM*，所以 *u-boot-spl.bin* 只需初始化时钟、 *SDRAM*、 *NAND*，然后调用一个拷贝函数，拷贝完成后直接跳转到 *SDRAM* 执行*BL2*，就完事了，我们可以修改 *crt0.S*，不让其调用 *board\_init\_f*。

首先在 *u-boot-2014.04/arch/arm/include/asm/arch-s5pc1xx/cpu.h* 中添加与 *S5PV210* 相关的寄存器定  
义，后面会用到

/\* S5PV210 add by zwf \*/

#define S5PV210\_PRO\_ID 0xE0000000

#define S5PV210\_CLOCK\_BASE 0xE0100000

#define S5PV210\_GPIO\_BASE 0xE0200000

#define S5PV210\_PWMTIMER\_BASE 0xE2500000

#define S5PV210\_WATCHDOG\_BASE 0xE2700000

#define S5PV210\_UART\_BASE 0xE2900000

#define S5PV210\_SROMC\_BASE 0xE8000000

#define S5PV210\_MMC\_BASE 0xEB000000

#define S5PV210\_DMC0\_BASE 0xF0000000

#define S5PV210\_DMC1\_BASE 0xF1400000

#define S5PV210\_VIC0\_BASE 0xF2000000

#define S5PV210\_VIC1\_BASE 0xF2100000

#define S5PV210\_VIC2\_BASE 0xF2200000

#define S5PV210\_VIC3\_BASE 0xF2300000

在 *u-boot-2014.04/board/samsung/smdkv210/lowlevel\_init.S* 中的初始化都是汇编代码， 我还是喜欢用  
*C* 语言，再说 *S5PV210* 的性能如此之高，也不在乎损耗一点性能，将其修改为：

/\* modied by zwf \*/

.globl lowlevel\_init

lowlevel\_init:

mov r9, lr

#ifdef CONFIG\_SPL\_BUILD

bl clock\_init /\* 时钟初始化 \*/

bl ddr\_init /\* DDR初始化 \*/

#endif

mov pc, r9 /\* 返回 \*/

由于这些操作只需要在 *u-boot-spl.bin* 中进行，因此这里使用 *CONFIG\_SPL\_BUILD* 宏来控制，当编译*u-boot-spl.bin* 才会将这些初始化代码编译进 *u-boot-spl.bin*，而编译 *u-boot.bin* 时就不会。  
这里调用了两个函数， 我们在 *u-boot-2014.04/board/samsung/smdkv210/smdkv210.c* 中实现它，参照  
第 *4* 章裸机编程。由于这两个函数只需要在 *u-boot-spl.bin* 中实现，而 *u-boot.bin* 不需要，同时 *smdkv210.c*中的其他函数只需要在 *u-boot.bin* 中实现，前面分析过，编译 *u-boot-spl.bin* 时， *spl/Makefile* 会导出一个宏 *CONFIG\_SPL\_BUILD*，我们通过这个宏来控制代码是否被编译，下面列出修改后的框架：

*u-boot-2014.04/board/samsung/smdkv210/smdkv210.c*

#ifndef CONFIG\_SPL\_BUILD /\* add by zwf \*/

#else /\* CONFIG\_SPL\_BUILD (add by zwf) \*/

void clock\_init(void)

#endif /\* CONFIG\_SPL\_BUILD (add by zwf) \*/

实现 *clock\_init* 会用到时钟相关的寄存器，需要在  
*u-boot-2014.04/arch/arm/include/asm/arch-s5pc1xx/clock.h* 中添加 *S5PV210* 的时钟相关寄存器的定义

/\* add by zwf \*/

struct s5pv210\_clock {

unsigned int apll\_lock;

unsigned char res1[0x04];

unsigned int mpll\_lock;

unsigned char res2[0x04];

unsigned int epll\_lock;

unsigned char res3[0x0C];

unsigned int vpll\_lock;

unsigned char res4[0xdc];

unsigned int apll\_con0;

unsigned int apll\_con1;

unsigned int mpll\_con;

unsigned char res5[0x04];

unsigned int epll\_con0;

unsigned int epll\_con1;

unsigned char res6[0x08];

unsigned int vpll\_con;

unsigned char res7[0xdc];

unsigned int src0;

unsigned int src1;

unsigned int src2;

unsigned int src3;

unsigned int src4;

unsigned int src5;

unsigned int src6;

unsigned char res8[0x64];

unsigned int mask0;

unsigned int mask1;

unsigned char res9[0x78];

unsigned int div0;

unsigned int div1;

unsigned int div2;

unsigned int div3;

unsigned int div4;

unsigned int div5;

unsigned int div6;

unsigned int div7;

};

在 *u-boot-2014.04/board/samsung/smdkv210/smdkv210.c* 中添加头文件

#include <asm/arch/clock.h> /\* add by zwf \*/

这里的 *arch* 链接到 *arch-s5pc1xx*，前面分析过在*clock\_init* 函数中使用到了一个宏 *samsung\_get\_base\_clock*struct s5pv210\_clock \*const clock = (struct s5pv210\_clock \*)samsung\_get\_base\_clock();  
这个宏在 *u-boot-2014.04/arch/arm/include/asm/arch-s5pc1xx/cpu.h* 中定义

/\* modied by zwf \*/ \

#define SAMSUNG\_BASE(device, base) \

static inline unsigned int samsung\_get\_base\_##device(void) \

{ \

return S5PV210\_##base; \

}

SAMSUNG\_BASE(clock, CLOCK\_BASE)

SAMSUNG\_BASE(gpio, GPIO\_BASE)

SAMSUNG\_BASE(pro\_id, PRO\_ID)

SAMSUNG\_BASE(mmc, MMC\_BASE)

SAMSUNG\_BASE(sromc, SROMC\_BASE)

SAMSUNG\_BASE(timer, PWMTIMER\_BASE)

SAMSUNG\_BASE(uart, UART\_BASE)

SAMSUNG\_BASE(watchdog, WATCHDOG\_BASE)

/\* add by zwf \*/

SAMSUNG\_BASE(dmc0, DMC0\_BASE)

SAMSUNG\_BASE(dmc1, DMC1\_BASE)

这个宏展开即为：

SAMSUNG\_BASE(clock, CLOCK\_BASE)

static inline unsigned int samsung\_get\_base\_clock(void)\

{\

if(cpu\_is\_s5pc100()) \

return S5PC100\_CLOCK\_BASE; \

else if(cpu\_is\_s5pc110()) \

return S5PC110\_CLOCK\_BASE; \

else \

return 0; \

}

这里首先判断 *CPU* 类型，这里只判断了 *S5PC100*和 *S5PC110*，这里判断 *CPU* 类型同样使用的是宏

#define IS\_SAMSUNG\_TYPE(type, id) \

static inline int cpu\_is\_##type(void) \

{ \

return s5p\_cpu\_id == id ? 1 : 0; \

}

IS\_SAMSUNG\_TYPE(s5pc100, 0xc100)

IS\_SAMSUNG\_TYPE(s5pc110, 0xc110)

这个宏展开即为：

#define IS\_SAMSUNG\_TYPE(s5pc100, 0xc100) \

static inline int cpu\_is\_s5pc100(void) \

{ \

return s5p\_cpu\_id == 0xc100 ? 1 : 0; \

}

这里判断了一个变量 *s5p\_cpu\_id*，这个变量通过 *s5p\_set\_cpu\_id* 函数读取寄存器 *PRO\_ID* 来设置  
我们不用那么麻烦，直接修改 *SAMSUNG\_BASE* 这个宏，让它直接返回 *S5PV210\_##base*

/\* modied by zwf \*/ \

#define SAMSUNG\_BASE(device, base) \

static inline unsigned int samsung\_get\_base\_##device(void) \

{ \

return S5PV210\_##base; \

}

*clock\_init* 函数完成了，接下来需要实现 *ddr\_init*，同样首先需要定义寄存器，仿照 *clock.h*，编写  
*u-boot-2014.04/arch/arm/include/asm/arch-s5pc1xx/dmc.h*

/\* add by zwf \*/

#ifndef \_\_ASM\_ARM\_ARCH\_DRAM\_H\_

#define \_\_ASM\_ARM\_ARCH\_DRAM\_H\_

#ifndef \_\_ASSEMBLY\_\_

struct s5pv210\_dmc0 {

unsigned int concontrol;

unsigned int memcontrol;

在 *u-boot-2014.04/board/samsung/smdkv210/smdkv210.c* 中添加头文件

#include <asm/arch/dmc.h> /\* add by zwf \*/

同时在 *u-boot-2014.04/arch/arm/include/asm/arch-s5pc1xx/cpu.h* 中添加宏

/\* add by zwf \*/

SAMSUNG\_BASE(dmc0, DMC0\_BASE)

SAMSUNG\_BASE(dmc1, DMC1\_BASE)

现在就可以仿照 *clock\_init* 实现 *ddr\_init* 了，具体请看源码

接着执行到 *u-boot-2014.04/arch/arm/lib/crt0.S* 中的*\_main* 函数

|  |  |
| --- | --- |
| 修改前：  #if defined(CONFIG\_SPL\_BUILD) && defined(CONFIG\_SPL\_STACK)  ldr sp, =(CONFIG\_SPL\_STACK)  #else  ldr sp, =(CONFIG\_SYS\_INIT\_SP\_ADDR)  #endif | 修改后：  /\* modied by zwf \*/  #if !defined(CONFIG\_SPL\_BUILD)  ldr sp, =(CONFIG\_SYS\_INIT\_SP\_ADDR) |

这里在设置栈指针， 由于 *u-boot-spl.bin* 相当小，使用 *SRAM* 为其提供栈内存就够了，所以可以将这  
里的栈指针设置代码屏蔽掉（为 *u-boot-spl.bin* 屏蔽，对 *u-boot.bin* 没有影响），如下：

#ifdef CONFIG\_SPL\_BUILD

bl copy\_bl2\_to\_ram /\* 拷贝BL2到DDR \*/

ldr pc, =CONFIG\_SYS\_SDRAM\_BASE /\* 跳转到DDR的起始地址执行BL2 \*/

#else

bl board\_init\_f

#endif

这里调用了一个函数 *copy\_bl2\_to\_ram*， 用于从 *SD* 卡或 *NAND* 拷贝 *BL2* 到 *SDRAM*在 *u-boot-2014.04/board/samsung/smdkv210/smdkv210.c* 中实现它，拷贝完成后，直接跳转到 *DDR*的起始地址执行 *BL2*。*CONFIG\_SYS\_SDRAM\_BASE* 在 *u-boot-2014.04/include/configs/smdkv210.h* 中定义，我们将其修改为我们实际*DDR* 的起始地址 *0x20000000*

/\* DRAM Base \*/

#define CONFIG\_SYS\_SDRAM\_BASE 0x20000000 /\* modied by zwf \*/

在 *copy\_bl2\_to\_ram* 中我们实现从 *SD* 卡的扇区 *32* 拷贝 *BL2* 到 *DDR* 的起始地址 *0x20000000*

问：为什么是 *32*？

答：三星手册明确规定 *BL1* 的大小最大为 *16KB*，而每一个扇区为 *512B*， *16KB* 刚好在扇区 *32*。 这不是强制要求的， 只要这个数字乘以 *512* 小于你的 *u-boot-spl.bin* 就可以了。

void copy\_bl2\_to\_ram(void)

{

/\*

\*\* ch: 通道

\*\* sb: 起始块

\*\* bs: 块大小

\*\* dst: 目的地

\*\* i: 是否初始化

\*/

#define CopySDMMCtoMem(ch, sb, bs, dst, i) \

(((unsigned char(\*)(int, unsigned int, unsigned short, unsigned int\*, unsigned char))\

(\*((unsigned int \*)0xD0037F98)))(ch, sb, bs, dst, i))

unsigned int V210\_SDMMC\_BASE = \*(volatile unsigned int \*)(0xD0037488); // V210\_SDMMC\_BASE

unsigned char ch = 0;

/\* 参考S5PV210手册7.9.1 SD/MMC REGISTER MAP \*/

if (V210\_SDMMC\_BASE == 0xEB000000) // 通道0

ch = 0;

else if (V210\_SDMMC\_BASE == 0xEB200000) // 通道2

ch = 2;

CopySDMMCtoMem(ch, 32, 10, (unsigned int \*)CONFIG\_SYS\_SDRAM\_BASE, 0);

}

由于现在我们还不知道 *BL2* 有多大，所以拷贝多大先暂时随便写个值。

现在执行 *make spl/u-boot-spl.bin*，出现错误

myroot@myroot:~/TQ210N/u-boot-2014.04$ make ARCH=arm CROSS\_COMPILE=arm-linux- spl/u-boot-spl.bin

OBJCOPY spl/u-boot-spl.bin

/home/myroot/TQ210N/u-boot-2014.04/tools/mkexynosspl spl/u-boot-spl.bin spl/smdkv210-spl.bin

make[1]: /home/myroot/TQ210N/u-boot-2014.04/tools/mkexynosspl: Command not found

/home/myroot/TQ210N/u-boot-2014.04/spl/Makefile:200: recipe for target 'spl/smdkv210-spl.bin' failed

make[1]: \*\*\* [spl/smdkv210-spl.bin] Error 127

Makefile:1082: recipe for target 'spl/u-boot-spl' failed

make: \*\*\* [spl/u-boot-spl] Error 2

分析 *u-boot-2014.04/spl/Makefile*

ifdef CONFIG\_SAMSUNG

ALL-y += $(obj)/$(BOARD)-spl.bin

endif

all: $(ALL-y)

ifdef CONFIG\_SAMSUNG

ifdef CONFIG\_VAR\_SIZE\_SPL

VAR\_SIZE\_PARAM = --vs

else

VAR\_SIZE\_PARAM =

endif

$(obj)/$(BOARD)-spl.bin: $(obj)/u-boot-spl.bin

$(if $(wildcard $(objtree)/spl/board/samsung/$(BOARD)/tools/mk$(BOARD)spl),\

$(objtree)/spl/board/samsung/$(BOARD)/tools/mk$(BOARD)spl,\

$(objtree)/tools/mkexynosspl) $(VAR\_SIZE\_PARAM) $< $@

endif

如果定义了*CONFIG\_SAMSUNG*，就添加目标*$(obj)/$(BOARD)-spl.bin*

*CONFIG\_SAMSUNG* 在*u-boot-2014.04/include/configs/smdkv210.h* 中定义  
*$(obj)/$(BOARD)-spl.bin* 展开为 *spl/smdkv210-spl.bin*

这里需要调用一个工具用来给 *u-boot-spl.bin* 添加头信息，我们将之前制作头信息的工具（*addheader*）  
拷贝到*u-boot-2014.04/tools*里面，并更名为*mksmdkv210spl*

myroot@myroot:~/TQ210N/u-boot-2014.04$ ls tools/| grep mks

mksmdkv210spl

然后修改 *u-boot-2014.04/spl/Makefile*

$(obj)/$(BOARD)-spl.bin: $(obj)/u-boot-spl.bin

$(objtree)/tools/mk$(BOARD)spl $< $@

endif

现在再次执行 *make spl/u-boot-spl.bin* 编译，最终生成 *spl/smdkv210-spl.bin*，已经添加了 *16B* 头信息

OBJCOPY spl/u-boot-spl.bin

/home/myroot/TQ210N/u-boot-2014.04/tools/mksmdkv210spl spl/u-boot-spl.bin spl/smdkv210-spl.bin

测试一下 *spl/smdkv210-spl.bin*使用 *DD\_For\_Windows.exe* 将 *spl/smdkv210-spl.bin* 烧写到 *SD* 的扇区 *1*，将裸机程序  
*tq210\_NoOS/1-led\_on\_c* 中的 *led.bin* 烧写到 *SD* 卡的扇区 *32*



注意： 这里要取消“添加头信息”，因为 *spl/smdkv210-spl.bin* 已经添加了头信息， 如果烧写  
*spl/u-boot-spl.bin* 就需要选中“添加头信息”



然后取下 *SD* 卡，将其插入开发板，可以看到 *LED* 流水灯效果。