# 从零开始移植uboot2017.01

S5PV210开发板

**一、移植前的准备**

**交叉编译工具链的安装**

1.在/usr/local/下面创建一个arm的文件夹，把交叉编译工具解压到下面去。

/usr/local/arm/7.3.1/ 所用的是7.3.1的版本

1. 解压好uboot，添加好架构和交叉编译工具链
2. 修改/Makefile,配置好交叉编译器

VERSION = 2017

PATCHLEVEL = 01

SUBLEVEL =

EXTRAVERSION =

NAME =

ARCH=arm

CROSS\_COMPILE=arm-linux-gnueabihf-

1. 安装设备树编译链

sudo apt-get install device-tree-compiler

1. 因为uboot-201701没有为S5PV210适配相应的单板，所以我们要自己找一个相似的移植。

其中有两块开发板比较相似，smdkc100和goni，因为smdkc100部分寄存器的基地址和s5pv210的不一样，移植更麻烦一下些，所以我们从goni上开始移植。

①：这里拷贝一份s5p\_goni\_defconfig的配置文件

cp configs/s5p\_goni\_defconfig configs/smdkv210\_defconfig

|  |  |
| --- | --- |
| 修改前：  CONFIG\_ARM=y  CONFIG\_ARCH\_S5PC1XX=y  CONFIG\_TARGET\_S5P\_GONI=y  CONFIG\_DEFAULT\_DEVICE\_TREE="s5pc1xx-goni"  CONFIG\_SYS\_CONSOLE\_IS\_IN\_ENV=y  CONFIG\_HUSH\_PARSER=y  CONFIG\_SYS\_PROMPT="Goni # " | 修改后：  CONFIG\_ARM=y  CONFIG\_ARCH\_S5PC1XX=y  CONFIG\_TARGET\_SMDKV210=y  CONFIG\_DEFAULT\_DEVICE\_TREE="s5pc1xx-goni"  CONFIG\_SYS\_CONSOLE\_IS\_IN\_ENV=y  CONFIG\_HUSH\_PARSER=y  CONFIG\_SYS\_PROMPT="Goni # " |

②：配置头文件

cp include/configs/s5p\_goni.h include/configs/smdkv210.h

③：拷贝一份单板相关的备份，将来我们在这里修改

cp board/samsung/goni/ board/samsung/smdkv210 -arf

修改smdkv210目录下面的Kconfig文件

|  |  |
| --- | --- |
| 修改前：  if TARGET\_S5P\_GONI  config SYS\_BOARD  default "goni"  config SYS\_VENDOR  default "samsung"  config SYS\_SOC  default "s5pc1xx"  config SYS\_CONFIG\_NAME  default "s5p\_goni"  endif | 修改后：  if TARGET\_SMDKV210  config SYS\_BOARD  default "smdkv210"  config SYS\_VENDOR  default "samsung"  config SYS\_SOC  default "s5pc1xx"  config SYS\_CONFIG\_NAME  default "smdkv210"  endif |

SYS\_BOARD代表board/目录下的板子类型 （对应第①步）

SYS\_VENDOR代表board/（代表 board/samsung/smdkv210文件夹）

SYS\_SOC代表SOC类型（因为这里没有我们的s5pv210，只能维持s5pc1xx）

SYS\_CONFIG\_NAME代表include/configs/目录下的头文件 （对应第②步）

④：配置文件添加

因为我们的SOC是固定的s5pc1xx，所以到这个目录向下增加我们的smdkv210单板配置

vim arch/arm/mach-s5pc1xx/Kconfig

|  |  |
| --- | --- |
| 修改前：  if ARCH\_S5PC1XX  choice  prompt "S5PC1XX board select"  optional  config TARGET\_S5P\_GONI  bool "S5P Goni board"  select OF\_CONTROL  config TARGET\_SMDKC100  bool "Support smdkc100 board"  select OF\_CONTROL  endchoice  config SYS\_SOC  default "s5pc1xx"  source "board/samsung/goni/Kconfig"  source "board/samsung/smdkc100/Kconfig"  endif | 修改后：  if ARCH\_S5PC1XX  choice  prompt "S5PC1XX board select"  optional  config TARGET\_S5P\_GONI  bool "S5P Goni board"  select OF\_CONTROL  config TARGET\_SMDKV210  bool "SMDKV210 board"  select OF\_CONTROL  config TARGET\_SMDKC100  bool "Support smdkc100 board"  select OF\_CONTROL  endchoice  config SYS\_SOC  default "s5pc1xx"  source "board/samsung/smdkv210/Kconfig"  source "board/samsung/goni/Kconfig"  source "board/samsung/smdkc100/Kconfig"  endif |

拷贝一份GONI的config文件，改成我们的TARGET\_SMDKV210（注意这里的名字和第①步的要一致）

拷贝一份source，改成我们的smdkv210的目录。

**先清理掉之前的配置在编译：**

make clean

make distclean

make smdkv210\_defconfig

make

编译通过，生成了对应的uboot.bin文件说我目前为止我们的修改都ok

1. 这几个宏，在make smdkv210\_defconfig的时候，会生成对应的CONFIG\_对应项的宏，在.config文件中

CONFIG\_SYS\_ARCH="arm"

CONFIG\_SYS\_CPU="armv7"

CONFIG\_SYS\_SOC="s5pc1xx"

CONFIG\_SYS\_VENDOR="samsung"

CONFIG\_SYS\_BOARD="smdkv210"

CONFIG\_SYS\_CONFIG\_NAME="smdkv210"

（2）这几个宏，分别指定了编译时，选择的路径

SYS\_ARCH指定编译路径为arch/<CONFIG\_SYS\_ARCH>

SYS\_CPU指定编译路径为arch/<CONFIG\_SYS\_ARCH>/cpu/<CONFIG\_SYS\_CPU>

SYS\_SOC指定编译路径为arch/<CONFIG\_SYS\_ARCH>/cpu/<CONFIG\_SYS\_CPU>/<CONFIG\_SYS\_SOC>.

SYS\_VENDOR指定编译路径为 board/<CONFIG\_SYS\_VENDOR>/common//\*或board/<CONFIG\_SYS\_VENDOR>/<CONFIG\_SYS\_BOARD>/\*

SYS\_BOARD指定编译路径为board/<CONFIG\_SYS\_BOARD>/\*或如果指定SYS\_VENDOR，则为board/<CONFIG\_SYS\_VENDOR>/<CONFIG\_SYS\_BOARD>//\*

SYS\_CONFIG\_NAME如果定义，include/config.h包含include/configs/<CONFIG\_SYS\_CONFIG\_NAME>.h

**二、从入口分析流程**

**1，连接脚本**

**arch/arm/cpu/u-boot.lds**

**2,中断向量表的存放**

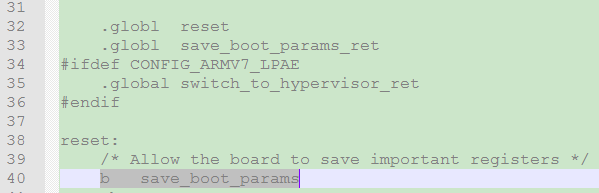
可以看到真正的入口是\_start，真正的.text段是从.vectors开始的。从字面上就能看出.vector段是中断向量表的存放处。下面u-boot源码所示是根据CPU的硬件特性所定义的中断向量表：

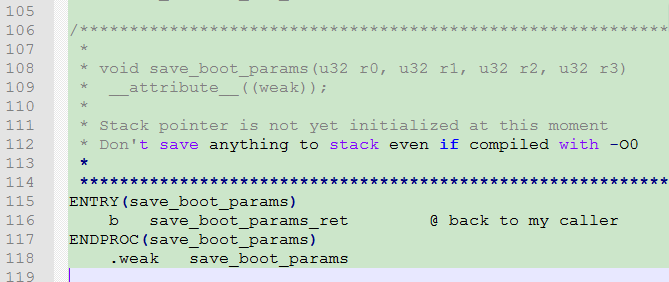
arch/arm/lib/vectors.S

**3,初始代码开始**

**arch/arm/cpu/armv7/start.S**

从异常向量表的reset跳转过来





**此处重点介绍两个宏：ENTRY和ENDPROC。#include <linux/linkage.h>**

**ENTRY：**

**#ifndef ASM\_NL**

**#define ASM\_NL ;**

**#endif**

**#ifndef \_\_ALIGN**

**#define \_\_ALIGN .align 4**

**#endif**

**#define ALIGN \_\_ALIGN**

**#ifdef \_\_STDC\_\_**

**#define SYMBOL\_NAME\_LABEL(X) X##:**

**#else**

**#define SYMBOL\_NAME\_LABEL(X) X:**

**#endif**

**#define SYMBOL\_NAME(X) X**

**#define LENTRY(name) \**

**ALIGN ASM\_NL \**

**SYMBOL\_NAME\_LABEL(name)**

**#define ENTRY(name) \**

**.globl SYMBOL\_NAME(name) ASM\_NL \**

**LENTRY(name)**

**#ifndef END**

**#define END(name) \**

**.size name, .-name**

**#endif**

**#ifndef ENDPROC**

**#define ENDPROC(name) \**

**.type name STT\_FUNC ASM\_NL \**

**END(name)**

**#endif**

**eg：ENTRY(save\_boot\_params)**

**b save\_boot\_params\_ret @ back to my caller**

**展开ENTRY,ENDPROC：**

**.globl save\_boot\_params;**

**.align 4;**

**save\_boot\_params:**

**b save\_boot\_params\_ret**

可以发现执行流程是这样的：

reset---->>>save\_boot\_params（什么都没干，跳回reset后面） ----->>> save\_boot\_params\_ret----->>关中断----->设置异常向量表的基地址------->cpu\_init\_cp15（无效icache，dcache等）------->cpu\_init\_crit（继续调用lowlevel）

因为后面的lowlevel是要跳转到别的文件进行初始化了，所以我们在它前面增加代码先进行调试。

在start.S的最后面增加led1的调试代码

gpio\_out:

ldr r11, =0xe0200240

ldr r12, =0x01111000

str r12, [r11]

ldr r11, =0xe0200244

ldr r12, =0xff

str r12, [r11]

mov pc, lr

.globl led1\_on

led1\_on:

ldr r11, =0xe0200244

ldr r12, [r11]

bic r12, r12,#(1<<3)

str r12, [r11]

mov pc, lr

.globl led1\_off

led1\_off:

ldr r11, =0xe0200244

ldr r12, [r11]

orr r12, r12,#(1<<3)

str r12, [r11]

mov pc, lr

在执行跳转代码前面让led灯点亮，确定执行到这里的代码都正确。

bl gpio\_out

bl led1\_on

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

#ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT

bl cpu\_init\_cp15

#ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT\_ONLY

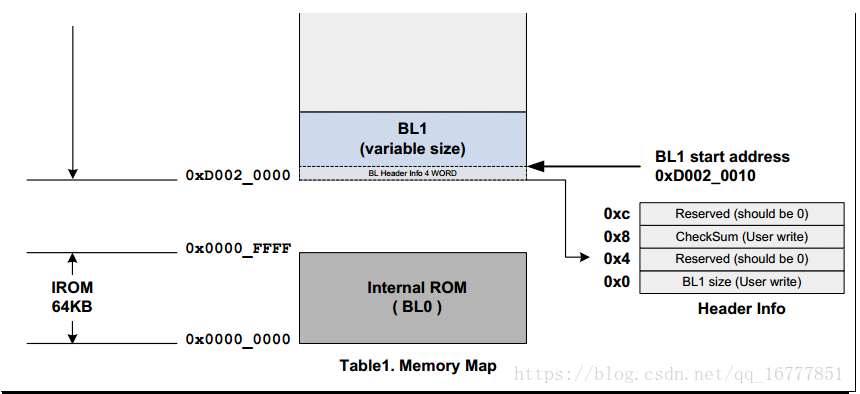
bl cpu\_init\_crit

#endif

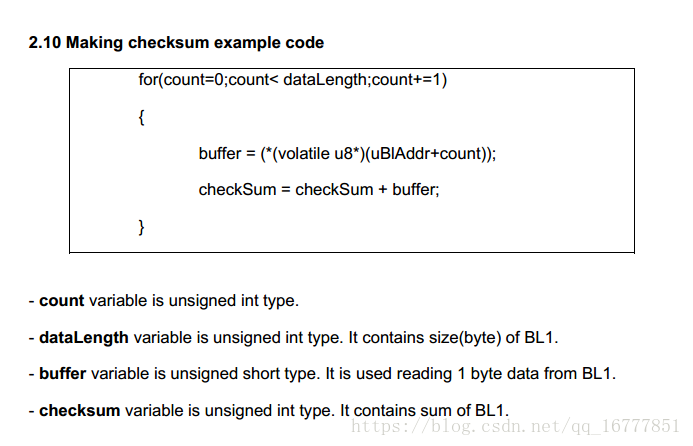
#endif

之后make编译生成u-boot.bin文件烧写，测试

因为s5pv210通过sd卡启动，需要内部的irom代码校验的。所以我们先编写校验代码。



校验算法applacation note也给出了



我这边为了方便，做成16k的校验，代码如下

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BL1\_SIZE (16\*1024)

#define BL\_HEADER\_INFO " " //16个空格

#define BL\_HEADER\_SIZE 16

int main(int argc,char \*argv[])

{

FILE \*fp = NULL;

int file\_len = 0;

unsigned char \*buff = NULL,data = 0; /\* 这里要定义成unsigned 否则,否则data加出来的会有负数,导致校验和错误,这里用三星官方给的有点小坑 \*/

int buf\_len = 0;

unsigned int checksum = 0,count;

int i = 0;

unsigned int nbytes = 0;

/\* 输入三个参数,分别是自己可执行文件,uboot.bin,输出的16k文件 \*/

if(3 != argc)

{

printf("argc paramenter number is %d,not 3\n",argc);

printf("%s <source file> <destination file>\n",argv[0]);

return -1;

}

buff = (char \*)malloc(BL1\_SIZE);

if(NULL == buff)

{

printf("malloc error\n");

return -1;

}

/\* clear the buff memary \*/

memset(buff, 0, BL1\_SIZE);

fp = fopen(argv[1],"rb");

if(NULL == fp)

{

printf("open source file error\n");

return -1;

}

/\* 定位到文件尾 \*/

fseek(fp,0,SEEK\_END);

/\* 得到文件长度 \*/

file\_len = ftell(fp);

/\* 定位到文件头 \*/

fseek(fp,0,SEEK\_SET);

/\* get write BL1 size \*/

count = (file\_len<(BL1\_SIZE- BL\_HEADER\_SIZE)) ? file\_len : (BL1\_SIZE - BL\_HEADER\_SIZE);

/\* write BL1 header info \*/

memcpy(buff, BL\_HEADER\_INFO, BL\_HEADER\_SIZE);

/\* read fp file count size of buff +... \*/

nbytes = fread(buff + BL\_HEADER\_SIZE, 1, count , fp);

if(count != nbytes)

{

printf("fread %s faile\n ",argv[1]);

free(buff);

fclose(fp);

return -1;

}

fclose(fp);

fp = NULL;

/\* calculate checksum \*/

for(i = 0; i < count; i++ )

{

data = \*(volatile unsigned char \*)(buff + BL\_HEADER\_SIZE + i);

checksum += data;

}

\*(volatile unsigned int \*)buff = BL1\_SIZE;

\*(volatile unsigned int \*)(buff + 8) = checksum;

fp = fopen(argv[2],"wb");

if(NULL == fp)

{

printf("open aim file error\n");

return -1;

}

nbytes = fwrite(buff, 1, BL1\_SIZE, fp);

if(BL1\_SIZE != nbytes)

{

printf("write aim file faile\n");

free(buff);

fclose(fp);

return -1;

}

fclose(fp);

free(buff);

return 0;

}

生成16k制作工具

gcc s5pv210\_image.c -o make-16k

生成16k烧写文件

./make-16k u-boot.bin u-boot-16k.bin

烧写sd卡

sudo dd iflag=dsync oflag=dsync if=./u-boot-16k.bin of=/dev/sdb seek=1

发现led1确实被点亮了，说明到目前为止的程序运行正确。

### **lowlevel\_init**

lowlevel\_init一般是由板级代码自己实现的.但是对于某些平台来说，也可以使用通用的lowlevel\_init，其定义在arch/arm/cpu/lowlevel\_init.S中。以goni为例，在移植goni的过程中，就需要在board/samsung/goni下，也就是板级目录下面创建lowlevel\_init.S，在内部实现lowlevel\_init。（其实只要实现了lowlevel\_init了就好，没必要说在哪里是实现，但是通常规范都是创建了lowlevel\_init.S来专门实现lowlevel\_init函数）

在查看lowlevel\_init函数之前，先找到代码在哪里，搜了一下发现先关的两个地方都有。没办法，只有查看Makefile

IMG_256

IMG_256

先看armv7下面的，可以知道这些系列的CPU我们都没定义，所以 它们是等于 NULL的，即不会包含lowlevel\_init.o

ifneq ($(CONFIG\_AM43XX)$(CONFIG\_AM33XX)$(CONFIG\_OMAP44XX)$(CONFIG\_OMAP54XX)$(CONFIG\_TEGRA)$(CONFIG\_MX6)$(CONFIG\_MX7)$(CONFIG\_TI81XX)$(CONFIG\_AT91FAMILY)$(CONFIG\_ARCH\_SUNXI)$(CONFIG\_ARCH\_SOCFPGA)$(CONFIG\_LS102XA),)

ifneq ($(CONFIG\_SKIP\_LOWLEVEL\_INIT),y)

obj-y += lowlevel\_init.o

endif

endif

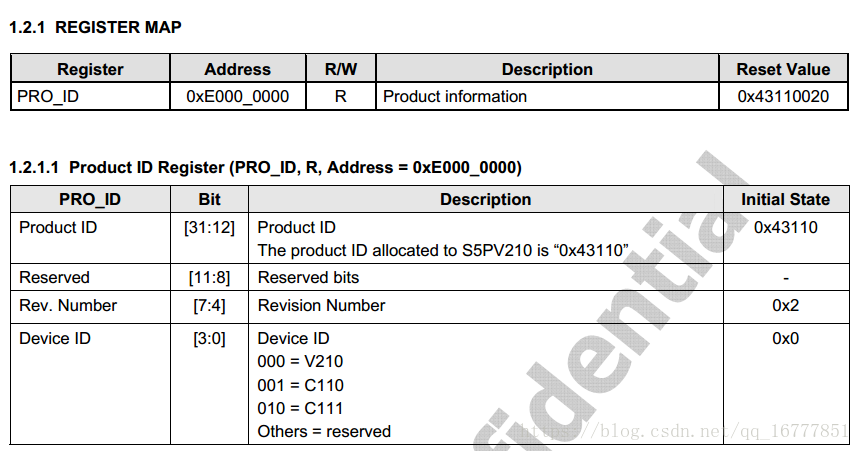
接下来看goni下面的，直接包含了，没有任何条件（即：只要是goni的板子，必然包含它下面的lowlevel.o）

obj-y := goni.o onenand.o

obj-y += lowlevel\_init.o

在lowlevel\_init中，要实现如下：   
\* 检查一些复位状态   
\* 关闭看门狗   
\* 系统时钟的初始化   
\* 内存、DDR的初始化   
\* 串口初始化（可选）

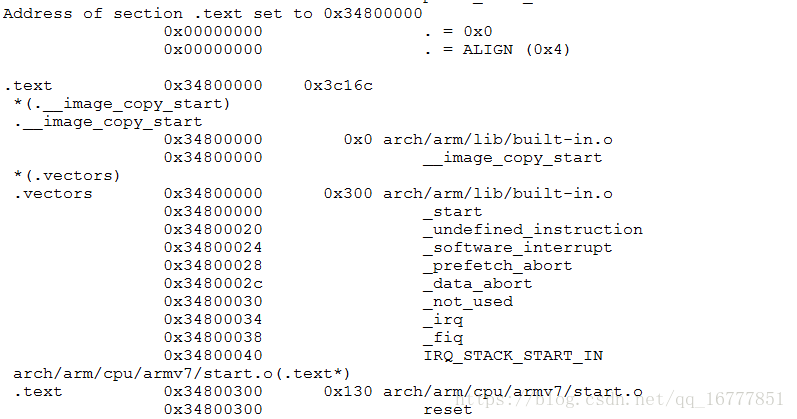
下面要用到读pro\_id来区分产品类型，我们s5pv210的如下图



可以看一下board/samsung/goni/lowlevel\_init.S的代码

在goni.h头文件定义了，text段的基地址0x34800000

IMG_256



在最终生成的u-boot.map表中查看lowlevel的地址，发现在16k以内，所以不用改这里。

在uart2的初始化的末尾添加代码，在串口发送一个字符'O'

ldr r1, =0xe2900820

ldr r2, =0x4f

str r2, [r1]

发现串口2可以打印出O字符

在start.S的  调用\_main之前调用，下面代码，发现不能正确打印'K'

ldr r1, =0xe2900820

ldr r2, =0x4b

str r2, [r1]