[置顶] 亮仔移植u-boot系列之-- S3c2440在最新版本U-boot-2015.10移植(支持SPL模式启动) -- 2

标签： u-boot 移植

2015年12月15日 11:58:17 676人阅读 评论(0) 收藏 举报

分类：

U-boot2015.10移植（1）

这章讲解BL1的移植方法.

打开Start.S文件,在Cpu上电后的第一条命令b reset后进入这个文件执行后续操作,和老的u-boot版本一样执行了以下步骤

1:切换到svc32模式,关闭icache和dcache

2:关中断

3:关狗

4:设置时钟分频比

5:设置各个Bank

6:搬移BL2的代码从Nand Flash到SDRAM

7:跳转到_main执行

显然过程1-过程5只需要在BL1阶段执行1次，BL2直接在b reset后直接跳转到过程6执行.(++表示添加的)

```cpp
++ #if defined(CONFIG_SPL_BUILD)
    /*
     * set the cpu to SVC32 mode
     */
    mrs  r0, cpsr
    bic  r0, r0, #0x1f
    orr  r0, r0, #0xd3
    msr  cpsr, r0
    .
    .
    .
    .
```

```
    .
++#else //#if defined(CONFIG_SPL_BUILD)
    ldr     r0, =0x56000054
    mov     r1, #0xffffff8f
    str     r1, [r0]    //Set Led1 2 3 On
++#endif

    bl _main
```

由于2440和2410时钟设置上有区别，因此我将时钟设置这段代码注释了，因为俺的汇编比较渣，所以针对2440的时钟初始化工作我移到设置完sp后跳转到c代码yl_clock_init去执行.

```cpp
/* FCLK:HCLK:PCLK = 1:2:4 */
/* default FCLK is 120 MHz ! */
-- ldr  r0, =CLKDIVN
-- mov  r1, #3
-- str  r1, [r0]
```

还要注意关闭中断，将ldr r1， =0x3ff修改为0x7fff

```cpp
ldr r1, =0x7ff
ldr r0, =INTSUBMSK
str r1, [r0]
```

_main在arch\arm\lib目录下的Crt0.S文件内，我在yl2440.h目录下设置了

#define CONFIG_SPL_STACK 0x1000

即SPL模式下的堆栈指针指向Cpu的Steppingstone的最顶部.

```cpp
#if defined(CONFIG_SPL_BUILD) && defined(CONFIG_SPL_STACK)
dr  sp, =(CONFIG_SPL_STACK)
#else
```

```
4.  dr  sp, =(CONFIG_SYS_INIT_SP_ADDR)
5.  #endif
```

当PC指针指向这个点，BL1的任务只剩下之前提到的step4，6，7了，在bl board_init_f处做如下处理，即SPL模式下不需要进入

board_init_f板块第一次初始化函数，修改为bl yl_clock_init通过C语言的方式初始化时钟，具体yl_clock_init()功能实现网上很多，本文就不多讲了.

```
[cpp]
1.  #if defined(CONFIG_SPL_BUILD)
2.        bl  yl_clock_init
3.       /* Read u-boot from Nandflash to SDRAM address $CONFIG_SYS_TEXT_BASE */
4.        ldr r0, =BL2_MTD_OFFSET
5.        ldr r1, =CONFIG_SYS_TEXT_BASE
6.        ldr r2, =BL2_MTD_LENTH
7.
8.        bl yl_copy_code_from_nand_to_sdram
9.        ldr pc, =CONFIG_SYS_TEXT_BASE
10.   #else
11. l   board_init_f
12.   #endif
```

在yl2440.h定义上面上个宏:

```
[cpp]
1.  #define BL2_MTD_OFFSET          0x20000  /* BL2: 128K nand offset - 1Block */
2.  #define BL2_MTD_LENTH           0x100000 /* BL2: 1M nand lenth */
3.  #define CONFIG_SYS_TEXT_BASE    0x30008000
```

BL2_MTD_OFFSET因为我板子上的Nand Flash是2K/Page，总共64个Page，那么整个Block大小是128K.规划BL1.bin存放在Block0,BL2.bin存放在Block1，BL2_MTD_LENTH表示了BL2占用从block1起头的1M空间.

CONFIG_SYS_TEXT_BASE是BL2链接时在SDRAM的运行地址,因此只要yl_copy_code_from_nand_to_sdram函数将Nand Flash所在的BL2位子搬移到SDRAM的CONFIG_SYS_TEXT_BASE处,再将pc指针指向这个BL2代码运行的首地址,则BL1就一去不复返了,将控制权交给了BL2.

BL2无非需要实现以下几个任务:

1：能通过串口打印相关信息，通过串口实现有关功能

2：能读写nand flash，将内核镜像和根文件系统写道某个nand flash特定分区内

3：能通过网线实现tftp传输相关文件

4：能引导linux内核

下一章将实力分析BL2代码的重定位和串口功能的实现.