

[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) -- 3

标签： [u-boot](#) [移植](#)

2015年12月29日 15:12:12

747人阅读

评论(0)

收藏

举报

分类：

[U-boot2015.10移植 \(1\)](#)

这章讲解BL2阶段的代码重定位和串口功能的实现.

在BL1阶段执行最后一句代码:

```
ldr pc, =CONFIG_SYS_TEXT_BASE
```

此时PC指针重新指向b reset命令，第二阶段无需重新做cpu，时钟之类的初始化,直接执行BL _main命令设置BL2模式下的sp.我将BL2模式下DDR的分布设置如下:

[cpp]  

```
1. -----0x30000000 + 64M
2. u-boot.bin
3. -----0x30008000
4. .
5. .(not use)
```

[cpp]  

```
1. -----0x30001000
2. global data
3. -----0x30001000 - 176
4. GD
```

[cpp]  

```
1. -----0x30001000 - 176 - GD_SIZE
2. .SP.
3. .
4. -----0x30000000</span>
```

设置完堆栈后执行

BL board_init_r, 下面是详细分析这段函数的相关代码和注释：

```
[cpp]
1.  gd->mon_len = (ulong)&__bss_end - (ulong)_start; //记录代码段起始地址到bss_end总大小
2.  ↓
3.  ↓
4.  init_baud_rate(); //设置串口波特率为115200
5.  ↓
6.  ↓
7.  serial_init(); //设置uart时钟, 新版的u-boot支持2440的时钟配置, 只需将CONFIG_S3C2440 = 1; 不用像u-boot-1.1.6用
   新写代码
8.  ↓
9.  ↓
10. get_current()->start(); //相当于调用了serial_init_dev(0);函数, 下面有分析
11. ↓
12. ↓
13. dev = default_serial_console(); //我们需要调用serial_s3c24x0.c中的函数, 查看MakeFile发现obj-$(CONFI
   3C24X0_SERIAL) += serial_s3c24x0.o
14. //因此需要在y12440.h中定义CONFIG_S3C24X0_SERIAL
15. ↓
16. return &s3c24xx_serial0_device; //我们默认使用serial0, 因此需要在y12440.h中#define C
   _SERIAL1 1
17. 而struct serial_device s3c24xx_serial0_device = INIT_S3C_SERIAL_STRUCTURE(0, "s3ser0"); 将
   T_S3C_SERIAL_STRUCTURE展开即
18. struct serial_device s3c24xx_serial0_device =
19. {
20.     .name   = "s3ser0",      \
21.     .start  = s3serial0_init, \
22.     .stop   = NULL,          \
23.     .setbrg = s3serial0_setbrg, \
24.     .getc  = s3serial0_getc,  \
25.     .tstc  = s3serial0_tstc,  \
26.     .putc  = s3serial0_putc,  \
27.     .puts  = s3serial0_puts,  \ /* printf()--->>> dev->puts() */
28. }
29. ↓
30. ↓
31. gd->ram_size = PHYS_SDRAM_1_SIZE; //在y12440.h中定义成64M
```

```
32. | ↓
33. | ↓
34. | setup_dest_addr(); //首先将gd->relocaddr设置为0x30000000+0x4000000(64M)
35. |     ↓
36. |     ↓
37. |     gd->relocaddr = 0x30000000+0x4000000;
38. | ↓
39. | ↓
40. | reserve_uboot
41. |     ↓
42. |     ↓
43. |     gd->relocaddr -= gd->mon_len; //relocaddr指向代码重定位的地址
44. |     gd->start_addr_sp = gd->relocaddr;
45. | ↓
46. | ↓
47. | reserve_malloc()
48. |     ↓
49. |     ↓
50. |     gd->start_addr_sp = gd->start_addr_sp - TOTAL_MALLOC_LEN;
51. | ↓
52. | ↓
53. | reserve_board
54. |     ↓
55. |     ↓
56. |     gd->start_addr_sp -= sizeof(bd_t);
57. |     gd->bd = gd->start_addr_sp;
58. |     将bd区域清0
59. | ↓
60. | ↓
61. | reserve_global_data()
62. |     ↓
63. |     ↓
64. |     gd->start_addr_sp -= sizeof(gd_t);
65. |     gd->new_gd = gd->start_addr_sp ;
66. | ↓
67. | ↓
68. | reserve_stacks()
69. |     ↓
70. |     ↓
```

```

71. |         gd->start_addr_sp -= 16;
72. |         gd->start_addr_sp &= ~0xf;
73. |         ↓
74. |         ↓
75. |         setup_reloc();
76. |         ↓
77. |         ↓
78. |         gd->reloc_off = gd->relocaddr - CONFIG_SYS_TEXT_BASE; //gd->reloc_off = gd->relocaddr - 0x30008000;
79. |         memcpy(gd->new_gd, (char *)gd, sizeof(gd_t)); // 0x30001000-176-GD_SIAZE ---->>> gd->new_gd搬移

```

需要说明的是在该函数调用的board_early_init_f()中将设置PLL相关部分全部注释掉，因为之前的时钟设置已经在BL1阶段设置好了。

board_init_r退出重新设置新的sp指针：LDR SP [r9 #GD_START_ADDR_SP] /* sp = gd->start_addr_sp这个值已经之前设置好了 */

从新的gd地址找到新的gd->relocaddr即代码重定位的地址，调用b_relocate_code后，此时新的DDR分布如下：

```

1. | -----0x304000000 sdram end(64M)
2. | .
3. | .
4. | <strong>u-boot.bin + 动态链接库(不是很清楚，估计和相对地址有关)</strong>
5. | .
6. | -----gd->relocaddr 0x304000000 - (gd->mon_len)
7. | malloc_lenth
8. | -----0x304000000 - (gd->mon_len + malloc_lenth)
9. | bd(记录板子的信息，如bd->baudrate)
10. | -----0x304000000 - (gd->mon_len + malloc_lenth + bd)
11. | gd(记录初始化参数,如gd->start_addr_sp)
12. | -----0x304000000 - (gd->mon_len + malloc_lenth + bd + gd)
13. | 16Byte
14. | -----gd->start_addr_sp 0x304000000 - (gd->mon_len + malloc_lenth + bd + gd + 16)
15. | sp
16. | -----
17. | .
18. | .(not use)
19. | -----

```

20.	kernel(2M)
21.	-----0x30800000
22.	.
23.	.boot param
24.	-----0x30000100
25.	(not use)
26.	-----0x30000000

由于在4K IRAM内的中断向量表是BL1阶段的向量表，因而需要执行搬移BL2的向量表到4K IRAM内

bl relocate_vectors

最后执行

ldr pc, =board_init_r

进入板子的第二阶段初始化工作，此时在串口已经可以显示U-BOOT2015.10的信息了.串口功能调试完毕.

总结：

U-Boot第二阶段(BL2)最开始将位于用户定义的代码链接地址从0x30008000重定位到SDRAM顶端，并生成了一个动态链接库rel_dyn_start到rel_dyn_end.并重新设置sp，拷贝就得bd数据到新的bd区域，重定位向量表，最终执行第二阶段的板卡初始化.

这个阶段代码重定位完毕，串口已经可以跑起来了.

我们知道U-boot终极目的是引导内核，因此我们还需要以下功能：

功能1：u-boot支持tftp下载内核到DDR中，因此需要配置DM9000网卡

功能2：u-boot需要支持Nand读写，将内核及环境变量写到Nand Flash的相关位置，上电后将镜像读到DDR

功能3：u-boot能将位于DDR的内核进行引导，最终U-boot寿命结束，控制权交给内核

本章结束，下一章将介绍如何配置DM9000,实现网卡的下载功能.

遗留问题：动态链接库、中断重定位的细节