u-boot启动的第一阶段 - 豆豆男孩 - 博客园

u-boot的第一阶段的任务是完成 部分硬件的设置:

1.设置CPU工作在管理员模式

```
1 mrs    r0,cpsr
2 bic    r0,r0,#0x1f
3 orr    r0,r0,#0xd3
4 msr    cpsr,r0
```

2.关闭看门狗

```
 1 /* turn off the watchdog */
 2 #if defined(CONFIG_S3C2400)
 3 # define pWTCON         0x15300000
 4 # define INTMSK         0x14400008    /* Interupt-Controller base addresses */
 5 # define CLKDIVN    0x14800014    /* clock divisor register */
 6 #elif defined(CONFIG_S3C2410)
 7 # define pWTCON         0x53000000
 8 # define INTMOD     0X4A000004
 9 # define INTMSK         0x4A000008    /* Interupt-Controller base addresses */
10 # define INTSUBMSK    0x4A00001C
11 # define CLKDIVN    0x4C000014    /* clock divisor register */
12 #endif
13
14 #if defined(CONFIG_S3C2400) || defined(CONFIG_S3C2410)
15     ldr     r0, =pWTCON
16     mov     r1, #0x0
17     str     r1, [r0]
```

3.禁止所有中断

```
1     mov    r1, #0xffffffff
2     ldr    r0, =INTMSK
3     str    r1, [r0]
4 # if defined(CONFIG_S3C2410)
5     ldr    r1, =0x3ff
6     ldr    r0, =INTSUBMSK
7     str    r1, [r0]
8 # endif
```

4.设置时钟

```
 1 #define S3C2440_MPLL_400MHZ     ((0x5c<<12)|(0x01<<4)|(0x01))
 2 #define S3C2440_UPLL_48MHZ      ((0x38<<12)|(0x02<<4)|(0x02))
 3 #define S3C2440_CLKDIV          (0x05) // | (1<<3))    /* FCLK:HCLK:PCLK = 1:4:8, UCLK = UPLL/2 */
 4
 5     ldr r1, =CLKDIVN
 6     mov r2, #S3C2440_CLKDIV
 7     str r2, [r1]
 8
 9     mrc p15, 0, r1, c1, c0, 0       // read ctrl register
10     orr r1, r1, #0xc0000000         // Asynchronous
11     mcr p15, 0, r1, c1, c0, 0       // write ctrl register
12
13     ldr r0,=LOCKTIME
14     ldr r1,=0xffffff
15     str r1,[r0]
16     // delay
17     mov     r0, #0x200
18 1:  subs    r0, r0, #1
```

```
19     bne     1b
20
21     // Configure MPLL
22     ldr r0,=MPLLCON
23     ldr r1,=S3C2440_MPLL_400MHZ
24     str r1,[r0]
25     // delay
26     mov     r0, #0x200
27 1:  subs    r0, r0, #1
28     bne     1b
29
30     //Configure UPLL
31     ldr     r0, =UPLLCON
32     ldr     r1, =S3C2440_UPLL_48MHZ
33     str     r1, [r0]
34     // delay
35     mov     r0, #0x200
36 1:  subs    r0, r0, #1
37     bne     1b
```

5.使能SDRAM

```
 1 adr    r0, _start        /* r0 <- current position of code   */
 2     ldr    r1, _TEXT_BASE        /* test if we run from flash or RAM */
 3     cmp     r0, r1                 /* don't reloc during debug         */
 4     blne    cpu_init_crit
 5
 6
 7
 8 cpu_init_crit:
 9     /*
10      * flush v4 I/D caches
11      */
12     mov    r0, #0
13     mcr    p15, 0, r0, c7, c7, 0    /* flush v3/v4 cache */
14     mcr    p15, 0, r0, c8, c7, 0    /* flush v4 TLB */
15
16     /*
17      * disable MMU stuff and caches
18      */
19     mrc    p15, 0, r0, c1, c0, 0
20     bic    r0, r0, #0x00002300    @ clear bits 13, 9:8 (--V- --RS)
21     bic    r0, r0, #0x00000087    @ clear bits 7, 2:0 (B--- -CAM)
22     orr    r0, r0, #0x00000002    @ set bit 2 (A) Align
23     orr    r0, r0, #0x00001000    @ set bit 12 (I) I-Cache
24     mcr    p15, 0, r0, c1, c0, 0
25
26     /*
27      * before relocating, we have to setup RAM timing
28      * because memory timing is board-dependend, you will
29      * find a lowlevel_init.S in your board directory.
30      */
31     mov    ip, lr
32     bl    lowlevel_init
33     mov    lr, ip
34     mov    pc, lr
35
36
37
38 _TEXT_BASE:
39     .word    TEXT_BASE
40
41 .globl lowlevel_init
42 lowlevel_init:
43     /* memory control configuration */
44     /* make r0 relative the current location so that it */
45     /* reads SMRDATA out of FLASH rather than memory ! */
46     ldr     r0, =SMRDATA
47     ldr    r1, _TEXT_BASE
48     sub    r0, r0, r1
49     ldr    r1, =BWSCON    /* Bus Width Status Controller */
50     add     r2, r0, #13*4
51 0:
52     ldr     r3, [r0], #4
53     str     r3, [r1], #4
54     cmp     r2, r0
```

```
55     bne     0b
56
57     /* everything is fine now */
58     mov    pc, lr
59
60     .ltorg
61 /* the literal pools origin */
62
63 SMRDATA:
64     .word (0+(B1_BWSCON<<4)+(B2_BWSCON<<8)+(B3_BWSCON<<12)+(B4_BWSCON<<16)+(B5_BWSCON<<20)+(B6_BWSCON<<24)+(B7_BWSCON<<28))
65     .word ((B0_Tacs<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+(B0_Tah<<4)+(B0_Tacp<<2)+(B0_PMC))
66     .word ((B1_Tacs<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+(B1_Tah<<4)+(B1_Tacp<<2)+(B1_PMC))
67     .word ((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(B2_PMC))
68     .word ((B3_Tacs<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+(B3_Tah<<4)+(B3_Tacp<<2)+(B3_PMC))
69     .word ((B4_Tacs<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+(B4_Tah<<4)+(B4_Tacp<<2)+(B4_PMC))
70     .word ((B5_Tacs<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+(B5_Tah<<4)+(B5_Tacp<<2)+(B5_PMC))
71     .word ((B6_MT<<15)+(B6_Trcd<<2)+(B6_SCAN))
72     .word ((B7_MT<<15)+(B7_Trcd<<2)+(B7_SCAN))
73     .word ((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)+REFCNT)
74     .word 0xb1
75     .word 0x30
76     .word 0x30
```

6.设置栈

```
1 stack_setup:
2     ldr    r0, _TEXT_BASE        /* upper 128 KiB: relocated uboot   */
3     sub    r0, r0, #CFG_MALLOC_LEN    /* malloc area                      */
4     sub    r0, r0, #CFG_GBL_DATA_SIZE /* bdinfo                        */
5
6 #ifdef CONFIG_USE_IRQ
7     sub    r0, r0, #(CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ)
8 #endif
9     sub    sp, r0, #12        /* leave 3 words for abort-stack    */
```

7.代码重定位

```
 1 relocate:                /* relocate U-Boot to RAM        */
 2     adr    r0, _start        /* r0 <- current position of code   */
 3     ldr    r1, _TEXT_BASE        /* test if we run from flash or RAM */
 4     cmp     r0, r1                /* don't reloc during debug         */
 5     beq     clear_bss
 6
 7     ldr    r2, _armboot_start
 8     ldr    r3, _bss_start
 9     sub    r2, r3, r2        /* r2 <- size of armboot            */
10 #if 1
11     bl  CopyCode2Ram    /* r0: source, r1: dest, r2: size */
12 #else
13     add    r2, r0, r2        /* r2 <- source end address         */
14
15 copy_loop:
16     ldmia    r0!, {r3-r10}        /* copy from source address [r0]    */
17     stmia    r1!, {r3-r10}        /* copy to   target address [r1]    */
18     cmp    r0, r2            /* until source end addreee [r2]    */
19     ble    copy_loop
```

```
 1 int CopyCode2Ram(unsigned long start_addr, unsigned char *buf, int size)
 2 {
 3     unsigned int *pdwDest;
 4     unsigned int *pdwSrc;
 5     int i;
 6
 7     if (bBootFrmNORFlash())
 8     {
 9         pdwDest = (unsigned int *)buf;
10         pdwSrc  = (unsigned int *)start_addr;
```

```
11          /* 从 NOR Flash启动 */
12          for (i = 0; i < size / 4; i++)
13          {
14              pdwDest[i] = pdwSrc[i];
15          }
16          return 0;
17      }
18      else
19      {
20          /* 初始化NAND Flash */
21          nand_init_ll();
22          /* 从 NAND Flash启动 */
23          nand_read_ll_lp(buf, start_addr, (size + NAND_BLOCK_MASK_LP)&~(NAND_BLOCK_MASK_LP));
24          return 0;
25      }
26 }
```

8.清除BSS段

```
clear_bss:
    ldr     r0, _bss_start        /* find start of bss segment        */
    ldr     r1, _bss_end        /* stop here                        */
    mov      r2, #0x00000000        /* clear                            */

clbss_l:str     r2, [r0]        /* clear loop...                    */
    add     r0, r0, #4
    cmp     r0, r1
    ble     clbss_l
```

第一阶段到这结束，后面经过一个C函数跳转至u-boot启动的第二阶段

```
    ldr pc, _start_armboot

_start_armboot: .word start_armboot
```