UBOOT启动流程_WY_study的博客-CSDN博客_uboot启动流程

转载请注明地址：**http://blog.csdn.net/zsy2020314/article/details/9824035**

1.关于启动流程

1.1 启动阶段分为3个，bl0，bl1，bl2。下面只是就功能方面对它们做说明，实际设计的时候，也许会对其具体功能做出调整，也就是说，这几个阶段的划分是就功能而言的，不能看得太死。

bl0：出厂的时候就固化在irom中一段代码，主要负责拷贝8kb的bl1到s5pv210的一个96kb大小内部sram(Internal SRAM)中运行。值得注意的是s5pv210的Internal SRAM支持的bl1的大小可以达到16kb，容量的扩增是为了适应bootloder变得越来复杂而做的。虽然如此，但目前我们制作出来的bl1的大小仍然可以保持在8kb以内，同样能满足需求。

bl1：u-boot的前8kb代码(s5pv210也支持16kb大小，原因上一点提过了)，除了初始化系统时钟和一些基本的硬件外，主要负责完成代码的搬运工作(我设计成搬运bl1+bl2，而不仅仅是bl2)，也就是将完整的u-boot代码(bl1+bl2)从nand flash或者mmcSD等的存储器中读取到内存中，然后跳转到内存中运行u-boot。

bl2：完成全面的硬件初始化和加载OS到内存中，接着运行OS。

上述几个阶段的流程描述在s5pv210_irom_application手册中有详细描述。见下图1：

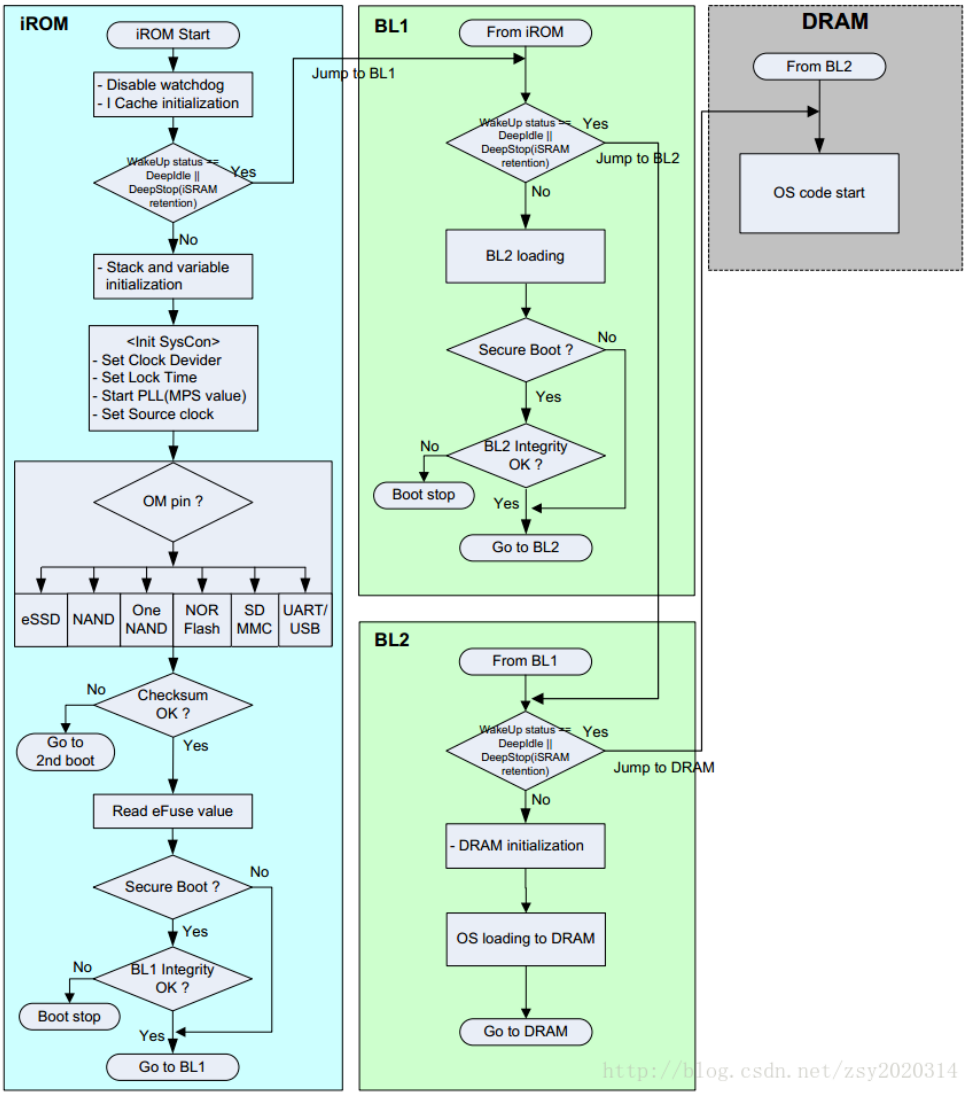

图1

1.2 首先把启动部分的代码分为3部分，以start.S为主，另外还有lowlevel_init.S，mem_setup.S，ctr0.S。

其中lowlevel_init.S主要是一部分硬件的初始化，尤其是系统时钟和DRAM的初始化。如果u-boot一旦被搬运到内存中运行，那么是必须要跳过时钟和DRAM的初始化的，因为这在搬运之前已经做过了。并且如果代码在内存中运行的时侯你却去初始化DRAM，那必然导致崩溃！

mem_setup.S：DRAM初始化代码和MMU相关代码放在这个文件中。

ctr0.S：u-boot自带的代码文件，存放汇编函数main。

1.3 启动代码相关的几个文件在u-boot中的路径

start.S： /arch/arm/cpu/armv7/start.S (需要自己修改)

lowlevel_init.S：/board/samsung/zsy210/ lowlevel_init.S (需要自己修改)

mem_setup.S：/board/samsung/zsy210/ mem_setup.S (u-boot没有，需要自己添加)

ctr0.S： /arch/arm/lib/ctr0.S (u-boot自带,一般不需要修改)

2. 启动过程原理

必须要明白的一点是，当代码从存储介质(nand flash，SD，norflash，onenand等)中搬运到了DRAM中后随即会跳转到内存中运行u-boot，接着会有一个重定位(relocate_code)的过程，relocate_code子函数在start.S中，而给relocate_code子函数传参数的是crt0.S中的main子函数。当判断到当前u-boot在内存的低地址处，那么relocate_code就会工作，把u-boot代码从低地址处再搬运到内存地址的顶端，然后跳转到新的位置去继续运行u-boot。而搬运的目标地址是在board_init_f()函数(此函数在/arch/arm/lib/board.c中)中计算出来的，见图2。

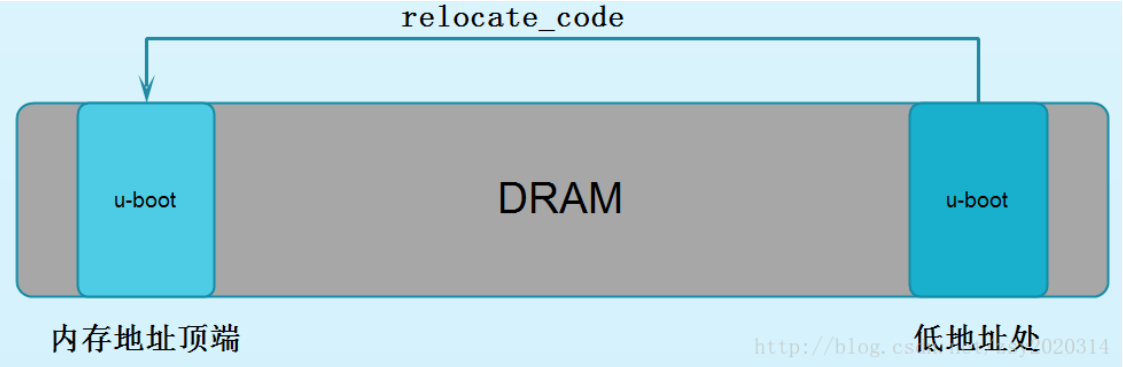

图2

下面，以start.S为主线，画出了其程序流程图，图中同样也表现出启动的整个流程和启动代码文件间的组织关系。所以后面直接贴出start.S的完整代码，大家结合流程图相信都可以看明白，至于逐句汇编的分析不是本文的重点。见图3.

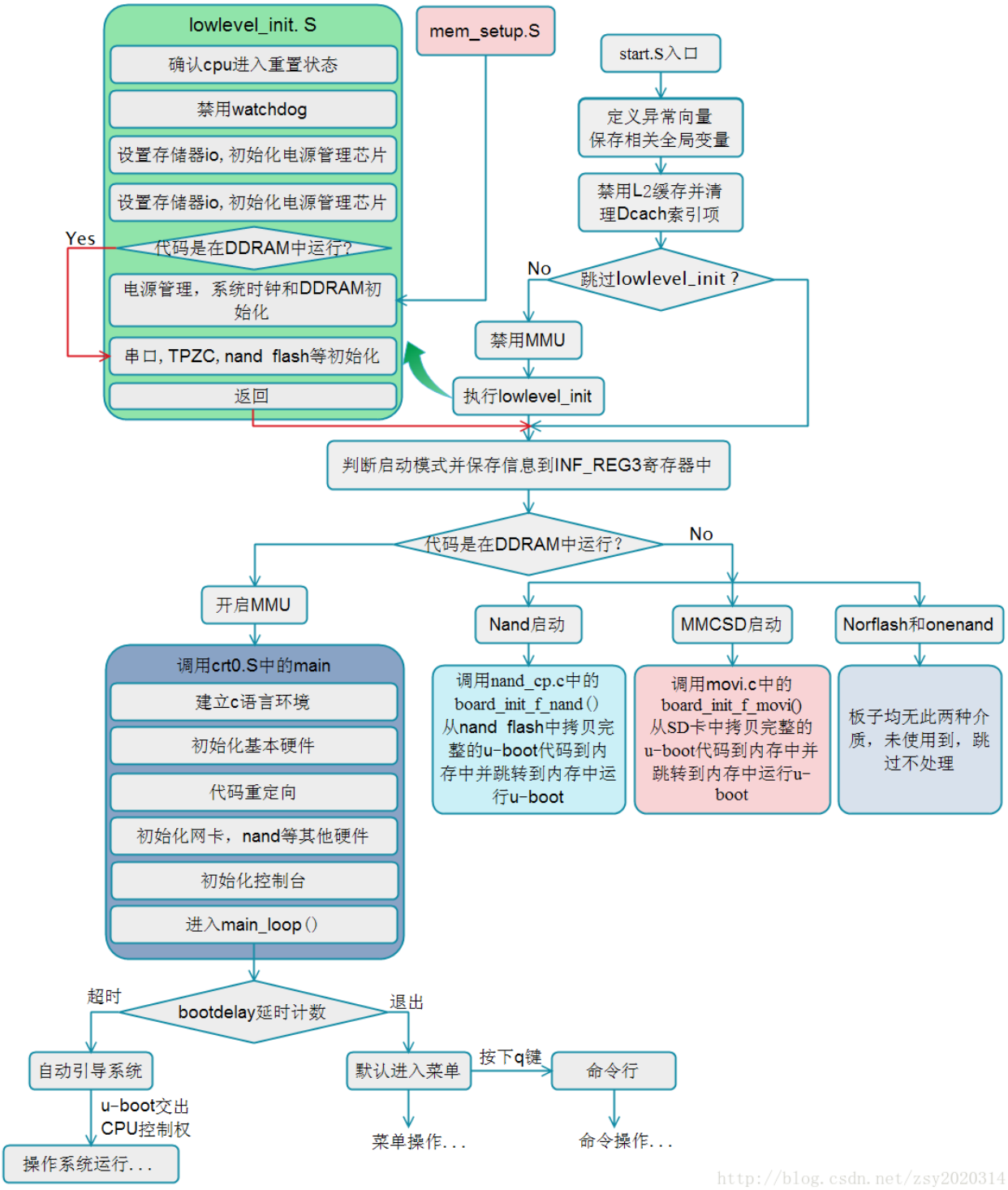

图3

3. start.S lowlevel_init.S mem_setup.S crt0.S的完整代码。

start.S的完整代码:

1.
2.
3.
4.
5.
6.

```
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.
31.
32. #include <asm-offsets.h>
33. #include <config.h>
34. #include <version.h>
35. #include <asm/system.h>
36. #include <linux/linkage.h>
37. #include <s5pv210.h>
38.
39. #ifndef CONFIG_ENABLE_MMU
40. #ifndef CONFIG_PHY_UBOOT_BASE
41. #define CONFIG_PHY_UBOOT_BASE   CONFIG_UBOOT_BASE
42. #endif /* CONFIG_PHY_UBOOT_BASE */
43. #endif /* CONFIG_ENABLE_MMU */
44.
45.
46. .globl _start
47. _start: b   reset
48.     ldr pc, _undefined_instruction
49.     ldr pc, _software_interrupt
50.     ldr pc, _prefetch_abort
51.     ldr pc, _data_abort
52.     ldr pc, _not_used
53.     ldr pc, _irq
54.     ldr pc, _fiq
55. #ifdef CONFIG_SPL_BUILD
56. _undefined_instruction: .word _undefined_instruction
57. _software_interrupt:    .word _software_interrupt
58. _prefetch_abort:    .word _prefetch_abort
59. _data_abort:        .word _data_abort
60. _not_used:      .word _not_used
61. _irq:           .word _irq
62. _fiq:           .word _fiq
63. _pad:           .word 0x12345678
64. #else
65. _undefined_instruction: .word undefined_instruction
66. _software_interrupt:    .word software_interrupt
67. _prefetch_abort:    .word prefetch_abort
68. _data_abort:        .word data_abort
69. _not_used:      .word not_used
70. _irq:           .word irq
71. _fiq:           .word fiq
72. _pad:           .word 0x12345678
73. #endif  /* CONFIG_SPL_BUILD */
74.
75. .global _end_vect
76. _end_vect:
77.
78.     .balignl 16,0xdeadbeef
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
```

```
89.
90. .globl _TEXT_BASE
91. _TEXT_BASE:
92.     .word   CONFIG_SYS_TEXT_BASE
93.
94.
95.
96.
97.
98.  .globl _TEXT_PHY_BASE
99.  _TEXT_PHY_BASE:
100.     .word   CONFIG_PHY_UBOOT_BASE
101.
102.
103.
104.
105. .globl _bss_start_ofs
106. _bss_start_ofs:
107.     .word __bss_start - _start
108.
109. .global _image_copy_end_ofs
110. _image_copy_end_ofs:
111.     .word   __image_copy_end - _start
112.
113. .globl _bss_end_ofs
114. _bss_end_ofs:
115.     .word __bss_end__ - _start
116.
117. .globl _end_ofs
118. _end_ofs:
119.     .word _end - _start
120.
121. #ifdef CONFIG_USE_IRQ
122.
123. .globl IRQ_STACK_START
124. IRQ_STACK_START:
125.     .word   0x0badc0de
126.
127.
128. .globl FIQ_STACK_START
129. FIQ_STACK_START:
130.     .word 0x0badc0de
131. #endif
132.
133.
134. .globl IRQ_STACK_START_IN
135. IRQ_STACK_START_IN:
136.     .word   0x0badc0de
137.
138.
139.
140.
141.
142. reset:
143.     bl  save_boot_params
144.
145.
146.
147.
148.     msr cpsr_c, #0xd3       @ I & F disable, Mode: 0x13 - SVC
149.
150.
151.
152.
153.
154.
155. #if !(defined(CONFIG_OMAP44XX) && defined(CONFIG_SPL_BUILD))
156.
157.     mrc p15, 0, r0, c1, c0, 0   @ Read CP15 SCTRL Register
158.     bic r0, #CR_V       @ V = 0
159.     mcr p15, 0, r0, c1, c0, 0   @ Write CP15 SCTRL Register
160.
161.
162.     ldr r0, =_start
163.     mcr p15, 0, r0, c12, c0, 0  @Set VBAR
164. #endif
165.
166. #ifndef CONFIG_EVT1
167.     bl  disable_l2cache
168.
169.     mov r0, #0x0    @
170.     mov r1, #0x0    @ i
```

```
171.    mov r3, #0x0
172.    mov r4, #0x0
173. lp1:
174.    mov r2, #0x0    @ j
175. lp2:
176.    mov r3, r1, LSL #29     @ r3 = r1(i) <<29
177.    mov r4, r2, LSL #6      @ r4 = r2(j) <<6
178.    orr r4, r4, #0x2        @ r3 = (i<<29)|(j<<6)|(1<<1)
179.    orr r3, r3, r4
180.    mov r0, r3          @ r0 = r3
181.    bl  CoInvalidateDCacheIndex
182.    add r2, #0x1        @ r2(j)++
183.    cmp r2, #1024       @ r2 < 1024
184.    bne lp2         @ jump to lp2
185.    add r1, #0x1        @ r1(i)++
186.    cmp r1, #8          @ r1(i) < 8
187.    bne lp1         @ jump to lp1
188.
189.    bl  set_l2cache_auxctrl
190.
191.    bl  enable_l2cache
192. #endif
193.
194.    bl  disable_l2cache
195.
196.    bl  set_l2cache_auxctrl_cycle
197.
198.    bl  enable_l2cache
199.
200.    bl  zsy210_iic_pm_open
201.
202.
203.
204. #ifndef CONFIG_SKIP_LOWLEVEL_INIT
205.    bl  cpu_init_cp15
206.    bl  cpu_init_crit
207. #endif
208.
209.
210.    ldr r0, =PRO_ID_BASE
211.    ldr r1, [r0,#OMR_OFFSET]
212.    bic r2, r1, #0xffffffc1
213.
214.
215.    cmp r2, #0x0        @ 512B 4-cycle
216.    moveq   r3, #BOOT_NAND
217.
218.    cmp r2, #0x2        @ 2KB 5-cycle
219.    moveq   r3, #BOOT_NAND
220.
221.    cmp r2, #0x4        @ 4KB 5-cycle   8-bit ECC
222.    moveq   r3, #BOOT_NAND
223.
224.    cmp r2, #0x6        @ 4KB 5-cycle   16-bit ECC
225.    moveq   r3, #BOOT_NAND
226.
227.    cmp r2, #0x8        @ OneNAND Mux
228.    moveq   r3, #BOOT_ONENAND
229.
230.
231.    cmp     r2, #0xc
232.    moveq   r3, #BOOT_MMCSD
233.
234.
235.    cmp     r2, #0x14
236.    moveq   r3, #BOOT_NOR
237.
238.
239.    cmp     r2, #(0x1<<4)
240.    moveq   r3, #BOOT_SEC_DEV
241.
242.    ldr r0, =INF_REG_BASE
243.    str r3, [r0, #INF_REG3_OFFSET]
244.
245.    ldr r0, =0xE010E81C
246.    ldr r1, =0x00005301
247.    str r1, [r0]
248.
249.
250.
251.
252.
```

```
253.    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
254.    bic sp, sp, #7
255.    ldr r0,=0x00000000
256.
257.    ldr r1, =0xff000fff
258.    bic r2, pc, r1
259.    ldr r3, _TEXT_BASE
260.    bic r3, r3, r1
261.    cmp r2, r3
262.    beq run_in_ram
263.
264. #if defined(CONFIG_EVT1)
265.
266.    ldr r0, =0xD0037488
267.    ldr r1, [r0]
268.    ldr r2, =0xEB200000
269.    cmp r1, r2
270.    beq mmcsd_boot
271. #endif
272.
273.    ldr r0, =INF_REG_BASE
274.    ldr r1, [r0, #INF_REG3_OFFSET]
275.    cmp r1, #BOOT_NAND
276.    beq nand_boot
277.    cmp r1, #BOOT_ONENAND
278.    beq onenand_boot
279.    cmp r1, #BOOT_MMCSD
280.    beq mmcsd_boot
281.    cmp r1, #BOOT_NOR
282.    beq nor_boot
283.    cmp r1, #BOOT_SEC_DEV
284.    beq mmcsd_boot
285.
286. nand_boot:
287.    bl  ledon_1
288.    bl  board_init_f_nand
289.
290. mmcsd_boot:
291.    bl  ledon_1
292.    bl  board_init_f_movi
293.
294. nor_boot:
295.    bl  ledon
296. onenand_boot:
297.    bl  ledon
298.
299.
300.
301.
302.
303.
304. run_in_ram:
305.
306. #if defined(CONFIG_ENABLE_MMU)
307. enable_mmu:
308.
309.    ldr r5, =0x0000ffff
310.    mcr p15, 0, r5, c3, c0, 0       @load domain access register
311.
312.
313.    ldr r0, _mmu_table_base
314.    ldr r1, =CONFIG_PHY_UBOOT_BASE
315.    ldr r2, =0xfff00000
316.    bic r0, r0, r2
317.    orr r1, r0, r1
318.    mcr p15, 0, r1, c2, c0, 0
319.
320.
321. mmu_on:
322.    mrc p15, 0, r0, c1, c0, 0
323.    orr r0, r0, #1
324.    mcr p15, 0, r0, c1, c0, 0
325.    nop
326.    nop
327.    nop
328.    nop
329. #endif
330.    bl ledon_2
331.    bl _main
332.
333.
334.
```

```
335. #ifndef CONFIG_SPL_BUILD
336.
337.
338.
339.
340.
341.
342.
343. ENTRY(relocate_code)
344.     mov r4, r0
345.     mov r5, r1
346.     mov r6, r2
347.
348.     adr r0, _start
349.     cmp r0, r6
350.     moveq   r9, #0
351.     beq relocate_done
352.     mov r1, r6
353.     ldr r3, _image_copy_end_ofs
354.     add r2, r0, r3
355.
356. copy_loop:
357.     ldmia   r0!, {r9-r10}
358.     stmia   r1!, {r9-r10}
359.     cmp r0, r2
360.     blo copy_loop
361.
362.
363.
364.
365.     ldr r0, _TEXT_BASE
366.     sub r9, r6, r0
367.     ldr r10, _dynsym_start_ofs
368.     add r10, r10, r0
369.     ldr r2, _rel_dyn_start_ofs
370.     add r2, r2, r0
371.     ldr r3, _rel_dyn_end_ofs
372.     add r3, r3, r0
373. fixloop:
374.     ldr r0, [r2]
375.     add r0, r0, r9
376.     ldr r1, [r2, #4]
377.     and r7, r1, #0xff
378.     cmp r7, #23
379.     beq fixrel
380.     cmp r7, #2
381.     beq fixabs
382.
383.     b   fixnext
384. fixabs:
385.
386.     mov r1, r1, LSR #4
387.     add r1, r10, r1
388.     ldr r1, [r1, #4]
389.     add r1, r1, r9
390.     b   fixnext
391. fixrel:
392.
393.     ldr r1, [r0]
394.     add r1, r1, r9
395. fixnext:
396.     str r1, [r0]
397.     add r2, r2, #8
398.     cmp r2, r3
399.     blo fixloop
400.
401. relocate_done:
402.
403.     bx  lr   /*返回到从定位的高端地址执行新的boot code？？ */
404.
405. _rel_dyn_start_ofs:
406.     .word __rel_dyn_start - _start
407. _rel_dyn_end_ofs:
408.     .word __rel_dyn_end - _start
409. _dynsym_start_ofs:
410.     .word __dynsym_start - _start
411. ENDPROC(relocate_code)
412.
413. #endif
414.
415. ENTRY(c_runtime_cpu_setup)
416.
```

```
417.
418.
419. #ifndef CONFIG_SYS_ICACHE_OFF
420.     mcr p15, 0, r0, c7, c5, 0   @ invalidate icache
421.     mcr     p15, 0, r0, c7, c10, 4  @ DSB
422.     mcr     p15, 0, r0, c7, c5, 4   @ ISB
423. #endif
424.
425.
426.
427. #if !defined(CONFIG_TEGRA20)
428.
429.     ldr     r0, =_start
430.     add     r0, r0, r9
431.     mcr     p15, 0, r0, c12, c0, 0  @Set VBAR
432. #endif /* !Tegra20 */
433.
434.     bx  lr
435.
436. ENDPROC(c_runtime_cpu_setup)
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447. ENTRY(save_boot_params)
448.     bx  lr          @ back to my caller
449. ENDPROC(save_boot_params)
450.     .weak   save_boot_params
451.
452.
453.
454.
455.
456.
457.
458.
459.
460. ENTRY(cpu_init_cp15)
461.
462.
463.
464.
465.     mov r0, #0                  @ set up for MCR
466.     mcr p15, 0, r0, c8, c7, 0   @ invalidate TLBs
467.     mcr p15, 0, r0, c7, c5, 0   @ invalidate icache
468.
469.
470.
471.
472.     mrc p15, 0, r0, c1, c0, 0
473.     bic r0, r0, #0x00002000     @ clear bits 13 (–V-)
474.     bic r0, r0, #0x00000007     @ clear bits 2:0 (-CAM)
475.     orr r0, r0, #0x00000002     @ set bit 1 (–A-) Align
476.     orr r0, r0, #0x00000800     @ set bit 12 (Z—) BTB
477.     mcr     p15, 0, r0, c1, c0, 0
478.
479.     mov pc, lr          @ back to my caller
480. ENDPROC(cpu_init_cp15)
481.
482. #ifndef CONFIG_SKIP_LOWLEVEL_INIT
483.
484.
485.
486.
487.
488.
489.
490.
491. ENTRY(cpu_init_crit)
492.
493.
494.
495.
496.
497.
498.     b   lowlevel_init       @ go setup pll,mux,memory
```

```
499. ENDPROC(cpu_init_crit)
500. #endif
501.
502.
503.
504.
505.
506.
507. ENTRY(zsy210_iic_pm_open)
508.
509. #ifdef CONFIG_ZSY210_IIC_PM_CHIP
510.
511.     ldr     r0, =ELFIN_GPIO_BASE
512.     ldr     r1, =0x00100000
513.     str     r1, [r0, #GPJ2CON_OFFSET]
514.
515.     ldr     r1, =0x0400
516.     str     r1, [r0, #GPJ2PUD_OFFSET]
517.
518.     ldr     r1, =0x20
519.     str     r1, [r0, #GPJ2DAT_OFFSET]
520. #endif /* CONFIG_ZSY210_IIC_PM_CHIP */
521.     mov pc, lr          @ back to my caller
522. ENDPROC(zsy210_iic_pm_open)
523.
524.
525. #if defined(CONFIG_ENABLE_MMU)
526. _mmu_table_base:
527.     .word mmu_table
528. #endif
529.
530.
531.
532.
533.
534.
535.
536.
537.
538. #if defined(CONFIG_ENABLE_MMU)
539.     .globl theLastJump
540. theLastJump:
541.     mov r9, r0
542.     ldr r3, =0xfff00000
543.     ldr r4, _TEXT_PHY_BASE
544.     adr r5, phy_last_jump
545.     bic r5, r5, r3
546.     orr r5, r5, r4
547.     mov pc, r5
548. phy_last_jump:
549.
550.
551.
552.     mrc p15, 0, r0, c1, c0, 0
553.     bic r0, r0, #0x00002300
554.     bic r0, r0, #0x00000087
555.     orr r0, r0, #0x00000002
556.     orr r0, r0, #0x00001000
557.     mcr p15, 0, r0, c1, c0, 0
558.
559.     mcr p15, 0, r0, c8, c7, 0
560.
561.     mov r0, #0
562.     mov pc, r9
563. #endif
564.
565.
566.
567. #ifndef CONFIG_SPL_BUILD
568.
569.
570.
571.
572.
573.
574.
575. @
576. @ IRQ stack frame.
577. @
578. #define S_FRAME_SIZE    72
579.
580. #define S_OLD_R0    68
```

```
581. #define S_PSR       64
582. #define S_PC        60
583. #define S_LR        56
584. #define S_SP        52
585.
586. #define S_IP        48
587. #define S_FP        44
588. #define S_R10       40
589. #define S_R9        36
590. #define S_R8        32
591. #define S_R7        28
592. #define S_R6        24
593. #define S_R5        20
594. #define S_R4        16
595. #define S_R3        12
596. #define S_R2        8
597. #define S_R1        4
598. #define S_R0        0
599.
600. #define MODE_SVC 0x13
601. #define I_BIT    0x80
602.
603.
604.
605.
606.
607.
608.    .macro  bad_save_user_regs
609.    sub sp, sp, #S_FRAME_SIZE       @ carve out a frame on current
610.                    @ user stack
611.    stmia   sp, {r0 - r12}          @ Save user registers (now in
612.                    @ svc mode) r0-r12
613.    ldr r2, IRQ_STACK_START_IN      @ set base 2 words into abort
614.                    @ stack
615.    ldmia   r2, {r2 - r3}           @ get values for “aborted” pc
616.                    @ and cpsr (into parm regs)
617.    add r0, sp, #S_FRAME_SIZE       @ grab pointer to old stack
618.
619.    add r5, sp, #S_SP
620.    mov r1, lr
621.    stmia   r5, {r0 - r3}           @ save sp_SVC, lr_SVC, pc, cpsr
622.    mov r0, sp              @ save current stack into r0
623.                    @ (param register)
624.    .endm
625.
626.    .macro  irq_save_user_regs
627.    sub sp, sp, #S_FRAME_SIZE
628.    stmia   sp, {r0 - r12}          @ Calling r0-r12
629.    add r8, sp, #S_PC           @ !! R8 NEEDS to be saved !!
630.                    @ a reserved stack spot would
631.                    @ be good.
632.    stmdb   r8, {sp, lr}^           @ Calling SP, LR
633.    str lr, [r8, #0]            @ Save calling PC
634.    mrs r6, spsr
635.    str r6, [r8, #4]            @ Save CPSR
636.    str r0, [r8, #8]            @ Save OLD_R0
637.    mov r0, sp
638.    .endm
639.
640.    .macro  irq_restore_user_regs
641.    ldmia   sp, {r0 - lr}^          @ Calling r0 - lr
642.    mov r0, r0
643.    ldr lr, [sp, #S_PC]         @ Get PC
644.    add sp, sp, #S_FRAME_SIZE
645.    subs    pc, lr, #4          @ return & move spsr_svc into
646.                    @ cpsr
647.    .endm
648.
649.    .macro get_bad_stack
650.    ldr r13, IRQ_STACK_START_IN     @ setup our mode stack (enter
651.                    @ in banked mode)
652.
653.    str lr, [r13]           @ save caller lr in position 0
654.                    @ of saved stack
655.    mrs lr, spsr            @ get the spsr
656.    str lr, [r13, #4]           @ save spsr in position 1 of
657.                    @ saved stack
658.
659.    mov r13, #MODE_SVC          @ prepare SVC-Mode
660.    @ msr   spsr_c, r13
661.    msr spsr, r13           @ switch modes, make sure
662.                    @ moves will execute
```

```
663.    mov lr, pc              @ capture return pc
664.    movs    pc, lr          @ jump to next instruction &
665.                    @ switch modes.
666.    .endm
667.
668.    .macro get_bad_stack_swi
669.    sub r13, r13, #4        @ space on current stack for
670.                    @ scratch reg.
671.    str r0, [r13]           @ save R0's value.
672.    ldr r0, IRQ_STACK_START_IN      @ get data regions start
673.                    @ spots for abort stack
674.    str lr, [r0]            @ save caller lr in position 0
675.                    @ of saved stack
676.    mrs r0, spsr            @ get the spsr
677.    str lr, [r0, #4]        @ save spsr in position 1 of
678.                    @ saved stack
679.    ldr r0, [r13]           @ restore r0
680.    add r13, r13, #4        @ pop stack entry
681.    .endm
682.
683.    .macro get_irq_stack        @ setup IRQ stack
684.    ldr sp, IRQ_STACK_START
685.    .endm
686.
687.    .macro get_fiq_stack        @ setup FIQ stack
688.    ldr sp, FIQ_STACK_START
689.    .endm
690.
691.
692.
693.
694.    .align  5
695. undefined_instruction:
696.    get_bad_stack
697.    bad_save_user_regs
698.    bl  do_undefined_instruction
699.
700.    .align  5
701. software_interrupt:
702.    get_bad_stack_swi
703.    bad_save_user_regs
704.    bl  do_software_interrupt
705.
706.    .align  5
707. prefetch_abort:
708.    get_bad_stack
709.    bad_save_user_regs
710.    bl  do_prefetch_abort
711.
712.    .align  5
713. data_abort:
714.    get_bad_stack
715.    bad_save_user_regs
716.    bl  do_data_abort
717.
718.    .align  5
719. not_used:
720.    get_bad_stack
721.    bad_save_user_regs
722.    bl  do_not_used
723.
724. #ifdef CONFIG_USE_IRQ
725.
726.    .align  5
727. irq:
728.    get_irq_stack
729.    irq_save_user_regs
730.    bl  do_irq
731.    irq_restore_user_regs
732.
733.    .align  5
734. fiq:
735.    get_fiq_stack
736.
737.    irq_save_user_regs
738.    bl  do_fiq
739.    irq_restore_user_regs
740.
741. #else
742.
743.    .align  5
744. irq:
```

```
745.    get_bad_stack
746.    bad_save_user_regs
747.    bl  do_irq
748.
749.    .align  5
750. fiq:
751.    get_bad_stack
752.    bad_save_user_regs
753.    bl  do_fiq
754.
755. #endif /* CONFIG_USE_IRQ */
756. #endif /* CONFIG_SPL_BUILD */
```

lowlevel_init.S的完整代码:

```
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25. #include <config.h>
26. #include <version.h>
27. #include <asm/arch/cpu.h>
28. #include <asm/arch/power.h>
29. #include <s5pv210.h>
30. #include “zsy210_val.h”
31.
32.
33.
34.
35.
36.
37. _TEXT_BASE:
38.    .word   CONFIG_SYS_TEXT_BASE
39.
40.    .globl lowlevel_init
41. lowlevel_init:
42.    mov r9, lr
43.
44.
45.    ldr r0, =(ELFIN_CLOCK_POWER_BASE+RST_STAT_OFFSET)
46.    ldr r1, [r0]
47.    bic r1, r1, #0xfff6ffff
48.    cmp r1, #0x10000
49.    beq wakeup_reset_pre
50.    cmp r1, #0x80000
51.    beq wakeup_reset_from_didle
52.
53.
54.    ldr r0, =(ELFIN_CLOCK_POWER_BASE + OTHERS_OFFSET)
55.    ldr r1, [r0]
56.    ldr r2, =IO_RET_REL
57.    orr r1, r1, r2
58.    str r1, [r0]
59.
60.
61.    ldr r0, =ELFIN_WATCHDOG_BASE
62.    mov r1, #0
63.    str r1, [r0]
64.
65.
66.    ldr r0, =ELFIN_GPIO_BASE
67.
```

```
    ldr r1, [r0, #GPJ1CON_OFFSET]
    bic r1, r1, #0xFFFFFF
    ldr r2, =0x444444
    orr r1, r1, r2
    str r1, [r0, #GPJ1CON_OFFSET]

    ldr r1, [r0, #GPJ1PUD_OFFSET]
    ldr r2, =0x3ff
    bic r1, r1, r2
    str r1, [r0, #GPJ1PUD_OFFSET]


    ldr r1, [r0, #GPJ4CON_OFFSET]
    bic r1, r1, #(0xf<<16)
    ldr r2, =(0x4<<16)
    orr r1, r1, r2
    str r1, [r0, #GPJ4CON_OFFSET]

    ldr r1, [r0, #GPJ4PUD_OFFSET]
    ldr r2, =(0x3<<8)
    bic r1, r1, r2
    str r1, [r0, #GPJ4PUD_OFFSET]


    ldr r0, =ELFIN_SROM_BASE
    mov r1, #0x1
    str r1, [r0]


    ldr r0, =(ELFIN_CLOCK_POWER_BASE + PS_HOLD_CONTROL_OFFSET)
    ldr r1, [r0]
    orr r1, r1, #0x300
    orr r1, r1, #0x1
    str r1, [r0]




    ldr r0, =0xff000fff
    bic r1, pc, r0
    ldr r2, _TEXT_BASE
    bic r2, r2, r0
    cmp     r1, r2
    beq     1f


#ifdef CONFIG_ZSY210_IIC_PM_CHIP
    bl PMIC_InitIp
#endif


    bl system_clock_init

    bl mem_ctrl_asm_init

1:

    bl uart_asm_init


    bl tzpc_asm_init

#if defined(CONFIG_NAND)

    bl nand_asm_init
#endif


    ldr r0, =(ELFIN_CLOCK_POWER_BASE+RST_STAT_OFFSET)
    ldr r1, [r0]
    bic r1, r1, #0xfffeffff
    cmp r1, #0x10000
    beq wakeup_reset_pre


    ldr r0, =0xE010C300
    orr r1, r1, #(0x1<<23)
    str r1, [r0]


    ldr r0, =ELFIN_UART_CONSOLE_BASE
```

```
150.    ldr r1, =0x4b4b4b4b
151.    str r1, [r0, #UTXH_OFFSET]
152.
153.    mov lr, r9
154.    mov pc, lr
155.
156.
157.
158.
159.
160. uart_asm_init:
161.
162.    @ GPIO setting for UART
163.    ldr r0, =ELFIN_GPIO_BASE
164.    ldr r1, =0x22222222
165.    str     r1, [r0, #GPA0CON_OFFSET]
166.
167.    ldr     r1, =0x2222
168.    str     r1, [r0, #GPA1CON_OFFSET]
169.
170.    ldr r0, =ELFIN_UART_CONSOLE_BASE        @0xEC000000
171.    mov r1, #0x0
172.    str r1, [r0, #UFCON_OFFSET]
173.    str r1, [r0, #UMCON_OFFSET]
174.
175.    mov r1, #0x3
176.    str r1, [r0, #ULCON_OFFSET]
177.
178.    ldr r1, =0x3c5
179.    str r1, [r0, #UCON_OFFSET]
180.
181.    ldr r1, =UART_UBRDIV_VAL
182.    str r1, [r0, #UBRDIV_OFFSET]
183.
184.    ldr r1, =UART_UDIVSLOT_VAL
185.    str r1, [r0, #UDIVSLOT_OFFSET]
186.
187.    ldr r1, =0x4f4f4f4f
188.    str r1, [r0, #UTXH_OFFSET]      @'O'
189.
190.    mov pc, lr
191.
192.
193.
194.
195. tzpc_asm_init:
196.
197.    ldr r0, =ELFIN_TZPC0_BASE
198.    mov r1, #0x0
199.    str r1, [r0]
200.    mov r1, #0xff
201.    str r1, [r0, #TZPC_DECPROT0SET_OFFSET]
202.    str r1, [r0, #TZPC_DECPROT1SET_OFFSET]
203.    str r1, [r0, #TZPC_DECPROT2SET_OFFSET]
204.
205.    ldr     r0, =ELFIN_TZPC1_BASE
206.    str r1, [r0, #TZPC_DECPROT0SET_OFFSET]
207.    str r1, [r0, #TZPC_DECPROT1SET_OFFSET]
208.    str r1, [r0, #TZPC_DECPROT2SET_OFFSET]
209.
210.    ldr r0, =ELFIN_TZPC2_BASE
211.    str r1, [r0, #TZPC_DECPROT0SET_OFFSET]
212.    str r1, [r0, #TZPC_DECPROT1SET_OFFSET]
213.    str r1, [r0, #TZPC_DECPROT2SET_OFFSET]
214.    str r1, [r0, #TZPC_DECPROT3SET_OFFSET]
215.
216.    ldr r0, =ELFIN_TZPC3_BASE
217.    str r1, [r0, #TZPC_DECPROT0SET_OFFSET]
218.    str r1, [r0, #TZPC_DECPROT1SET_OFFSET]
219.    str r1, [r0, #TZPC_DECPROT2SET_OFFSET]
220.
221.    mov pc, lr
222.
223.
224.
225.
226.
227.
228. nand_asm_init:
229.
230.
231.
```

```
232.
233.    ldr     r0, =ELFIN_GPIO_BASE
234.
235.    ldr     r1, [r0, #MP01CON_OFFSET]
236.    bic     r1, r1, #(0xf<<8)
237.    orr     r1, r1, #(0x3<<8)
238.    str     r1, [r0, #MP01CON_OFFSET]
239.
240.    ldr     r1, [r0, #MP01PUD_OFFSET]
241.    bic     r1, r1, #(0x3<<4)
242.    str     r1, [r0, #MP01PUD_OFFSET]
243.
244.    ldr     r1, [r0, #MP03CON_OFFSET]
245.    bic     r1, r1, #0xFFFFFF
246.    ldr     r2, =0x22222222
247.    orr     r1, r1, r2
248.    str     r1, [r0, #MP03CON_OFFSET]
249.
250.    ldr     r1, [r0, #MP03PUD_OFFSET]
251.    ldr     r2, =0x3fff
252.    bic     r1, r1, r2
253.    str     r1, [r0, #MP03PUD_OFFSET]
254.
255.    ldr     r0, =ELFIN_NAND_BASE
256.
257.    ldr     r1, [r0, #NFCONF_OFFSET]
258.    ldr     r2, =0x777F
259.    bic     r1, r1, r2
260.    ldr     r2, =NFCONF_VAL
261.    orr     r1, r1, r2
262.    str     r1, [r0, #NFCONF_OFFSET]
263.
264.    ldr     r1, [r0, #NFCONT_OFFSET]
265.    ldr     r2, =0x707C7
266.    bic     r1, r1, r2
267.    ldr     r2, =NFCONT_VAL
268.    orr     r1, r1, r2
269.    str     r1, [r0, #NFCONT_OFFSET]
270.
271.    ldr     r1, [r0, #NFCONF_OFFSET]
272.    orr     r1, r1, #0x70
273.    orr     r1, r1, #0x7700
274.    str     r1, [r0, #NFCONF_OFFSET]
275.
276.    ldr     r1, [r0, #NFCONT_OFFSET]
277.    orr     r1, r1, #0x03
278.    str     r1, [r0, #NFCONT_OFFSET]
279.
280.    mov     pc, lr
281.
282.
283.
284.    wakeup_reset_from_didle:
285.
286.    ldr r0, =ELFIN_CLOCK_POWER_BASE
287. lockloop:
288.    ldr r1, [r0, #APLL_CON0_OFFSET]
289.    and r1, r1, #(1<<29)
290.    cmp r1, #(1<<29)
291.    bne     lockloop
292.    beq exit_wakeup
293.
294. wakeup_reset_pre:
295.    mrc p15, 0, r1, c1, c0, 1   @Read CP15 Auxiliary control register
296.    and r1, r1, #0x80000000 @Check L2RD is disable or not
297.    cmp r1, #0x80000000
298.    bne wakeup_reset        @if L2RD is not disable jump to wakeup_reset
299.
300.    bl  disable_l2cache
301.    bl  v7_flush_dcache_all
302.
303.
304.
305. #ifdef CONFIG_ZSY210
306.    bl  enable_l2cache
307. #endif
308.
309. wakeup_reset:
310.
311.    bl system_clock_init
312.    bl mem_ctrl_asm_init
313.    bl tzpc_asm_init
```

```
#if defined(CONFIG_NAND)
    bl nand_asm_init
#endif

exit_wakeup:

    ldr r0, =(INF_REG_BASE+INF_REG0_OFFSET)
    ldr r1, [r0]

    mov pc, r1
    nop
    nop




system_clock_init:

    ldr r0, =ELFIN_CLOCK_POWER_BASE @0xe0100000


    ldr r1, =0x0
    str r1, [r0, #CLK_SRC0_OFFSET]

    ldr r1, =APLL_LOCKTIME_VAL
    str r1, [r0, #APLL_LOCK_OFFSET]


#if defined(CONFIG_CHECK_MPLL_LOCK)
retryloop:
#endif
    ldr r1, =0x0
    str r1, [r0, #APLL_CON0_OFFSET]
    ldr r1, =0x0
    str r1, [r0, #MPLL_CON_OFFSET]

    ldr r1, =0x0
    str r1, [r0, #MPLL_CON_OFFSET]

    ldr     r1, [r0, #CLK_DIV0_OFFSET]
    ldr r2, =CLK_DIV0_MASK
    bic r1, r1, r2

    ldr r2, =CLK_DIV0_VAL
    orr r1, r1, r2
    str r1, [r0, #CLK_DIV0_OFFSET]

    ldr r1, =APLL_VAL
    str r1, [r0, #APLL_CON0_OFFSET]

    ldr r1, =MPLL_VAL
    str r1, [r0, #MPLL_CON_OFFSET]

    ldr r1, =VPLL_VAL
    str r1, [r0, #VPLL_CON_OFFSET]
#if defined(CONFIG_EVT1)
    ldr r1, =AFC_ON
    str r1, [r0, #APLL_CON1_OFFSET]
#endif
    mov r1, #0x10000
1:  subs    r1, r1, #1
    bne 1b

#if defined(CONFIG_CHECK_MPLL_LOCK)

    ldr r1, [r0, #MPLL_CON_OFFSET]
    orr     r1, r1, #(1<<28)
    str r1, [r0, #MPLL_CON_OFFSET]

    mov r1, #0x100
1:  subs    r1, r1, #1
    bne 1b

    ldr r1, [r0, #MPLL_CON_OFFSET]
    and r1, r1, #(1<<29)
    cmp r1, #(1<<29)
    bne     retryloop


    ldr r1, [r0, #MPLL_CON_OFFSET]
    bic     r1, r1, #(1<<28)
```

```
396.     str r1, [r0, #MPLL_CON_OFFSET]
397. #endif
398.
399.     ldr r1, [r0, #CLK_SRC0_OFFSET]
400.     ldr r2, =0x10001111
401.     orr r1, r1, r2
402.     str r1, [r0, #CLK_SRC0_OFFSET]
403.
404. #if defined(CONFIG_MCP_AC)
405.
406.
407.     ldr r1, [r0, #CLK_SRC6_OFFSET]
408.     bic r1, r1, #(0x3<<24)
409.     orr r1, r1, #0x01000000
410.     str r1, [r0, #CLK_SRC6_OFFSET]
411.
412.
413.     ldr r1, [r0, #CLK_DIV6_OFFSET]
414.     bic r1, r1, #(0xF<<28)
415.     bic r1, r1, #(0x7<<12)    @; ONENAND_RATIO: 0
416.     orr r1, r1, #0x30000000
417.     str r1, [r0, #CLK_DIV6_OFFSET]
418.
419. #elif defined (CONFIG_MCP_N)
420.
421.     ldr r1, [r0, #CLK_SRC6_OFFSET]
422.     mov r1, #0x00000000
423.     str r1, [r0, #CLK_SRC6_OFFSET]
424.
425.
426.     ldr r1, [r0, #CLK_DIV6_OFFSET]
427.     mov r1, #0x00000000
428.     str r1, [r0, #CLK_DIV6_OFFSET]
429.
430.
431. #elif defined (CONFIG_MCP_H)
432.
433.
434.     ldr r1, [r0, #CLK_SRC6_OFFSET]
435.     bic r1, r1, #(0x3<<24)
436.     orr r1, r1, #0x00000000
437.     str r1, [r0, #CLK_SRC6_OFFSET]
438.
439.
440.     ldr r1, [r0, #CLK_DIV6_OFFSET]
441.     bic r1, r1, #(0xF<<28)
442.     bic r1, r1, #(0x7<<12)    @; ONENAND_RATIO: 0
443.     orr r1, r1, #0x00000000
444.     str r1, [r0, #CLK_DIV6_OFFSET]
445.
446. #elif defined (CONFIG_MCP_B) || defined (CONFIG_MCP_D)
447.
448.
449.     ldr r1, [r0, #CLK_SRC6_OFFSET]
450.     bic r1, r1, #(0x3<<24)
451.     orr r1, r1, #0x01000000
452.     str r1, [r0, #CLK_SRC6_OFFSET]
453.
454.
455.     ldr r1, [r0, #CLK_DIV6_OFFSET]
456.     bic r1, r1, #(0xF<<28)
457.     bic r1, r1, #(0x7<<12)    @; ONENAND_RATIO: 0
458.     orr r1, r1, #0x30000000
459.     str r1, [r0, #CLK_DIV6_OFFSET]
460.
461. #elif defined (CONFIG_MCP_SINGLE)
462.
463.
464.     ldr r1, [r0, #CLK_DIV6_OFFSET]
465.     bic r1, r1, #(0x7<<12)    @; ONENAND_RATIO: 0
466.     str r1, [r0, #CLK_DIV6_OFFSET]
467.
468. #endif
469.     mov pc, lr
470.
471.
472.
473.
474. #ifdef CONFIG_ENABLE_MMU
475.
476. #ifdef CONFIG_MCP_SINGLE
477.
```

```
478.
479.
480.
481.
482.
483.
484.
485.
486.
487. .macro FL_SECTION_ENTRY base,ap,d,c,b
488.     .word (\base << 20) | (\ap << 10) | \
489.           (\d << 5) | (1<<4) | (\c << 3) | (\b << 2) | (1<<1)
490. .endm
491. .section .mmudata, ”a”
492.     .align 14
493.
494.     .globl mmu_table
495. mmu_table:
496.     .set __base,0
497.
498.     .rept 0x100
499.     FL_SECTION_ENTRY __base,3,0,0,0
500.     .set __base,__base+1
501.     .endr
502.
503.
504.     .rept 0x200 - 0x100
505.     .word 0x00000000
506.     .endr
507. #ifdef CONFIG_ZSY10_1G
508.     .set __base,0x200
509.
510.     .rept 0x600 - 0x200
511.     FL_SECTION_ENTRY __base,3,0,1,1
512.     .set __base,__base+1
513.     .endr
514. #else
515.     .set __base,0x200
516.
517.     .rept 0x600 - 0x200
518.     FL_SECTION_ENTRY __base,3,0,1,1
519.     .set __base,__base+1
520.     .endr
521.
522.
523.
524.
525.
526.
527.
528. #endif /* CONFIG_ZSY10_1G */
529.     .rept 0x800 - 0x600
530.     .word 0x00000000
531.     .endr
532.
533.
534.     .set __base,0x800
535.
536.     .rept 0xb00 - 0x800
537.     FL_SECTION_ENTRY __base,3,0,0,0
538.     .set __base,__base+1
539.     .endr
540.
541.
542.
543.
544.
545.     .set __base,0xB00
546.     .rept 0xc00 - 0xb00
547.     FL_SECTION_ENTRY __base,3,0,0,0
548.     .set __base,__base+1
549.     .endr
550. #ifdef CONFIG_ZSY10_1G
551.     .set __base,0x200
552.
553.     .rept 0xD00 - 0xC00
554.     FL_SECTION_ENTRY __base,3,0,1,1
555.     .set __base,__base+1
556.     .endr
557.
558.
559.     @.rept 0xD00 - 0xC80
```

```
560.    @.word 0x00000000
561.    @.endr
562.
563.    .set __base,0xD00
564.
565.    .rept 0x1000 - 0xD00
566.    FL_SECTION_ENTRY __base,3,0,0,0
567.    .set __base,__base+1
568.    .endr
569. #else
570.    .set __base,0x200
571.
572.    .rept 0xD00 - 0xC00
573.    FL_SECTION_ENTRY __base,3,0,1,1
574.    .set __base,__base+1
575.    .endr
576.
577.
578.    @.rept 0xD00 - 0xC80
579.    @.word 0x00000000
580.    @.endr
581.
582.    .set __base,0xD00
583.
584.    .rept 0x1000 - 0xD00
585.    FL_SECTION_ENTRY __base,3,0,0,0
586.    .set __base,__base+1
587.    .endr
588. #endif /* CONFIG_ZSY10_1G */
589.    #else   // CONFIG_MCP_AC, CONFIG_MCP_H, CONFIG_MCP_B
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601. .macro FL_SECTION_ENTRY base,ap,d,c,b
602.    .word (\base << 20) | (\ap << 10) | \
603.          (\d << 5) | (1<<4) | (\c << 3) | (\b << 2) | (1<<1)
604. .endm
605. .section .mmudata, ”a”
606.    .align 14
607.
608.    .globl mmu_table
609. mmu_table:
610.    .set __base,0
611.
612.    .rept 0x100
613.    FL_SECTION_ENTRY __base,3,0,0,0
614.    .set __base,__base+1
615.    .endr
616.
617.
618.    .rept 0x300 - 0x100
619.    .word 0x00000000
620.    .endr
621.
622. #if defined(CONFIG_MCP_N)
623.    .set __base,0x300
624.
625.    .rept 0x400 - 0x300
626.    FL_SECTION_ENTRY __base,3,0,1,1
627.    .set __base,__base+1
628.    .endr
629. #else
630.    .set __base,0x300
631.
632.    .rept 0x350 - 0x300
633.    FL_SECTION_ENTRY __base,3,0,1,1
634.    .set __base,__base+1
635.    .endr
636.
637.
638.    .rept 0x400 - 0x350
639.    .word 0x00000000
640.    .endr
641. #endif
```

```
642.
643.    .set __base,0x400
644.
645.    .rept 0x500 - 0x400
646.    FL_SECTION_ENTRY __base,3,0,1,1
647.    .set __base,__base+1
648.    .endr
649.
650.    .rept 0x800 - 0x500
651.    .word 0x00000000
652.    .endr
653.
654.    .set __base,0x800
655.
656.    .rept 0xb00 - 0x800
657.    FL_SECTION_ENTRY __base,3,0,0,0
658.    .set __base,__base+1
659.    .endr
660.
661.    .set __base,0xB00
662.    .rept 0xc00 - 0xb00
663.    FL_SECTION_ENTRY __base,3,0,0,0
664.    .set __base,__base+1
665.    .endr
666.
667. #if defined(CONFIG_MCP_N)
668.    .set __base,0x300
669.
670.    .rept 0xD00 - 0xC00
671.    FL_SECTION_ENTRY __base,3,0,1,1
672.    .set __base,__base+1
673.    .endr
674. #else
675.    .set __base,0x300
676.
677.    .rept 0xC50 - 0xC00
678.    FL_SECTION_ENTRY __base,3,0,1,1
679.    .set __base,__base+1
680.    .endr
681.
682.
683.    .rept 0xD00 - 0xC50
684.    .word 0x00000000
685.    .endr
686. #endif
687.
688.    .set __base,0xD00
689.
690.    .rept 0x1000 - 0xD00
691.    FL_SECTION_ENTRY __base,3,0,0,0
692.    .set __base,__base+1
693.    .endr
694.    #endif
695. #endif
```

mem_setup.S的完整代码:

```
1. #include <config.h>
2. #include <s5pv210.h>
3.
4.    .globl mem_ctrl_asm_init
5. mem_ctrl_asm_init:
6.
7. #ifndef CONFIG_EVT1
8.
9.    ldr     r0, =ASYNC_MSYS_DMC0_BASE
10.
11.   ldr     r1, =0x0
12.   str     r1, [r0, #0x0]
13.
14.
15.   ldr     r1, =0x0
16.   str     r1, [r0, #0xC]
17.
18. #endif
19.
20. #ifdef CONFIG_MCP_SINGLE
21.
22.
23.
24.   ldr r0, =ELFIN_GPIO_BASE
25.
```

```
    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_0DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_1DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_2DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_3DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_4DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_5DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_6DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP1_7DRV_SR_OFFSET]

    ldr r1, =0x00002AAA
    str r1, [r0, #MP1_8DRV_SR_OFFSET]




    ldr r0, =ELFIN_GPIO_BASE

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_0DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_1DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_2DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_3DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_4DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_5DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_6DRV_SR_OFFSET]

    ldr r1, =0x0000AAAA
    str r1, [r0, #MP2_7DRV_SR_OFFSET]

    ldr r1, =0x00002AAA
    str r1, [r0, #MP2_8DRV_SR_OFFSET]


    ldr r0, =APB_DMC_0_BASE

    ldr r1, =0x00101000         @PhyControl0 DLL parameter setting, manual 0x00101000
    str r1, [r0, #DMC_PHYCONTROL0]

    ldr r1, =0x00000086         @PhyControl1 DLL parameter setting, LPDDR/LPDDR2 Case
    str r1, [r0, #DMC_PHYCONTROL1]

    ldr r1, =0x00101002         @PhyControl0 DLL on
    str r1, [r0, #DMC_PHYCONTROL0]

    ldr r1, =0x00101003         @PhyControl0 DLL start
    str r1, [r0, #DMC_PHYCONTROL0]

find_lock_val:
    ldr r1, [r0, #DMC_PHYSTATUS]    @Load Phystatus register value
    and r2, r1, #0x7
    cmp r2, #0x7            @Loop until DLL is locked
    bne find_lock_val

    and r1, #0x3fc0
    mov r2, r1, LSL #18
```

```
108.    orr r2, r2, #0x100000
109.    orr r2 ,r2, #0x1000
110.
111.    orr r1, r2, #0x3              @Force Value locking
112.    str r1, [r0, #DMC_PHYCONTROL0]
113.
114. #if 0   /* Memory margin test 10.01.05 */
115.    orr r1, r2, #0x1              @DLL off
116.    str r1, [r0, #DMC_PHYCONTROL0]
117. #endif
118.
119.    ldr r1, =0x0FFF2010           @ConControl auto refresh off
120.    str r1, [r0, #DMC_CONCONTROL]
121.
122.    ldr r1, =0x00212400           @MemControl BL=4, 2 chip, DDR2 type, dynamic self refresh, force precharge, dynamic power down off
123.    str r1, [r0, #DMC_MEMCONTROL]
124.
125.    ldr r1, =DMC0_MEMCONFIG_0         @MemConfig0 256MB config, 8 banks,Mapping Method[12:15]0:linear, 1:linterleaved, 2:Mixed
126.    str r1, [r0, #DMC_MEMCONFIG0]
127.
128.    ldr r1, =DMC0_MEMCONFIG_1         @MemConfig1
129.    str r1, [r0, #DMC_MEMCONFIG1]
130.
131.    ldr r1, =0xFF000000           @PrechConfig
132.    str r1, [r0, #DMC_PRECHCONFIG]
133.
134.    ldr r1, =DMC0_TIMINGA_REF         @TimingAref 7.8us*133MHz=1038(0x40E), 100MHz=780(0x30C), 20MHz=156(0x9C), 10MHz=78(0x4E)
135.    str r1, [r0, #DMC_TIMINGAREF]
136.
137.    ldr r1, =DMC0_TIMING_ROW          @TimingRow  for @200MHz
138.    str r1, [r0, #DMC_TIMINGROW]
139.
140.    ldr r1, =DMC0_TIMING_DATA         @TimingData CL=3
141.    str r1, [r0, #DMC_TIMINGDATA]
142.
143.    ldr r1, =DMC0_TIMING_PWR          @TimingPower
144.    str r1, [r0, #DMC_TIMINGPOWER]
145.
146.    ldr r1, =0x07000000           @DirectCmd  chip0 Deselect
147.    str r1, [r0, #DMC_DIRECTCMD]
148.
149.    ldr r1, =0x01000000           @DirectCmd  chip0 PALL
150.    str r1, [r0, #DMC_DIRECTCMD]
151.
152.    ldr r1, =0x00020000           @DirectCmd  chip0 EMRS2
153.    str r1, [r0, #DMC_DIRECTCMD]
154.
155.    ldr r1, =0x00030000           @DirectCmd  chip0 EMRS3
156.    str r1, [r0, #DMC_DIRECTCMD]
157.
158.    ldr r1, =0x00010400           @DirectCmd  chip0 EMRS1 (MEM DLL on, DQS# disable)
159.    str r1, [r0, #DMC_DIRECTCMD]
160.
161.    ldr r1, =0x00000542           @DirectCmd  chip0 MRS (MEM DLL reset) CL=4, BL=4
162.    str r1, [r0, #DMC_DIRECTCMD]
163.
164.    ldr r1, =0x01000000           @DirectCmd  chip0 PALL
165.    str r1, [r0, #DMC_DIRECTCMD]
166.
167.    ldr r1, =0x05000000           @DirectCmd  chip0 REFA
168.    str r1, [r0, #DMC_DIRECTCMD]
169.
170.    ldr r1, =0x05000000           @DirectCmd  chip0 REFA
171.    str r1, [r0, #DMC_DIRECTCMD]
172.
173.    ldr r1, =0x00000442           @DirectCmd  chip0 MRS (MEM DLL unreset)
174.    str r1, [r0, #DMC_DIRECTCMD]
175.
176.    ldr r1, =0x00010780           @DirectCmd  chip0 EMRS1 (OCD default)
177.    str r1, [r0, #DMC_DIRECTCMD]
178.
179.    ldr r1, =0x00010400           @DirectCmd  chip0 EMRS1 (OCD exit)
180.    str r1, [r0, #DMC_DIRECTCMD]
181.
182.    ldr r1, =0x07100000           @DirectCmd  chip1 Deselect
183.    str r1, [r0, #DMC_DIRECTCMD]
184.
185.    ldr r1, =0x01100000           @DirectCmd  chip1 PALL
186.    str r1, [r0, #DMC_DIRECTCMD]
187.
188.    ldr r1, =0x00120000           @DirectCmd  chip1 EMRS2
189.    str r1, [r0, #DMC_DIRECTCMD]
```

```
190.
191.    ldr r1, =0x00130000           @DirectCmd  chip1 EMRS3
192.    str r1, [r0, #DMC_DIRECTCMD]
193.
194.    ldr r1, =0x00110400           @DirectCmd  chip1 EMRS1 (MEM DLL on, DQS# disable)
195.    str r1, [r0, #DMC_DIRECTCMD]
196.
197.    ldr r1, =0x00100542           @DirectCmd  chip1 MRS (MEM DLL reset) CL=4, BL=4
198.    str r1, [r0, #DMC_DIRECTCMD]
199.
200.    ldr r1, =0x01100000           @DirectCmd  chip1 PALL
201.    str r1, [r0, #DMC_DIRECTCMD]
202.
203.    ldr r1, =0x05100000           @DirectCmd  chip1 REFA
204.    str r1, [r0, #DMC_DIRECTCMD]
205.
206.    ldr r1, =0x05100000           @DirectCmd  chip1 REFA
207.    str r1, [r0, #DMC_DIRECTCMD]
208.
209.    ldr r1, =0x00100442           @DirectCmd  chip1 MRS (MEM DLL unreset)
210.    str r1, [r0, #DMC_DIRECTCMD]
211.
212.    ldr r1, =0x00110780           @DirectCmd  chip1 EMRS1 (OCD default)
213.    str r1, [r0, #DMC_DIRECTCMD]
214.
215.    ldr r1, =0x00110400           @DirectCmd  chip1 EMRS1 (OCD exit)
216.    str r1, [r0, #DMC_DIRECTCMD]
217.
218.    ldr r1, =0x0FF02030           @ConControl auto refresh on
219.    str r1, [r0, #DMC_CONCONTROL]
220.
221.    ldr r1, =0xFFFF00FF           @PwrdnConfig
222.    str r1, [r0, #DMC_PWRDNCONFIG]
223.
224.    ldr r1, =0x00202400           @MemControl BL=4, 2 chip, DDR2 type, dynamic self refresh, force precharge, dynamic power down off
225.    str r1, [r0, #DMC_MEMCONTROL]
226.
227.
228.    ldr r0, =APB_DMC_1_BASE
229.
230.    ldr r1, =0x00101000           @Phycontrol0 DLL parameter setting
231.    str r1, [r0, #DMC_PHYCONTROL0]
232.
233.    ldr r1, =0x00000086           @Phycontrol1 DLL parameter setting
234.    str r1, [r0, #DMC_PHYCONTROL1]
235.
236.    ldr r1, =0x00101002           @PhyControl0 DLL on
237.    str r1, [r0, #DMC_PHYCONTROL0]
238.
239.    ldr r1, =0x00101003           @PhyControl0 DLL start
240.    str r1, [r0, #DMC_PHYCONTROL0]
241. find_lock_val1:
242.    ldr r1, [r0, #DMC_PHYSTATUS]       @Load Phystatus register value
243.    and r2, r1, #0x7
244.    cmp r2, #0x7              @Loop until DLL is locked
245.    bne find_lock_val1
246.
247.    and r1, #0x3fc0
248.    mov r2, r1, LSL #18
249.    orr r2, r2, #0x100000
250.    orr r2, r2, #0x1000
251.
252.    orr r1, r2, #0x3              @Force Value locking
253.    str r1, [r0, #DMC_PHYCONTROL0]
254.
255. #if 0   /* Memory margin test 10.01.05 */
256.    orr r1, r2, #0x1              @DLL off
257.    str r1, [r0, #DMC_PHYCONTROL0]
258. #endif
259.
260.
261.    ldr r0, =APB_DMC_1_BASE
262.
263.    ldr r1, =0x0FFF2010           @auto refresh off
264.    str r1, [r0, #DMC_CONCONTROL]
265.
266.    ldr r1, =DMC1_MEMCONTROL          @MemControl BL=4, 2 chip, DDR2 type, dynamic self refresh, force precharge, dynamic power down off
267.    str r1, [r0, #DMC_MEMCONTROL]
268.
269.    ldr r1, =DMC1_MEMCONFIG_0         @MemConfig0 512MB config, 8 banks,Mapping Method[12:15]0:linear, 1:linterleaved, 2:Mixed
270.    str r1, [r0, #DMC_MEMCONFIG0]
271.
```

```
272.  ldr r1, =DMC1_MEMCONFIG_1          @MemConfig1
273.  str r1, [r0, #DMC_MEMCONFIG1]
274.
275.  ldr r1, =0xFF000000
276.  str r1, [r0, #DMC_PRECHCONFIG]
277.
278.  ldr r1, =DMC1_TIMINGA_REF          @TimingAref 7.8us*133MHz=1038(0x40E), 100MHz=780(0x30C), 20MHz=156(0x9C), 10MHz=78(0x4
279.  str r1, [r0, #DMC_TIMINGAREF]
280.
281.  ldr r1, =DMC1_TIMING_ROW           @TimingRow  for @200MHz
282.  str r1, [r0, #DMC_TIMINGROW]
283.
284.  ldr r1, =DMC1_TIMING_DATA          @TimingData CL=3
285.  str r1, [r0, #DMC_TIMINGDATA]
286.
287.  ldr r1, =DMC1_TIMING_PWR           @TimingPower
288.  str r1, [r0, #DMC_TIMINGPOWER]
289.
290.
291.  ldr r1, =0x07000000           @DirectCmd  chip0 Deselect
292.  str r1, [r0, #DMC_DIRECTCMD]
293.
294.  ldr r1, =0x01000000           @DirectCmd  chip0 PALL
295.  str r1, [r0, #DMC_DIRECTCMD]
296.
297.  ldr r1, =0x00020000           @DirectCmd  chip0 EMRS2
298.  str r1, [r0, #DMC_DIRECTCMD]
299.
300.  ldr r1, =0x00030000           @DirectCmd  chip0 EMRS3
301.  str r1, [r0, #DMC_DIRECTCMD]
302.
303.  ldr r1, =0x00010400           @DirectCmd  chip0 EMRS1 (MEM DLL on, DQS# disable)
304.  str r1, [r0, #DMC_DIRECTCMD]
305.
306.  ldr r1, =0x00000542           @DirectCmd  chip0 MRS (MEM DLL reset) CL=4, BL=4
307.  str r1, [r0, #DMC_DIRECTCMD]
308.
309.  ldr r1, =0x01000000           @DirectCmd  chip0 PALL
310.  str r1, [r0, #DMC_DIRECTCMD]
311.
312.  ldr r1, =0x05000000           @DirectCmd  chip0 REFA
313.  str r1, [r0, #DMC_DIRECTCMD]
314.
315.  ldr r1, =0x05000000           @DirectCmd  chip0 REFA
316.  str r1, [r0, #DMC_DIRECTCMD]
317.
318.  ldr r1, =0x00000442           @DirectCmd  chip0 MRS (MEM DLL unreset)
319.  str r1, [r0, #DMC_DIRECTCMD]
320.
321.  ldr r1, =0x00010780           @DirectCmd  chip0 EMRS1 (OCD default)
322.  str r1, [r0, #DMC_DIRECTCMD]
323.
324.  ldr r1, =0x00010400           @DirectCmd  chip0 EMRS1 (OCD exit)
325.  str r1, [r0, #DMC_DIRECTCMD]
326.
327.  ldr r1, =0x07100000           @DirectCmd  chip1 Deselect
328.  str r1, [r0, #DMC_DIRECTCMD]
329.
330.  ldr r1, =0x01100000           @DirectCmd  chip1 PALL
331.  str r1, [r0, #DMC_DIRECTCMD]
332.
333.  ldr r1, =0x00120000           @DirectCmd  chip1 EMRS2
334.  str r1, [r0, #DMC_DIRECTCMD]
335.
336.  ldr r1, =0x00130000           @DirectCmd  chip1 EMRS3
337.  str r1, [r0, #DMC_DIRECTCMD]
338.
339.  ldr r1, =0x00110440           @DirectCmd  chip1 EMRS1 (MEM DLL on, DQS# disable)
340.  str r1, [r0, #DMC_DIRECTCMD]
341.
342.  ldr r1, =0x00100542           @DirectCmd  chip1 MRS (MEM DLL reset) CL=4, BL=4
343.  str r1, [r0, #DMC_DIRECTCMD]
344.
345.  ldr r1, =0x01100000           @DirectCmd  chip1 PALL
346.  str r1, [r0, #DMC_DIRECTCMD]
347.
348.  ldr r1, =0x05100000           @DirectCmd  chip1 REFA
349.  str r1, [r0, #DMC_DIRECTCMD]
350.
351.  ldr r1, =0x05100000           @DirectCmd  chip1 REFA
352.  str r1, [r0, #DMC_DIRECTCMD]
353.
```

```
354.    ldr r1, =0x00100442           @DirectCmd  chip1 MRS (MEM DLL unreset)
355.    str r1, [r0, #DMC_DIRECTCMD]
356.
357.    ldr r1, =0x00110780           @DirectCmd  chip1 EMRS1 (OCD default)
358.    str r1, [r0, #DMC_DIRECTCMD]
359.
360.    ldr r1, =0x00110400           @DirectCmd  chip1 EMRS1 (OCD exit)
361.    str r1, [r0, #DMC_DIRECTCMD]
362.
363.    ldr r1, =0x0FF02030           @ConControl auto refresh on
364.    str r1, [r0, #DMC_CONCONTROL]
365.
366.    ldr r1, =0xFFFF00FF           @PwrdnConfig
367.    str r1, [r0, #DMC_PWRDNCONFIG]
368.
369.    ldr r1, =DMC1_MEMCONTROL          @MemControl BL=4, 2 chip, DDR2 type, dynamic self refresh, force precharge, dynamic power down off
370.    str r1, [r0, #DMC_MEMCONTROL]
371.
372. #else   /* CONFIG_MCP_SINGLE */
373.
374.
375.    ldr r0, =APB_DMC_0_BASE
376.
377.    ldr r1, =0x00101000           @Phycontrol0 DLL parameter setting
378.    str r1, [r0, #DMC_PHYCONTROL0]
379.
380.    ldr r1, =0x00000084           @Phycontrol1 DLL parameter setting
381.    str r1, [r0, #DMC_PHYCONTROL1]
382.
383.    ldr r1, =0x00101002           @Phycontrol2 DLL parameter setting
384.    str r1, [r0, #DMC_PHYCONTROL0]
385.
386.    ldr r1, =0x00101003           @Dll on
387.    str r1, [r0, #DMC_PHYCONTROL0]
388.
389. find_lock_val:
390.    ldr r1, [r0, #DMC_PHYSTATUS]       @Load Phystatus register value
391.    and     r2, r1, #0x7
392.    cmp r2, #0x7              @Loop until DLL is locked
393.    bne find_lock_val
394.
395.    and r1, #0x3fc0
396.    mov r2, r1, LSL #18
397.    orr r2, r2, #0x100000
398.    orr r2, r2, #0x1000
399.
400.    orr r1, r2, #0x3              @Force Value locking
401.    str r1, [r0, #DMC_PHYCONTROL0]
402. #if 1 /* DRAM margin test 10.01.06 */
403.    orr r1, r2, #0x1              @DLL off
404.    str r1, [r0, #DMC_PHYCONTROL0]
405. #endif
406.    ldr r1, =0x0fff1010           @auto refresh off
407.    str r1, [r0, #DMC_CONCONTROL]
408.
409.    ldr r1, =0x00212100
410.    str r1, [r0, #DMC_MEMCONTROL]
411.
412.    ldr r1, =DMC0_MEMCONFIG_0
413.    str r1, [r0, #DMC_MEMCONFIG0]
414.
415.    ldr r1, =DMC0_MEMCONFIG_1
416.    str r1, [r0, #DMC_MEMCONFIG1]
417.
418.    ldr r1, =0xff000000
419.    str r1, [r0, #DMC_PRECHCONFIG]
420.
421.    ldr r1, =DMC0_TIMINGA_REF
422.    str r1, [r0, #DMC_TIMINGAREF]
423.
424.    ldr r1, =DMC0_TIMING_ROW          @TimingRow  @133MHz
425.    str r1, [r0, #DMC_TIMINGROW]
426.
427.    ldr r1, =DMC0_TIMING_DATA
428.    str r1, [r0, #DMC_TIMINGDATA]
429.
430.    ldr r1, =DMC0_TIMING_PWR          @Timing Power
431.    str r1, [r0, #DMC_TIMINGPOWER]
432.
433.    ldr r1, =0x07000000           @chip0 Deselect
434.    str r1, [r0, #DMC_DIRECTCMD]
435.
```

```
436.    ldr r1, =0x01000000             @chip0 PALL
437.    str r1, [r0, #DMC_DIRECTCMD]
438.
439.    ldr r1, =0x05000000             @chip0 REFA
440.    str r1, [r0, #DMC_DIRECTCMD]
441.
442.    ldr r1, =0x05000000             @chip0 REFA
443.    str r1, [r0, #DMC_DIRECTCMD]
444.
445.    ldr r1, =0x00000032             @chip0 MRS
446.    str r1, [r0, #DMC_DIRECTCMD]
447.
448.    ldr r1, =0x07100000             @chip1 Deselect
449.    str r1, [r0, #DMC_DIRECTCMD]
450.
451.    ldr r1, =0x01100000             @chip1 PALL
452.    str r1, [r0, #DMC_DIRECTCMD]
453.
454.    ldr r1, =0x05100000             @chip1 REFA
455.    str r1, [r0, #DMC_DIRECTCMD]
456.
457.    ldr r1, =0x05100000             @chip1 REFA
458.    str r1, [r0, #DMC_DIRECTCMD]
459.
460.    ldr r1, =0x00100032             @chip1 MRS
461.    str r1, [r0, #DMC_DIRECTCMD]
462.
463.    ldr r1, =0x0FFF20B0         @ConControl auto refresh on
464.    str r1, [r0, #DMC_CONCONTROL]
465.
466.    ldr r1, =0xFFFF00FF             @PwrdnConfig
467.    str r1, [r0, #DMC_PWRDNCONFIG]
468.
469.    ldr r1, =0x00212113             @MemControl
470.    str r1, [r0, #DMC_MEMCONTROL]
471.
472.
473.    ldr r0, =APB_DMC_1_BASE
474.
475.    ldr r1, =0x00101000             @Phycontrol0 DLL parameter setting
476.    str r1, [r0, #DMC_PHYCONTROL0]
477.
478.    ldr r1, =0x00000084             @Phycontrol1 DLL parameter setting
479.    str r1, [r0, #DMC_PHYCONTROL1]
480.
481.    ldr r1, =0x00101002             @Phycontrol2 DLL parameter setting
482.    str r1, [r0, #DMC_PHYCONTROL0]
483.
484.    ldr r1, =0x00101003             @Dll on
485.    str r1, [r0, #DMC_PHYCONTROL0]
486.
487. find_lock_val1:
488.    ldr r1, [r0, #DMC_PHYSTATUS]       @Load Phystatus register value
489.    and     r2, r1, #0x7
490.    cmp r2, #0x7                @Loop until DLL is locked
491.    bne find_lock_val1
492.
493.    and r1, #0x3fc0
494.    mov r2, r1, LSL #18
495.    orr r2, r2, #0x100000
496.    orr r2, r2, #0x1000
497.
498.    orr r1, r2, #0x3                @Force Value locking
499.    str r1, [r0, #DMC_PHYCONTROL0]
500.
501. #if 1   /* Memory margin test 10.01.05 */
502.    orr r1, r2, #0x1                @DLL off
503.    str r1, [r0, #DMC_PHYCONTROL0]
504. #endif
505.    ldr r0, =APB_DMC_1_BASE
506.
507.    ldr r1, =0x0FFF1010             @auto refresh off
508.    str r1, [r0, #DMC_CONCONTROL]
509.
510.    ldr r1, =DMC1_MEMCONTROL
511.    str r1, [r0, #DMC_MEMCONTROL]
512.
513.    ldr r1, =DMC1_MEMCONFIG_0
514.    str r1, [r0, #DMC_MEMCONFIG0]
515.
516.    ldr r1, =DMC1_MEMCONFIG_1
517.    str r1, [r0, #DMC_MEMCONFIG1]
```

```
518.
519.    ldr r1, =0xff000000
520.    str r1, [r0, #DMC_PRECHCONFIG]
521.
522.    ldr r1, =DMC1_TIMINGA_REF
523.    str r1, [r0, #DMC_TIMINGAREF]
524.
525.    ldr r1, =DMC1_TIMING_ROW          @TimingRow  @133MHz
526.    str r1, [r0, #DMC_TIMINGROW]
527.
528.    ldr r1, =DMC1_TIMING_DATA
529.    str r1, [r0, #DMC_TIMINGDATA]
530.
531.    ldr r1, =DMC1_TIMING_PWR          @Timing Power
532.    str r1, [r0, #DMC_TIMINGPOWER]
533.
534.    ldr r1, =0x07000000           @chip0 Deselect
535.    str r1, [r0, #DMC_DIRECTCMD]
536.
537.    ldr r1, =0x01000000           @chip0 PALL
538.    str r1, [r0, #DMC_DIRECTCMD]
539.
540.    ldr r1, =0x05000000           @chip0 REFA
541.    str r1, [r0, #DMC_DIRECTCMD]
542.
543.    ldr r1, =0x05000000           @chip0 REFA
544.    str r1, [r0, #DMC_DIRECTCMD]
545.
546.    ldr r1, =0x00000032           @chip0 MRS
547.    str r1, [r0, #DMC_DIRECTCMD]
548.
549.    ldr r1, =0x00020020           @chip0 EMRS
550.    str r1, [r0, #DMC_DIRECTCMD]
551.
552.    ldr r1, =0x07100000           @chip1 Deselect
553.    str r1, [r0, #DMC_DIRECTCMD]
554.
555.    ldr r1, =0x01100000           @chip1 PALL
556.    str r1, [r0, #DMC_DIRECTCMD]
557.
558.    ldr r1, =0x05100000           @chip1 REFA
559.    str r1, [r0, #DMC_DIRECTCMD]
560.
561.    ldr r1, =0x05100000           @chip1 REFA
562.    str r1, [r0, #DMC_DIRECTCMD]
563.
564.    ldr r1, =0x00100032           @chip1 MRS
565.    str r1, [r0, #DMC_DIRECTCMD]
566.
567.    ldr r1, =0x00120020           @chip0 EMRS
568.    str r1, [r0, #DMC_DIRECTCMD]
569.
570.    ldr r1, =0x0FFF10B0           @ConControl auto refresh on
571.    str r1, [r0, #DMC_CONCONTROL]
572.
573.    ldr r1, =0xFFFF00FF           @PwrdnConfig
574.    str r1, [r0, #DMC_PWRDNCONFIG]
575.
576.    ldr r1, =0x00212113           @MemControl
577.    str r1, [r0, #DMC_MEMCONTROL]
578.
579. #endif  /* CONFIG_MCP_AC / CONFIG_MCP_H / CONFIG_MCP_B / CONFIG_MCP_D / CONFIG_MCP_N */
580.
581.    mov pc, lr
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.       .align 5
593. .global v7_flush_dcache_all
594. v7_flush_dcache_all:
595.
596.    ldr r0, =0xffffffff
597.    mrc p15, 1, r0, c0, c0, 1       @ Read CLIDR
598.    ands    r3, r0, #0x7000000
599.    mov r3, r3, LSR #23           @ Cache level value (naturally aligned)
```

```
600.    beq     Finished
601.    mov r10, #0
602. Loop1:
603.    add r2, r10, r10, LSR #1        @ Work out 3xcachelevel
604.    mov r1, r0, LSR r2              @ bottom 3 bits are the Ctype for this level
605.    and r1, r1, #7                  @ get those 3 bits alone
606.    cmp r1, #2
607.    blt Skip                        @ no cache or only instruction cache at this level
608.    mcr p15, 2, r10, c0, c0, 0      @ write the Cache Size selection register
609.    mov r1, #0
610.    mcr p15, 0, r1, c7, c5, 4       @ PrefetchFlush to sync the change to the CacheSizeID reg
611.    mrc p15, 1, r1, c0, c0, 0       @ reads current Cache Size ID register
612.    and r2, r1, #0x7                @ extract the line length field
613.    add r2, r2, #4                  @ add 4 for the line length offset (log2 16 bytes)
614.    ldr r4, =0x3FF
615.    ands    r4, r4, r1, LSR #3      @ R4 is the max number on the way size (right aligned)
616.    clz r5, r4                      @ R5 is the bit position of the way size increment
617.    ldr r7, =0x00007FFF
618.    ands    r7, r7, r1, LSR #13     @ R7 is the max number of the index size (right aligned)
619. Loop2:
620.    mov r9, r4                      @ R9 working copy of the max way size (right aligned)
621. Loop3:
622.    orr r11, r10, r9, LSL r5        @ factor in the way number and cache number into R11
623.    orr r11, r11, r7, LSL r2        @ factor in the index number
624.    mcr p15, 0, r11, c7, c6, 2      @ invalidate by set/way
625.    subs    r9, r9, #1              @ decrement the way number
626.    bge Loop3
627.    subs    r7, r7, #1              @ decrement the index
628.    bge Loop2
629. Skip:
630.    add r10, r10, #2                @ increment the cache number
631.    cmp r3, r10
632.    bgt Loop1
633. Finished:
634.    mov pc, lr
635.
636.        .align  5
637. .global disable_l2cache
638. disable_l2cache:
639.    mrc     p15, 0, r0, c1, c0, 1
640.    bic     r0, r0, #(1<<1)
641.    mcr     p15, 0, r0, c1, c0, 1
642.    mov pc, lr
643.
644.
645.        .align  5
646. .global enable_l2cache
647. enable_l2cache:
648.    mrc     p15, 0, r0, c1, c0, 1
649.    orr     r0, r0, #(1<<1)
650.    mcr     p15, 0, r0, c1, c0, 1
651.    mov     pc, lr
652.
653.        .align  5
654. .global set_l2cache_auxctrl
655. set_l2cache_auxctrl:
656.    mov r0, #0x0
657.    mcr     p15, 1, r0, c9, c0, 2
658.    mov     pc, lr
659.
660.        .align  5
661. .global set_l2cache_auxctrl_cycle
662. set_l2cache_auxctrl_cycle:
663.    mrc     p15, 1, r0, c9, c0, 2
664.    bic     r0, r0, #(0x1<<29)
665.    bic     r0, r0, #(0x1<<21)
666.    bic     r0, r0, #(0x7<<6)
667.    bic     r0, r0, #(0x7<<0)
668.    mcr     p15, 1, r0, c9, c0, 2
669.    mov     pc,lr
670.
671.    .align 5
672. CoInvalidateDCacheIndex:
673.    ;
674.    mcr     p15, 0, r0, c7, c6, 2
675.    mov     pc,lr
676.
677.
678.
679.    .globl cleanDCache
680. cleanDCache:
681.    mrc p15, 0, pc, c7, c10, 3
```

```
682.    bne cleanDCache
683.    mov pc, lr
684.
685.    .globl cleanFlushDCache
686. cleanFlushDCache:
687.    mrc p15, 0, pc, c7, c14, 3
688.    bne cleanFlushDCache
689.    mov pc, lr
690.
691.    .globl cleanFlushCache
692. cleanFlushCache:
693.    mrc p15, 0, pc, c7, c14, 3
694.    bne cleanFlushCache
695.    mcr p15, 0, r0, c7, c5, 0
696.    mov pc, lr
697.
698.    .ltorg
699.
700.
701.
702.    .globl ledon_1
703. ledon_1:
704.
705.    ldr r0, =ELFIN_GPIO_BASE
706.    ldr r1, =0x11000
707.    str r1, [r0, #GPC0CON_OFFSET]
708.    ldr r2, [r0, #GPC0CON_OFFSET]
709.    orr r2, r2, #0x08
710.    str r2, [r0, #GPC0DAT_OFFSET]
711.    mov pc, lr
712.
713.    .globl ledon_2
714. ledon_2:
715.
716.    ldr r0, =ELFIN_GPIO_BASE
717.    ldr r1, =0x11000
718.    str r1, [r0, #GPC0CON_OFFSET]
719.    ldr r2, [r0, #GPC0CON_OFFSET]
720.    orr r2, r2, #0x10
721.    str r2, [r0, #GPC0DAT_OFFSET]
722.    mov pc, lr
723.
724.    .globl ledon
725. ledon:
726.
727.    ldr r0, =ELFIN_GPIO_BASE
728.    ldr r1, =0x11000
729.    str r1, [r0, #GPC0CON_OFFSET]
730.    ldr r2, =0x18
731.    str r2, [r0, #GPC0DAT_OFFSET]
732.    mov pc, lr
733.
734.    .globl ledoff
735. ledoff:
736.
737.    ldr r0, =ELFIN_GPIO_BASE
738.    ldr r1, =0x11000
739.    str r1, [r0, #GPC0CON_OFFSET]
740.    ldr r2, =0x00
741.    str r2, [r0, #GPC0DAT_OFFSET]
742.    mov pc, lr
```

crt0.S的完整代码(这个代码u-boot本身就有，并且不需要改动，但这里也贴出来):

```
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
```

```
19.
20.
21.
22.
23.
24.
25. #include <config.h>
26. #include <asm-offsets.h>
27.
28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73. #if defined(CONFIG_NAND_SPL)
74.
75. .globl nand_boot
76.
77.
78. #elif !defined(CONFIG_SPL_BUILD)
79.
80. .globl board_init_r
81.
82. #endif
83.
84.
85.
86.
87.
88. .globl __bss_start
89. .globl __bss_end__
90.
91.
92.
93.
94.
95. .global _main
96.
97. _main:
98.
99.
100.
```

```
101.
102.
103. #if defined(CONFIG_NAND_SPL)
104.
105.    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
106. #elif defined(CONFIG_SPL_BUILD) && defined(CONFIG_SPL_STACK)
107.    ldr sp, =(CONFIG_SPL_STACK)
108. #else
109.    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
110. #endif
111.    bic sp, sp, #7
112.    sub sp, #GD_SIZE
113.    bic sp, sp, #7
114.    mov r8, sp
115.    mov r0, #0
116.    bl  board_init_f
117.
118. #if ! defined(CONFIG_SPL_BUILD)
119.
120.
121.
122.
123.
124.
125.
126.    ldr sp, [r8, #GD_START_ADDR_SP]
127.    bic sp, sp, #7
128.    ldr r8, [r8, #GD_BD]
129.    sub r8, r8, #GD_SIZE
130.
131.    adr lr, here
132.    ldr r0, [r8, #GD_RELOC_OFF]
133.    add lr, lr, r0
134.    ldr r0, [r8, #GD_START_ADDR_SP]
135.    mov r1, r8
136.    ldr r2, [r8, #GD_RELOCADDR]
137.    b   relocate_code
138. here:
139.
140.
141.
142.    bl  c_runtime_cpu_setup
143.
144.    ldr r0, =__bss_start
145.    ldr r1, =__bss_end__
146.
147.    mov r2, #0x00000000
148.
149. clbss_l:cmp r0, r1
150.    strlo   r2, [r0]
151.    addlo   r0, r0, #4
152.    blo clbss_l
153.
154.    bl coloured_LED_init
155.    bl red_led_on
156.
157. #if defined(CONFIG_NAND_SPL)
158.
159.
160.    ldr     pc, =nand_boot
161.
162. #else
163.
164.    mov r0, r8
165.    ldr r1, [r8, #GD_RELOCADDR]
166.
167.    ldr pc, =board_init_r
168.
169. #endif
170.
171.
172.
173. #endif
```