uboot启动过程 **1 - kehuadong -** 博客园

kehuadong 关注 - 0 粉丝 - 0 +加关注

```
b reset
ldr pc, _undefined_instruction
ldr pc, _software_interrupt
ldr pc, _prefetch_abort
ldr pc, _data_abort
ldr pc, _not_used
ldr pc, _irq
ldr pc, _fiq

void s_init(void)
{
	struct anatop_regs *anatop = (struct anatop_regs
*)ANATOP_BASE_ADDR;
	struct mxc_ccm_reg *ccm = (struct mxc_ccm_reg *)CCM_BASE_ADDR;
	u32 mask480;
	u32 reg, periph1, periph2;

	if (is_cpu_type(MXC_CPU_MX6SX) || is_cpu_type(MXC_CPU_MX6UL)
||
		is_cpu_type(MXC_CPU_MX6ULL) ||
is_cpu_type(MXC_CPU_MX6SLL))
		return;		// 对于MXC_CPU_MX6ULL, 这里什么都没做

	/* Due to hardware limitation, on MX6Q we need to gate/ungate all PFDs
	* to make sure PFD is working right, otherwise, PFDs may
	* not output clock after reset, MX6DL and MX6SL have added 396M pfd
	* workaround in ROM code, as bus clock need it
	*/

	mask480 = ANATOP_PFD_CLKGATE_MASK(0) |
		ANATOP_PFD_CLKGATE_MASK(1) |
		ANATOP_PFD_CLKGATE_MASK(2) |
		ANATOP_PFD_CLKGATE_MASK(3);
	mask528 = ANATOP_PFD_CLKGATE_MASK(1) |
	ANATOP_PFD_CLKGATE_MASK(3);

	reg = readl(&ccm->cbcmr);
	periph2 = ((reg &
MXC_CCM_CBCMR_PRE_PERIPH2_CLK_SEL_MASK)
		>> MXC_CCM_CBCMR_PRE_PERIPH2_CLK_SEL_OFFSET);
	periph1 = ((reg &
MXC_CCM_CBCMR_PRE_PERIPH_CLK_SEL_MASK)
		>> MXC_CCM_CBCMR_PRE_PERIPH_CLK_SEL_OFFSET);
```

```
	/* Checking if PLL2 PFD0 or PLL2 PFD2 is using for periph clock */
	if ((periph2 != 0x2) && (periph1 != 0x2))
		mask528 |= ANATOP_PFD_CLKGATE_MASK(0);

	if ((periph2 != 0x1) && (periph1 != 0x1) &&
		(periph2 != 0x3) && (periph1 != 0x3))
		mask528 |= ANATOP_PFD_CLKGATE_MASK(2);

	writel(mask480, &anatop->pfd_480_set);
	writel(mask528, &anatop->pfd_528_set);
	writel(mask480, &anatop->pfd_480_clr);
	writel(mask528, &anatop->pfd_528_clr);
}

ENTRY(_main)

	// Set up initial C runtime environment and call board_init_f(0).

#if defined(CONFIG_SPL_BUILD) && defined(CONFIG_SPL_STACK)
	ldr sp, =(CONFIG_SPL_STACK)
#else
	ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)	// 设置 sp 指针为
CONFIG_SYS_INIT_SP_ADDR，也就是 sp 指向 0X0091FF00

#endif
#if defined(CONFIG_CPU_V7M) /* v7M forbids using SP as BIC destination */
	mov r3, sp
	bic r3, r3, #7
	mov sp, r3
#else
	bic sp, sp, #7 /* 8-byte alignment for ABI compliance */	// sp 做 8 字节
对齐

#endif
	mov r0, sp			// r0=0X0091FF00

	bl board_init_f_alloc_reserve	// board_init.c中， 参数为r0的值

	mov sp, r0			// top=0X0091FA00, 内存布局为gd_t:
[0X0091FA00, 0X0091FB00-8)  alloc:[0X0091FB00, 0X0091FF00)

	/* set up gd here, outside any C code */

	// include/asm-generic/global_data.h中定义了 typedef struct global_data
{...} gd_t;

	// arch/arm/include/asm/global_data.h中有#define
DECLARE_GLOBAL_DATA_PTR register volatile gd_t *gd asm ("r9"), 地址
0X0091FA00

	mov r9, r0

	bl board_init_f_init_reserve	// board_init.c中， 参数为r0的值, 最终
```

gd->malloc_base=0X0091FB00这个也就是 **early malloc** 的起始地址

```
    mov r0, #0
    bl board_init_f                 // 函数定义在文件 common/board_f.c 中！主要用来初始化 DDR，定时器，完成代码拷贝

#if ! defined(CONFIG_SPL_BUILD)

    // 重新设置环境(sp 和 gd)、获取 gd->start_addr_sp 的值赋给 sp，在函数 board_init_f

    // 中会初始化 gd 的所有成员变量，其中 gd->start_addr_sp=0X9EF44E90， 所以这里相当于设置

    // sp=gd->start_addr_sp=0X9EF44E90。0X9EF44E90 是 DDR 中的地址，说明新的 sp 和 gd 将会存

    // 放到 DDR 中，而不是内部的 RAM 了。GD_START_ADDR_SP=64

    ldr sp, [r9, #GD_START_ADDR_SP] /* sp = gd->start_addr_sp */
#if defined(CONFIG_CPU_V7M) /* v7M forbids using SP as BIC destination */
    mov r3, sp
    bic r3, r3, #7
    mov sp, r3
#else
    bic sp, sp, #7 /* 8-byte alignment for ABI compliance */
#endif

// 获取 gd->bd 的地址赋给 r9，此时 r9 存放的是老的 gd，这里通过获取 gd->bd 的地址来计算出新的 gd 的位置。GD_BD=0

    ldr r9, [r9, #GD_BD] /* r9 = gd->bd */

    // 新的 gd 在 bd 下面，所以 r9 减去 gd 的大小就是新的 gd 的位置，获取到新的 gd的位置以后赋值给 r9。

    sub r9, r9, #GD_SIZE /* new GD is below bd */

    adr lr, here        // 后面执行其他函数返回here

    ldr r0, [r9, #GD_RELOC_OFF] /* r0 = gd->reloc_off */

    // lr 中的 here 要使用重定位后的位置

    add lr, lr, r0
#if defined(CONFIG_CPU_V7M)
    orr lr, #1 /* As required by Thumb-only */
#endif
    ldr r0, [r9, #GD_RELOCADDR] /* r0 = gd->relocaddr */    // uboot要从0x87800000复制到0X9FF47000

    b relocate_code         //  arch/arm/lib/relocate.S
```

```
here:
	/*
	* now relocate vectors
	*/

	bl relocate_vectors // arch/arm/lib/relocate.S

	/* Set up final (full) environment */

	bl c_runtime_cpu_setup /* we still call old routine here */	// arch/arm
/cpu/armv7/start.S

#endif
#if !defined(CONFIG_SPL_BUILD) || defined(CONFIG_SPL_FRAMEWORK)
# ifdef CONFIG_SPL_BUILD
	/* Use a DRAM stack for the rest of SPL, if requested */
	bl spl_relocate_stack_gd
	cmp r0, #0
	movne sp, r0
	movne r9, r0
# endif

	// 清除bss开始
	ldr r0, =__bss_start /* this is auto-relocated! */

#ifdef CONFIG_USE_ARCH_MEMSET
	ldr r3, =__bss_end /* this is auto-relocated! */
	mov r1, #0x00000000 /* prepare zero to clear BSS */

	subs r2, r3, r0 /* r2 = memset len */
	bl memset
#else
	ldr r1, =__bss_end /* this is auto-relocated! */
	mov r2, #0x00000000 /* prepare zero to clear BSS */

	clbss_l:cmp r0, r1 /* while not at end of BSS */
#if defined(CONFIG_CPU_V7M)
	itt lo
#endif
	strlo r2, [r0] /* clear 32-bit BSS word */
	addlo r0, r0, #4 /* move to next */
	blo clbss_l

   // 清除bss完成
#endif

#if ! defined(CONFIG_SPL_BUILD)
	bl coloured_LED_init
	bl red_led_on
#endif

	// 第一个参数是 gd，因此读取 r9 保存到 r0 里面
```

```
	// 设置函数 board_init_r 的第二个参数是目的地址，因此 r1=
gd->relocaddr

	/* call board_init_r(gd_t *id, ulong dest_addr) */
	mov r0, r9 /* gd_t */
	ldr r1, [r9, #GD_RELOCADDR] /* dest_addr */
	/* call board_init_r */
#if defined(CONFIG_SYS_THUMB_BUILD)
	ldr lr, =board_init_r /* this is auto-relocated! */
	bx lr
#else
	ldr pc, =board_init_r /* this is auto-relocated! */
#endif
	/* we should not return here. */
#endif

ENDPROC(_main)
```

可见**_main**主要是 调用了**board_init_f**、**relocate_code**、**relocate_vectors** 和 **board_init_r** 这 **4** 个函数

board_init.c中的**board_init_f_alloc_reserve**

```
ulong board_init_f_alloc_reserve(ulong top)
{
	/* Reserve early malloc arena */
#if defined(CONFIG_SYS_MALLOC_F)

// CONFIG_SYS_MALLOC_F_LEN=0X400( 在文件 include/generated
/autoconf.h

	top -= CONFIG_SYS_MALLOC_F_LEN;
#endif
	/* LAST : reserve GD (rounded up to a multiple of 16 bytes) */

	//  sizeof(struct global_data)=248(GD_SIZE 值),  最终top = 0X0091FF00
-> top=0X0091FA00

	top = rounddown(top-sizeof(struct global_data), 16);

	return top;
}
```

board_init.c中的**board_init_f_init_reserve**

```
void board_init_f_init_reserve(ulong base)
{
	struct global_data *gd_ptr;
#ifndef _USE_MEMCPY
	int *ptr;
#endif
```

```
        /*
        * clear GD entirely and set it up.
        * Use gd_ptr, as gd may not be properly set yet.
        */

        gd_ptr = (struct global_data *)base;
        /* zero the area */
#ifdef _USE_MEMCPY
        memset(gd_ptr, '\0', sizeof(*gd));
#else
        for (ptr = (int *)gd_ptr; ptr < (int *)(gd_ptr + 1); )
                *ptr++ = 0;
#endif
        /* set GD unless architecture did it already */
#if !defined(CONFIG_ARM)
        arch_setup_gd(gd_ptr);
#endif
        /* next alloc will be higher by one GD plus 16-byte alignment */
        base += roundup(sizeof(struct global_data), 16);

        /*
        * record early malloc arena start.
        * Use gd as it is now properly set for all architectures.
        */

#if defined(CONFIG_SYS_MALLOC_F)
        /* go down one 'early malloc arena' */
```

// arch/arm/include/asm/global_data.h中有

```
//  #define DECLARE_GLOBAL_DATA_PTR register volatile gd_t *gd asm
("r9")
```

// 所以**gd**的指针来自这里

```
        gd->malloc_base = base;          // 16字节对齐后
```

gd->malloc_base=0X0091FB00*这个也就是* ***early malloc*** *的起始地址*

```
        /* next alloc will be higher by one 'early malloc arena' size */
        base += CONFIG_SYS_MALLOC_F_LEN;
#endif
}
```

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

```
ENTRY(c_runtime_cpu_setup)
/*
* If I-cache is enabled invalidate it
*/
#ifndef CONFIG_SYS_ICACHE_OFF
        mcr p15, 0, r0, c7, c5, 0 @ invalidate icache
        mcr p15, 0, r0, c7, c10, 4 @ DSB
```

```
        mcr p15, 0, r0, c7, c5, 4 @ ISB
#endif

        bx lr

ENDPROC(c_runtime_cpu_setup)
```