U-boot

Bootload

- What is bootloader
- Boot : short bootstrap
  - Initialize basic of SOC (CPU, RAM, CLK)
- Down loader
  - Download Image or Application to ram from host (developer host PC)
- Loader
  - Load OS to ram form volatile memory

All kinds of embedded Linux bootloader

- **U-boot**
- UEFI
- Redboot
- Stubby (Linaro) …
- Anyway, they are same target
  - Load and boot OS to RAM from storage

Concepts of the Boot Loader

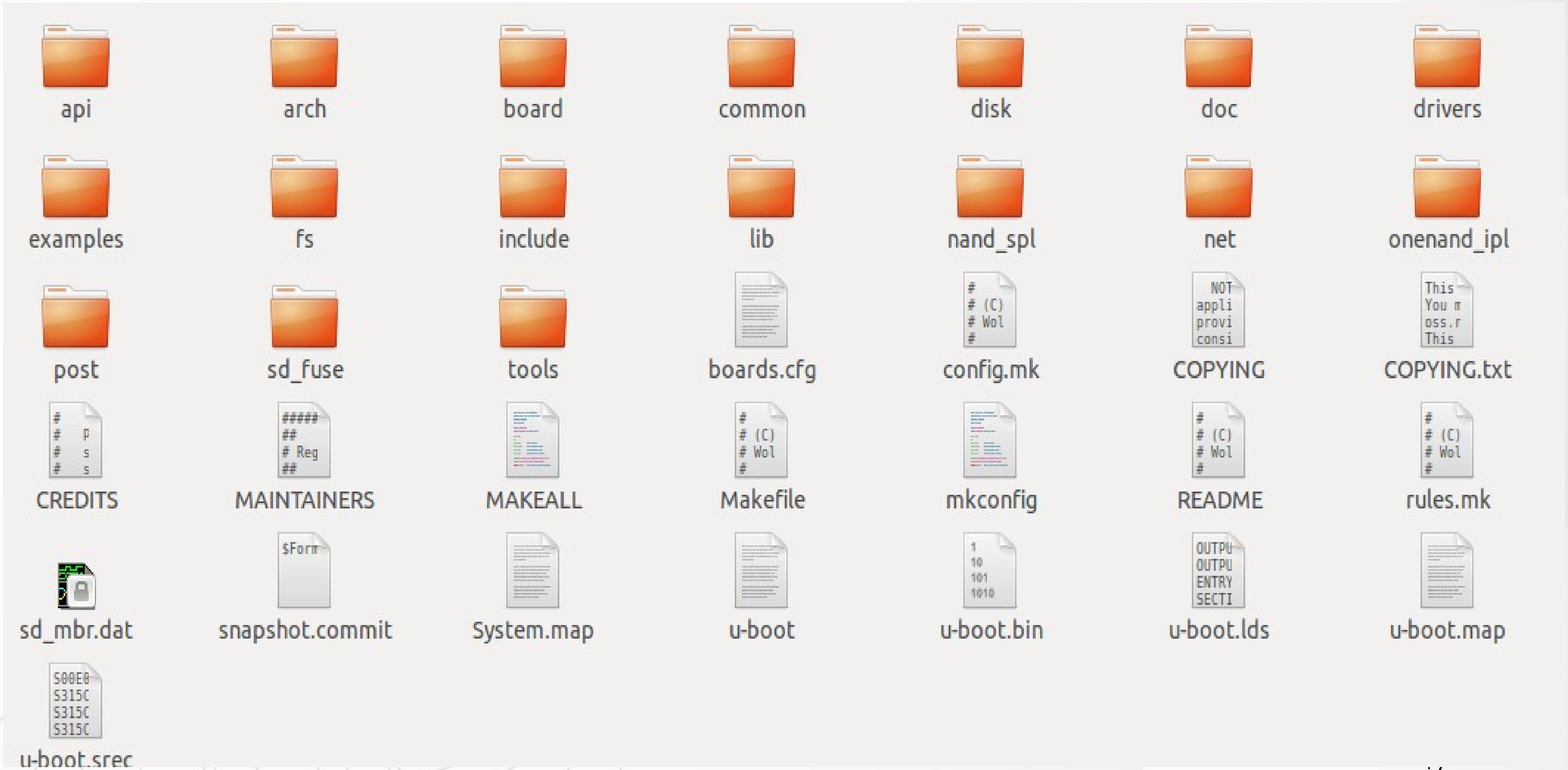
- Boot Loader is varied from CPU to CPU,from board to board.
- All the system software and data are stored in some kind of nonvolatile memory.
- Operation Mode of Boot Loader
  - Boot : Initialize basic of SOC
  - Load : load OS to RAM

Introduce U-boot

U-boot

- Das U-Boot -- the Universal Boot Loader
- http://www.denx.de/wiki/U-Boot
- GitHub for u-boot
- Open Source follow GPL
- Supply many CPU
    - PPC, ARM, x86, MIPS, AVR32 …
- Supply basic periphery devices
    - UART, Flash, SD/MMC ….

u-boot directory structure



```
api        arch         board       common      disk      doc         drivers
examples   fs           include     lib         nand_spl  net         onenand_ipl
post       sd_fuse      tools       boards.cfg  config.mk COPYING     COPYING.txt
CREDITS    MAINTAINERS  MAKEALL     Makefile    mkconfig  README      rules.mk
sd_mbr.dat snapshot.commit System.map u-boot    u-boot.bin u-boot.lds u-boot.map
u-boot.srec
```

u-boot directory structure

- **Arch**
  - Many types CPU : Arm, mips, i386 ...
- **Board**
  - Many types develop board : Samsung, ti, davinci ...
- Tools
  - Make Image (u-boot, linux) or S-Record image tool
- **Drivers**
  - Some HW control code

u-boot directory structure

- **Common**
  - Major command and relation environment setting source code
- Api
  - Implement unrelated hardware code
- nand_spl, onenand_ipl
  - Related nand/onenand flash control
- Example
  - Standalone application

u-boot directory structure

- Post
  - Supply Power On Self Test function
- Fs
  - Supply file system : fat, jffs2, ext2, cramfs
- Lib
  - General public library : CRC32/16, bzlib, ldiv ..
- Disk
  - Supply disk driver and partition handling

u-boot directory structure about tiny4412

- **arch/arm/cpu/armv7/exynos/**
  - Samsung exynos CPU related
  - Clock, i2c, irom, mmc, emmc ...
- **board/samsung/tiny4412/**
  - Tiny4412 EVB related
  - Low level init, memory init, link script ...
- Common
  - Tiny4412 u-boot command related
- **include/configs/tiny4412.h**
  - Tiny4412 EVB build configure related

How to build u-boot

- Clear
  - **#make distclean**
- Configure
  - **#make BoardName_config**
  - **#make tiny4412_config**
- Build
  - **#make -j4**
- Result of build
  - u-boot : ELF format file
  - **u-boot.bin** : raw data binary

Operating U-boot

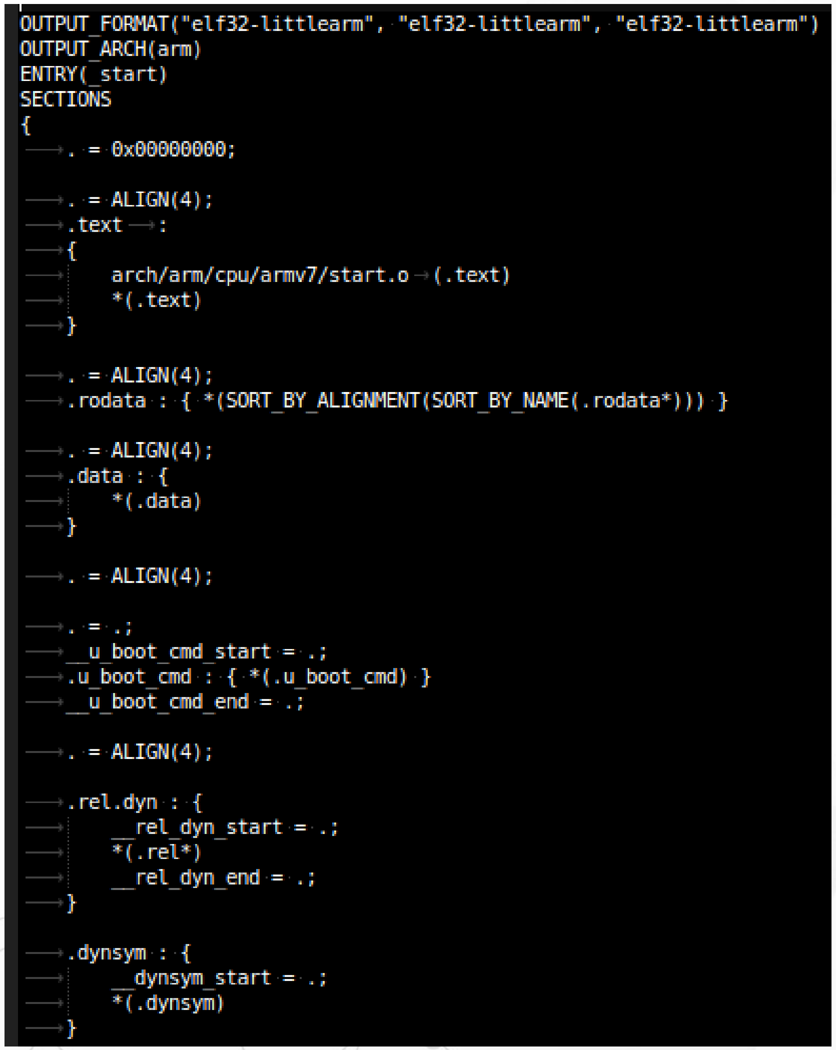
- Understand and use command
- Understand and modify parameters
- Run Application

Link Script

- u-boot.lds
  - board/samsung/tiny4412/u-boot.lds
  - The link script will pack all into binary.
  - The binary file will put in storage.
  - The start address (.TEXT) can be modified.
- CONFIG_SYS_TEXT_BASE
  - board/samsung/tiny4412/config.mk
  - Check u-boot.map

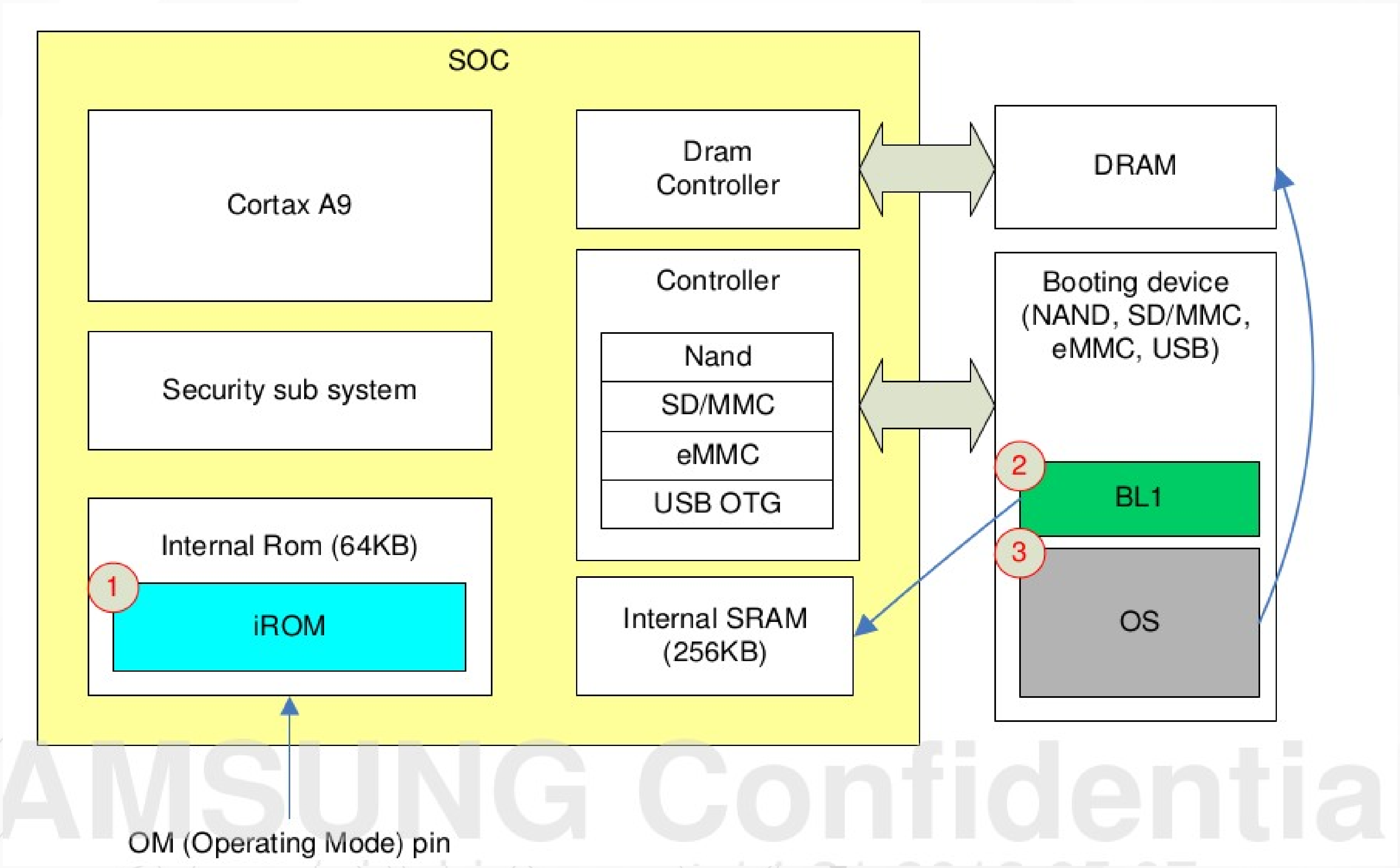
Link Script

arch/arm/cpu/armv7

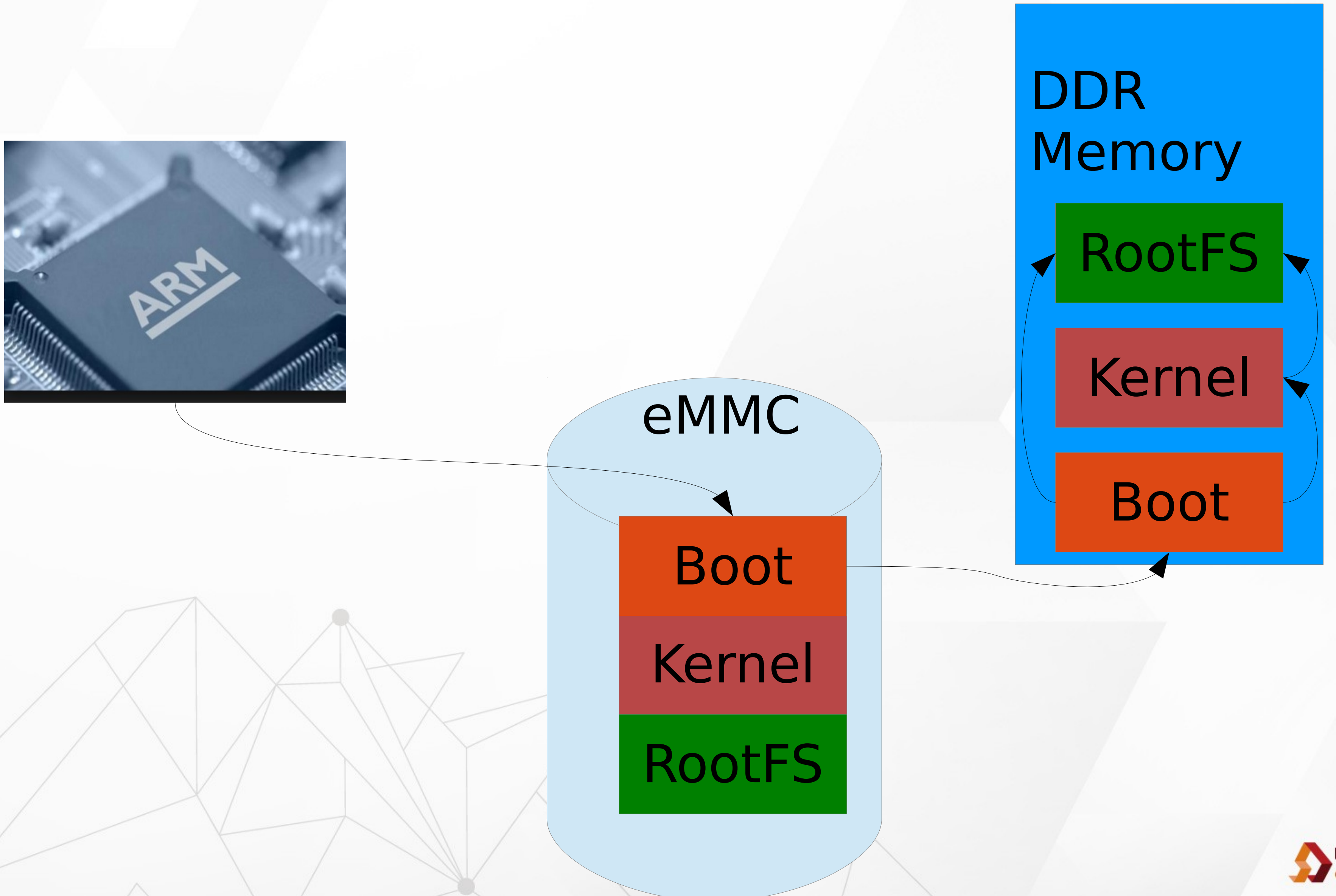


```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
	. = 0x00000000;

	. = ALIGN(4);
	.text	:
	{
		arch/arm/cpu/armv7/start.o	(.text)
		*(.text)
	}

	. = ALIGN(4);
	.rodata : { *(SORT_BY_ALIGNMENT(SORT_BY_NAME(.rodata*))) }

	. = ALIGN(4);
	.data : {
		*(.data)
	}

	. = ALIGN(4);

	. = .;
	__u_boot_cmd_start = .;
	.u_boot_cmd : { *(.u_boot_cmd) }
	__u_boot_cmd_end = .;

	. = ALIGN(4);

	.rel.dyn : {
		__rel_dyn_start = .;
		*(.rel*)
		__rel_dyn_end = .;
	}

	.dynsym : {
		__dynsym_start = .;
		*(.dynsym)
	}
```

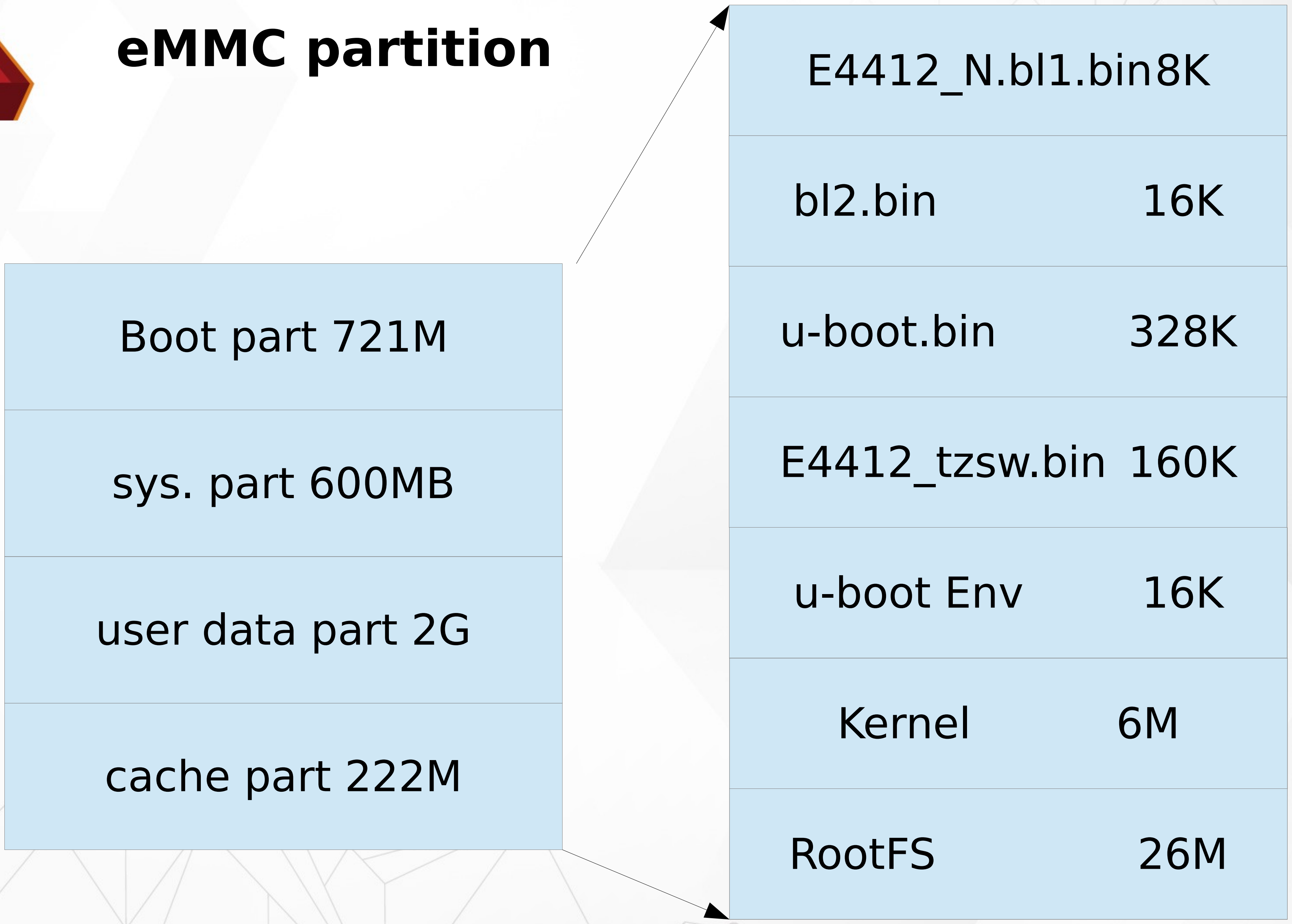
Exynos 4412 Booting Sequence



Loading through

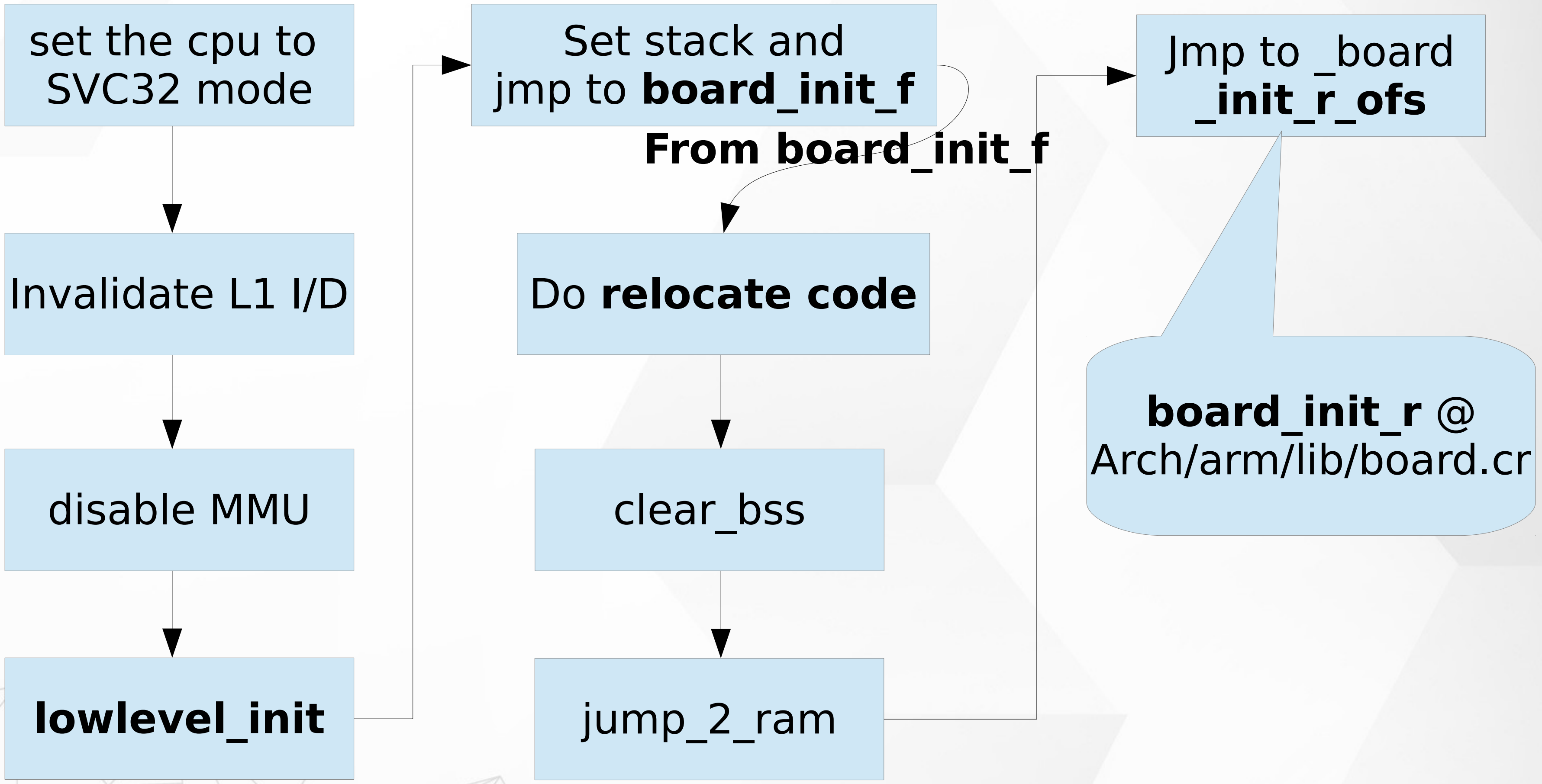


eMMC partition



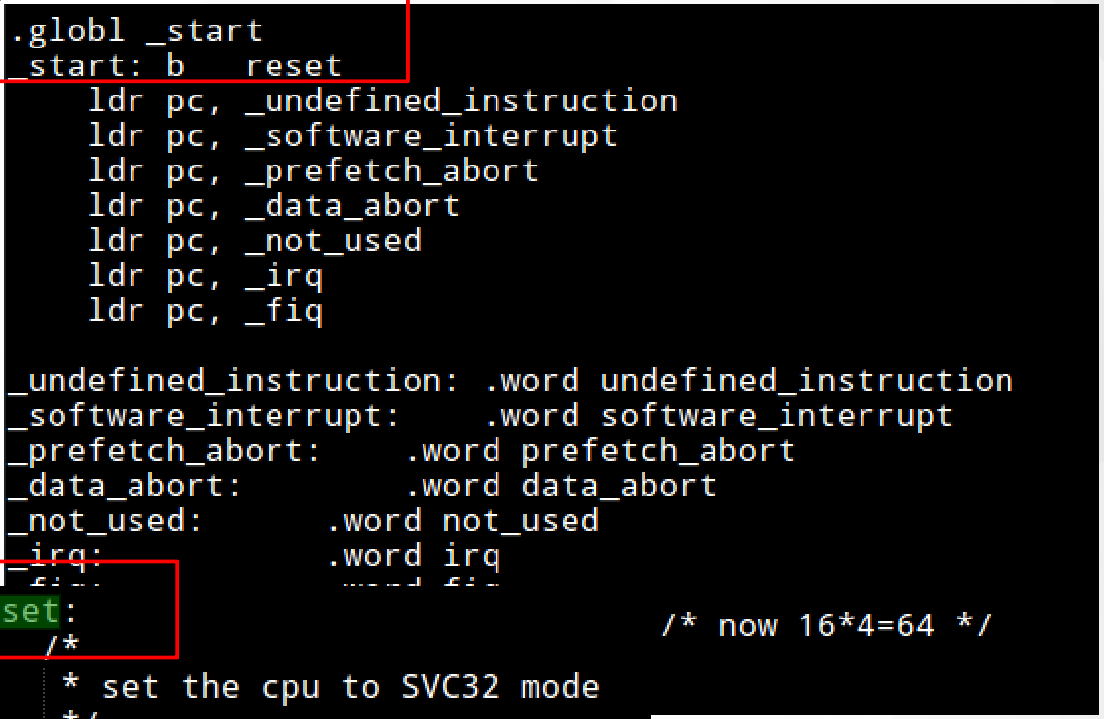
U-boot initialize sequence

U-boot start up sequence

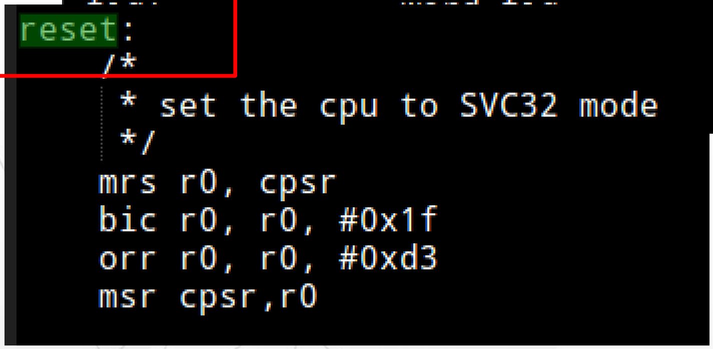

set the cpu to SVC32 mode
Invalidate L1 I/D
disable MMU
lowlevel_init
Set stack and jmp to **board_init_f**
From board_init_f
Do **relocate code**
clear_bss
jump_2_ram
Jmp to _board **_init_r_ofs**
board_init_r @ Arch/arm/lib/board.cr

CPU will start from power on

```
uboot_tiny4412-
master/arch/arm/cpu/armv7
├── config.mk
├── cpu.c
├── exynos
│   ├── ace_sha1.c
│   ├── clock.c
│   ├── gpio.c
│   ├── i2c.c
│   ├── irom_copy.c
~~~~~~~~~~~~~~~~   ~~~~~
│   ├── Makefile
│   ├── movi_pa   tion.c
│   ├── nand.c
│   ├── nand_cp.c
│   └── UBOOT_SB20_S5PC210S.h
├── start.S
├── syslib.c
└── u-boot.lds
```



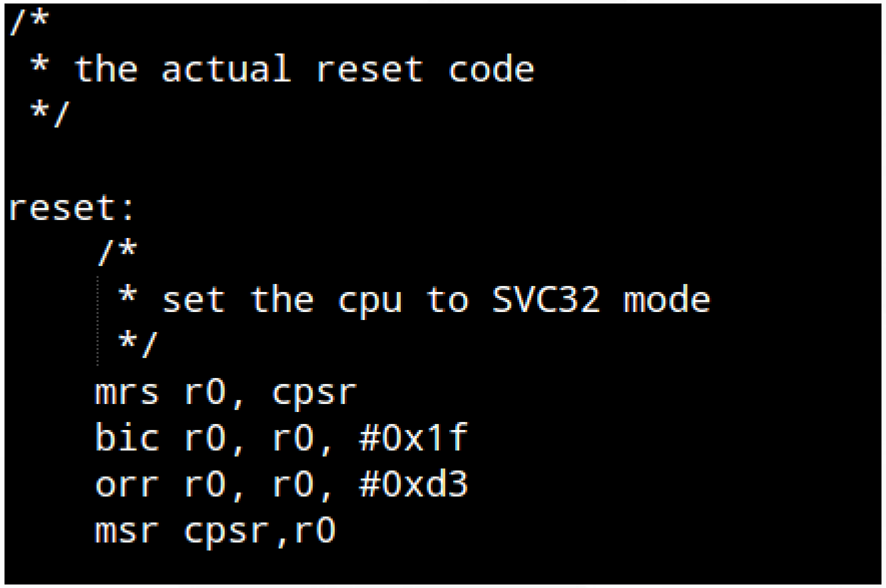
```
.globl _start
_start: b    reset
    ldr pc, _undefined_instruction
    ldr pc, _software_interrupt
    ldr pc, _prefetch_abort
    ldr pc, _data_abort
    ldr pc, _not_used
    ldr pc, _irq
    ldr pc, _fiq

_undefined_instruction:  .word undefined_instruction
_software_interrupt:     .word software_interrupt
_prefetch_abort:     .word prefetch_abort
_data_abort:         .word data_abort
_not_used:       .word not_used
_irq:            .word irq
```



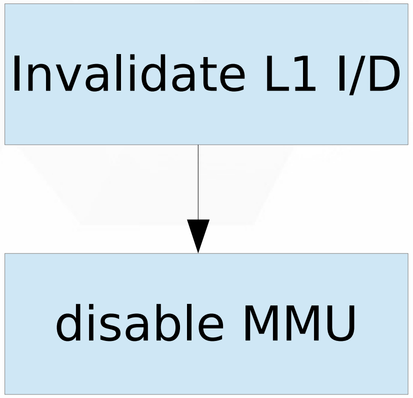
```
reset:
    /*
     * set the cpu to SVC32 mode
     */
    mrs r0, cpsr
    bic r0, r0, #0x1f
    orr r0, r0, #0xd3
    msr cpsr,r0
```

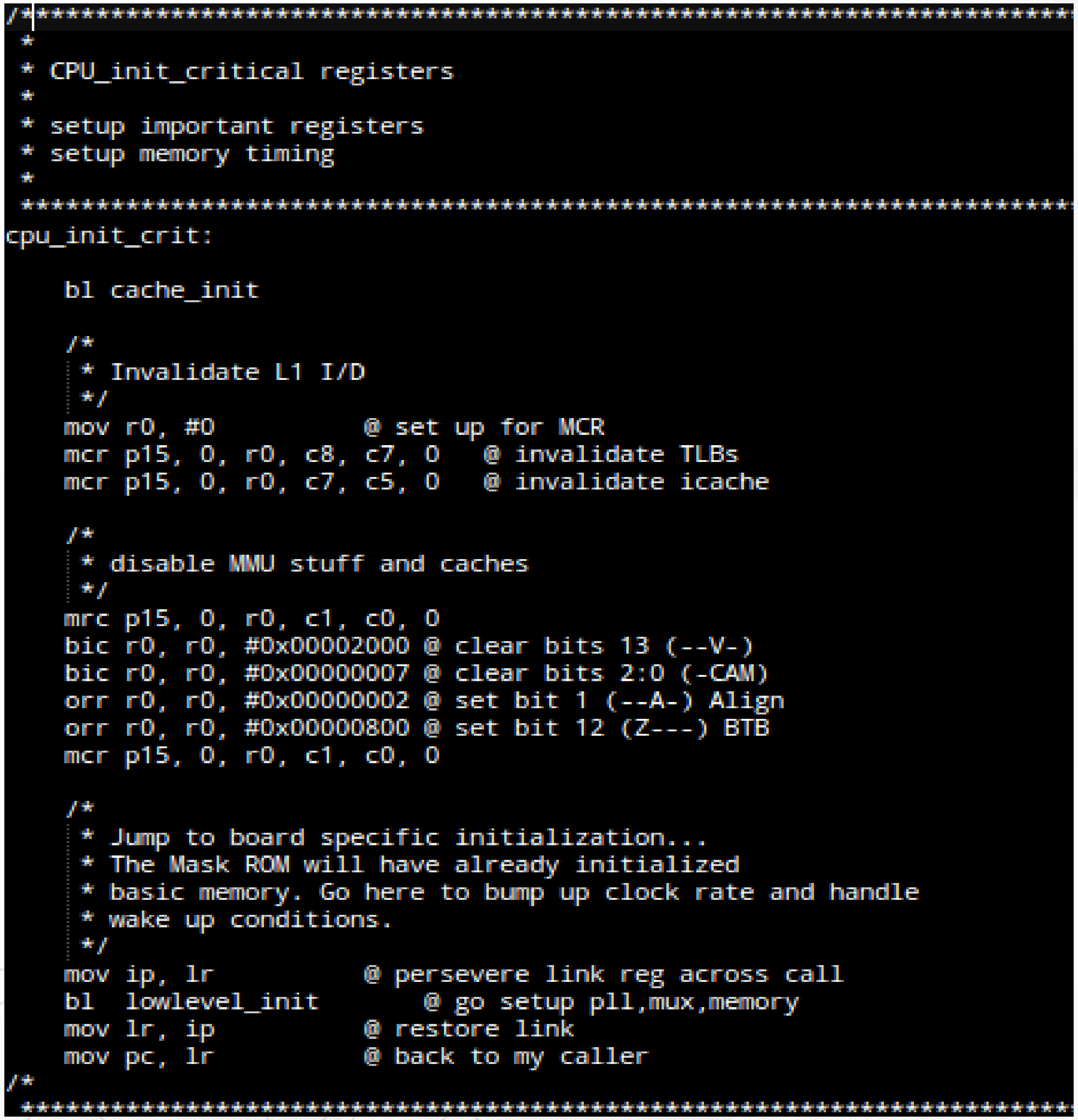
```
                        /* now 16*4=64 */
```

set the cpu to SVC32 mode



```
/*
 * the actual reset code
 */

reset:
    /*
     * set the cpu to SVC32 mode
     */
    mrs r0, cpsr
    bic r0, r0, #0x1f
    orr r0, r0, #0xd3
    msr cpsr,r0
```



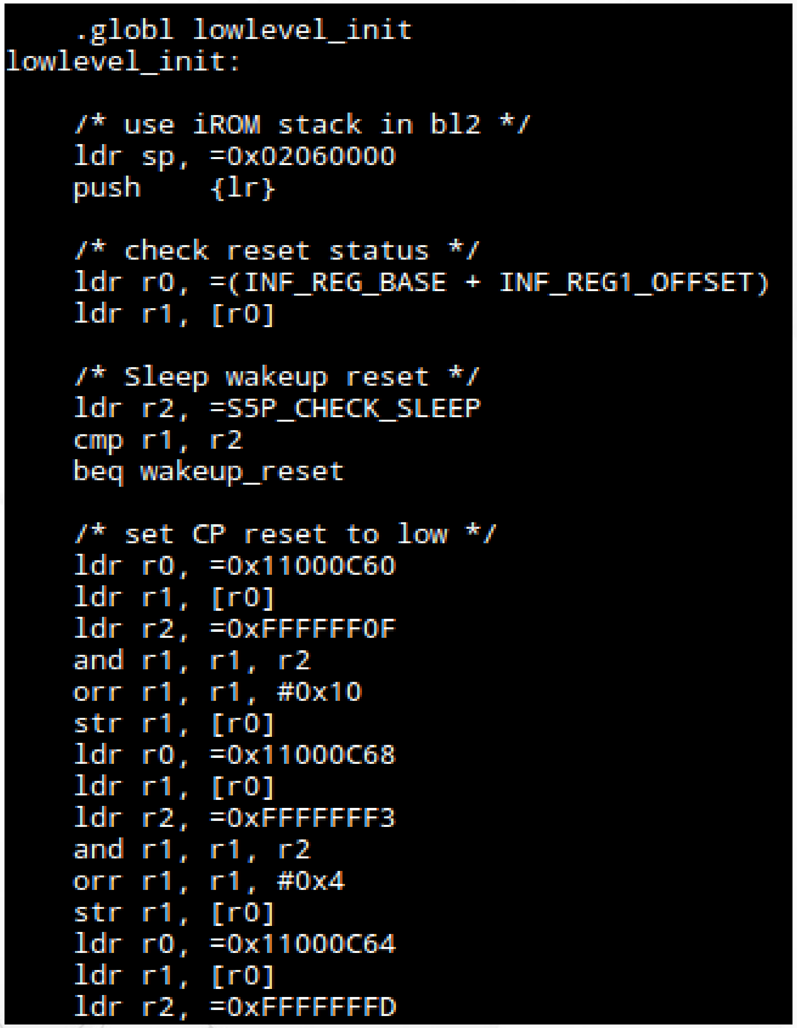


```
/*************************************************************************
 *
 * CPU_init_critical registers
 *
 * setup important registers
 * setup memory timing
 *
 *************************************************************************
cpu_init_crit:

    bl cache_init

    /*
     * Invalidate L1 I/D
     */
    mov r0, #0          @ set up for MCR
    mcr p15, 0, r0, c8, c7, 0   @ invalidate TLBs
    mcr p15, 0, r0, c7, c5, 0   @ invalidate icache

    /*
     * disable MMU stuff and caches
     */
    mrc p15, 0, r0, c1, c0, 0
    bic r0, r0, #0x00002000 @ clear bits 13 (--V-)
    bic r0, r0, #0x00000007 @ clear bits 2:0 (-CAM)
    orr r0, r0, #0x00000002 @ set bit 1 (--A-) Align
    orr r0, r0, #0x00000800 @ set bit 12 (Z---) BTB
    mcr p15, 0, r0, c1, c0, 0

    /*
     * Jump to board specific initialization...
     * The Mask ROM will have already initialized
     * basic memory. Go here to bump up clock rate and handle
     * wake up conditions.
     */
    mov ip, lr          @ persevere link reg across call
    bl  lowlevel_init       @ go setup pll,mux,memory
    mov lr, ip          @ restore link
    mov pc, lr          @ back to my caller
/*
 *************************************************************************
```

uboot_tiny4412-master/board/samsung/tiny4412

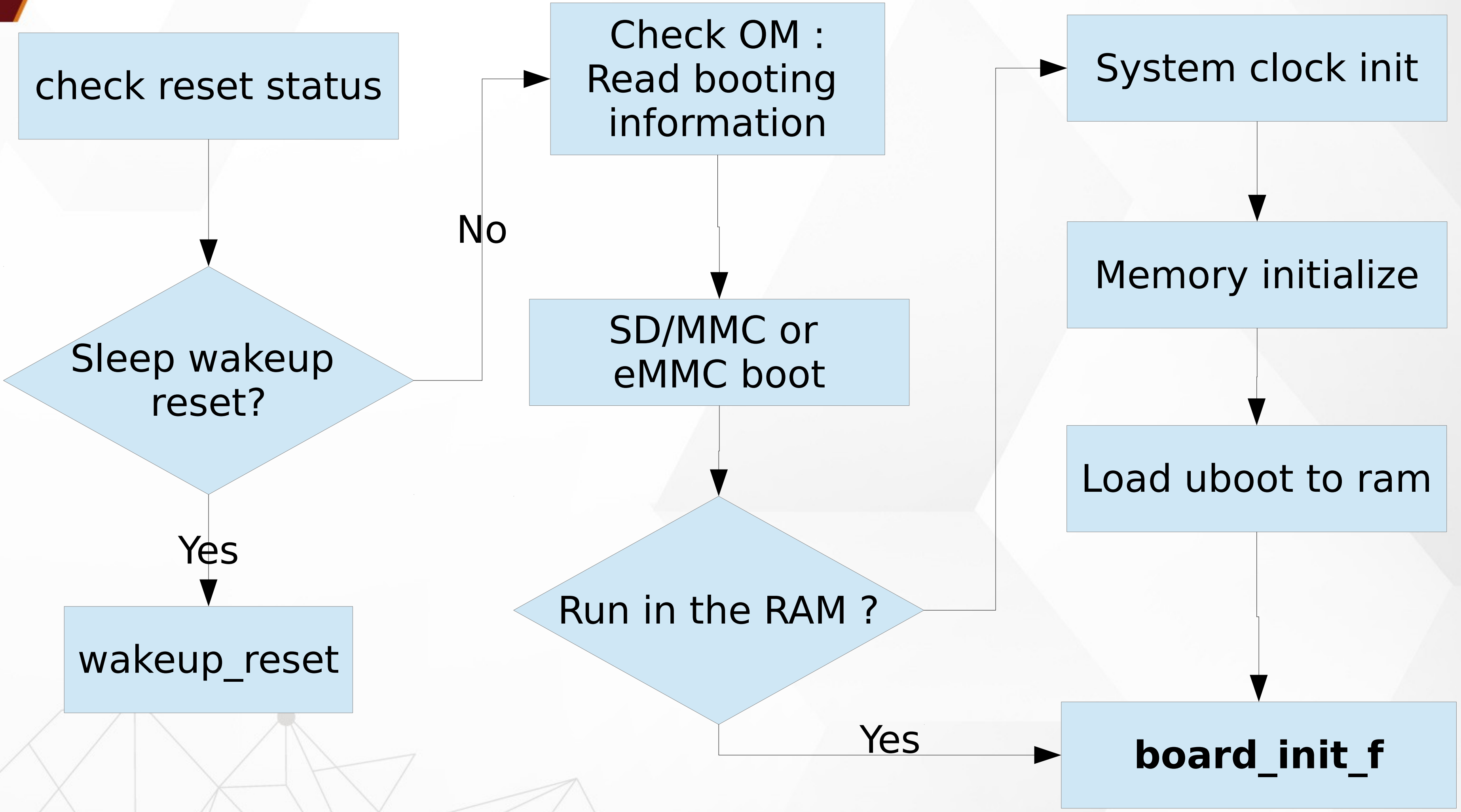
```
.
├── clock_init_tiny4412.S
├── config.mk
├── lowlevel_init.S
├── Makefile
├── mem_init_tiny4412_r1.S
├── mem_init_tiny4412_r2.S
├── mem_init_tiny4412.S
├── pmic.c
├── tiny4212_val.h
├── tiny4412.c
├── tiny4412_val.h
└── u-boot.lds
```

lowlevel_init



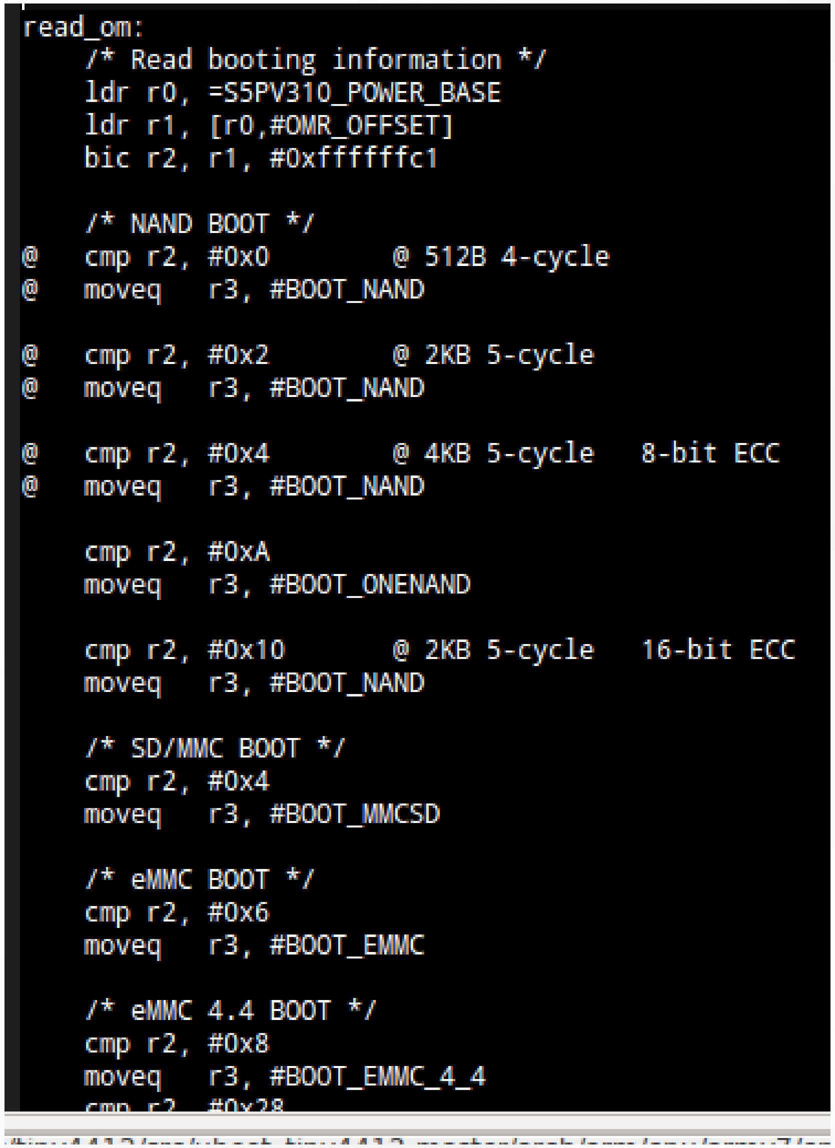
```
	.globl lowlevel_init
lowlevel_init:

	/* use iROM stack in bl2 */
	ldr sp, =0x02060000
	push	{lr}

	/* check reset status */
	ldr r0, =(INF_REG_BASE + INF_REG1_OFFSET)
	ldr r1, [r0]

	/* Sleep wakeup reset */
	ldr r2, =S5P_CHECK_SLEEP
	cmp r1, r2
	beq wakeup_reset

	/* set CP reset to low */
	ldr r0, =0x11000C60
	ldr r1, [r0]
	ldr r2, =0xFFFFFF0F
	and r1, r1, r2
	orr r1, r1, #0x10
	str r1, [r0]
	ldr r0, =0x11000C68
	ldr r1, [r0]
	ldr r2, =0xFFFFFFF3
	and r1, r1, r2
	orr r1, r1, #0x4
	str r1, [r0]
	ldr r0, =0x11000C64
	ldr r1, [r0]
	ldr r2, =0xFFFFFFFD
```

lowlevel_init

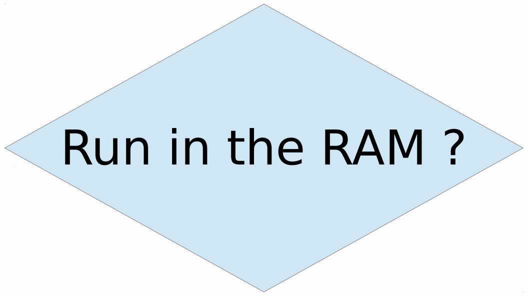


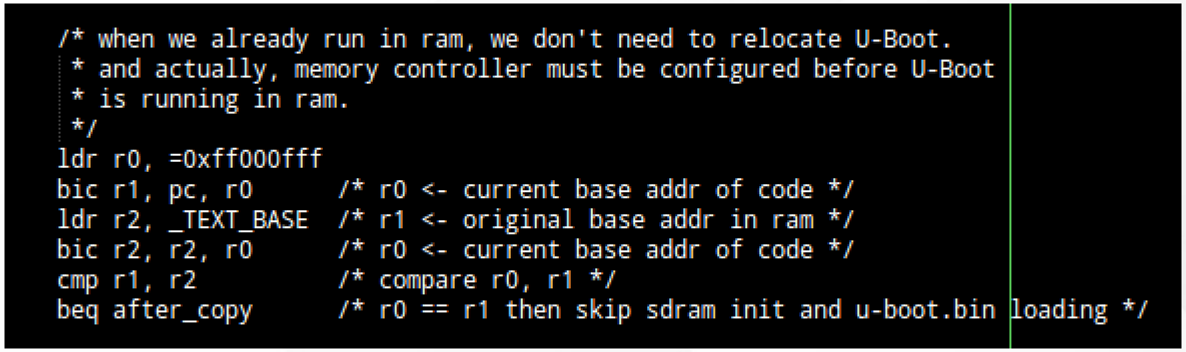
Check OM : Read booting information

SD/MMC or eMMC boot



```
read_om:
    /* Read booting information */
    ldr r0, =S5PV310_POWER_BASE
    ldr r1, [r0,#OMR_OFFSET]
    bic r2, r1, #0xffffffc1

    /* NAND BOOT */
@   cmp r2, #0x0        @ 512B 4-cycle
@   moveq   r3, #BOOT_NAND

@   cmp r2, #0x2        @ 2KB 5-cycle
@   moveq   r3, #BOOT_NAND

@   cmp r2, #0x4        @ 4KB 5-cycle   8-bit ECC
@   moveq   r3, #BOOT_NAND

    cmp r2, #0xA
    moveq   r3, #BOOT_ONENAND

    cmp r2, #0x10       @ 2KB 5-cycle   16-bit ECC
    moveq   r3, #BOOT_NAND

    /* SD/MMC BOOT */
    cmp r2, #0x4
    moveq   r3, #BOOT_MMCSD

    /* eMMC BOOT */
    cmp r2, #0x6
    moveq   r3, #BOOT_EMMC

    /* eMMC 4.4 BOOT */
    cmp r2, #0x8
    moveq   r3, #BOOT_EMMC_4_4
    cmp r2, #0x28
```

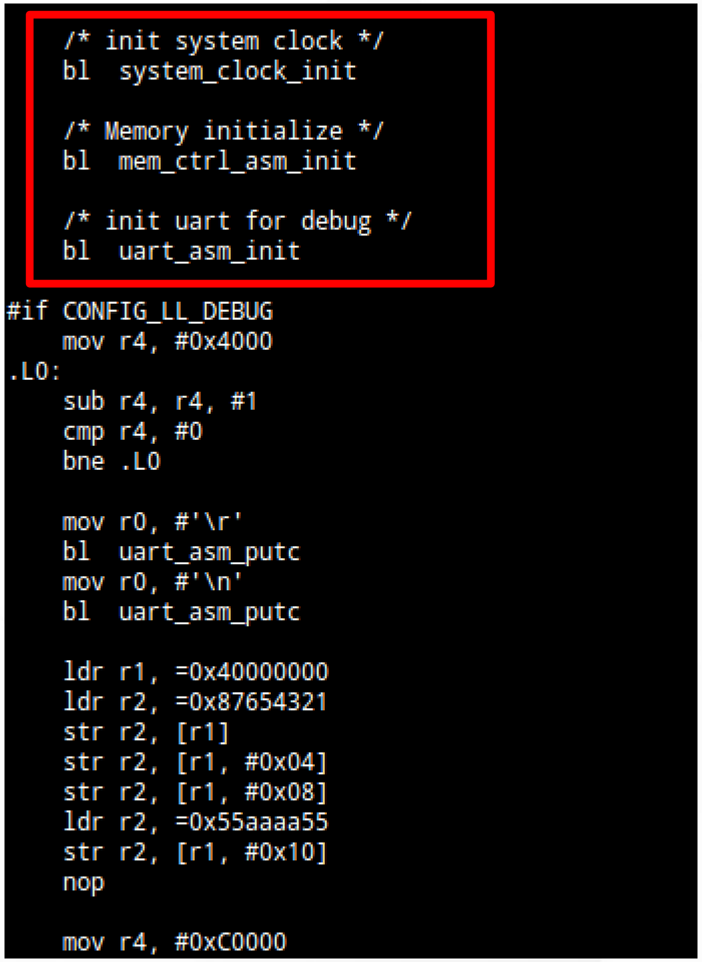




```
/* when we already run in ram, we don't need to relocate U-Boot.
 * and actually, memory controller must be configured before U-Boot
 * is running in ram.
 */
ldr r0, =0xff000fff
bic r1, pc, r0      /* r0 <- current base addr of code */
ldr r2, _TEXT_BASE  /* r1 <- original base addr in ram */
bic r2, r2, r0      /* r0 <- current base addr of code */
cmp r1, r2          /* compare r0, r1 */
beq after_copy      /* r0 == r1 then skip sdram init and u-boot.bin loading */
```

System clock init

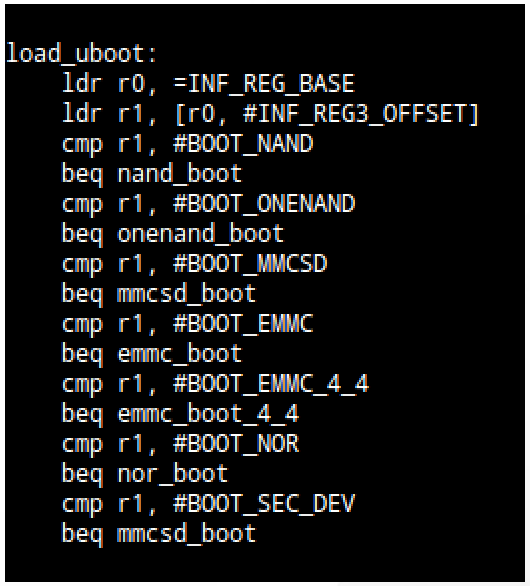
Memory initialize



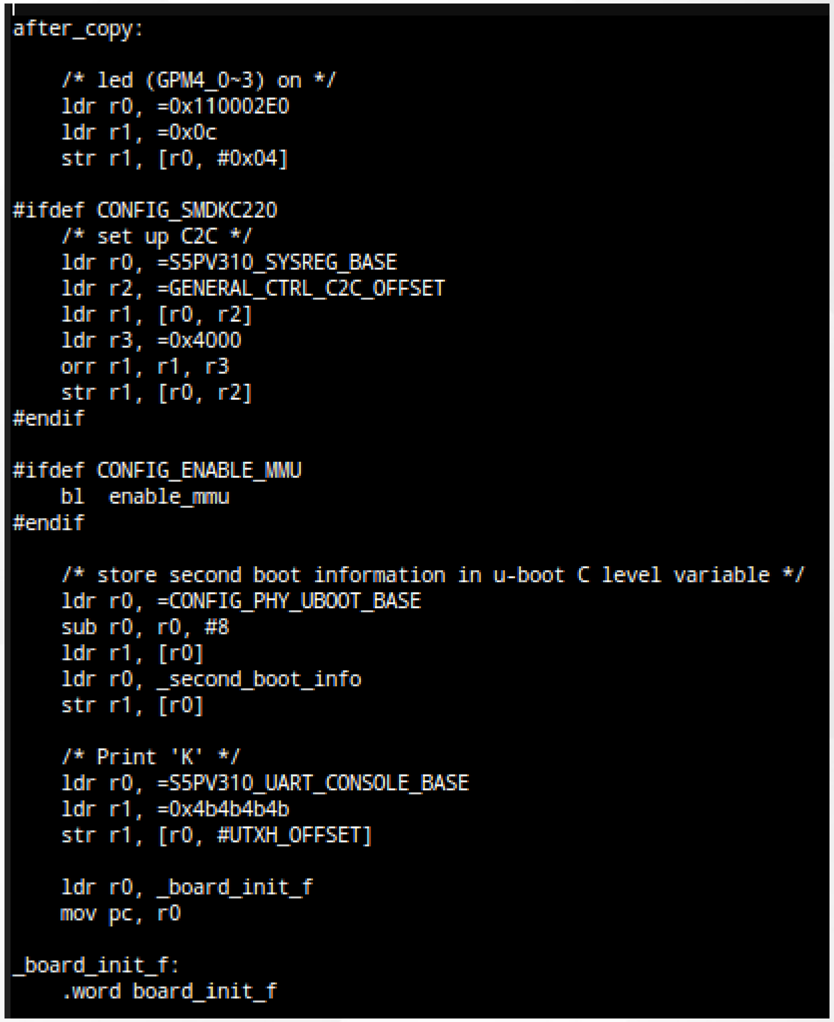
```
    /* init system clock */
    bl  system_clock_init

    /* Memory initialize */
    bl  mem_ctrl_asm_init

    /* init uart for debug */
    bl  uart_asm_init

#if CONFIG_LL_DEBUG
    mov r4, #0x4000
.L0:
    sub r4, r4, #1
    cmp r4, #0
    bne .L0

    mov r0, #'\r'
    bl  uart_asm_putc
    mov r0, #'\n'
    bl  uart_asm_putc

    ldr r1, =0x40000000
    ldr r2, =0x87654321
    str r2, [r1]
    str r2, [r1, #0x04]
    str r2, [r1, #0x08]
    ldr r2, =0x55aaaa55
    str r2, [r1, #0x10]
    nop

    mov r4, #0xC0000
```

Load uboot to ram

board_init_f

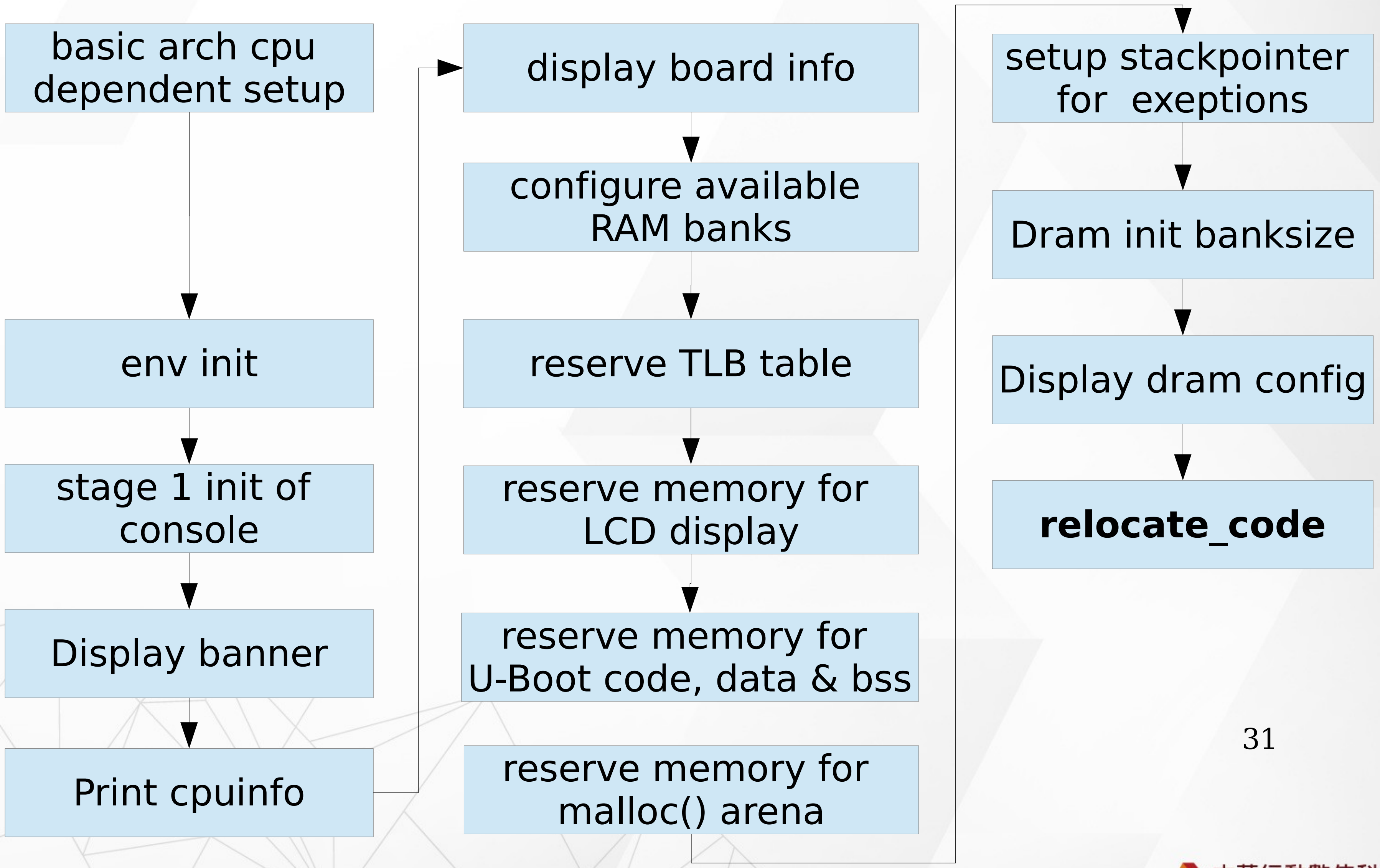


```
load_uboot:
    ldr r0, =INF_REG_BASE
    ldr r1, [r0, #INF_REG3_OFFSET]
    cmp r1, #BOOT_NAND
    beq nand_boot
    cmp r1, #BOOT_ONENAND
    beq onenand_boot
    cmp r1, #BOOT_MMCSD
    beq mmcsd_boot
    cmp r1, #BOOT_EMMC
    beq emmc_boot
    cmp r1, #BOOT_EMMC_4_4
    beq emmc_boot_4_4
    cmp r1, #BOOT_NOR
    beq nor_boot
    cmp r1, #BOOT_SEC_DEV
    beq mmcsd_boot
```



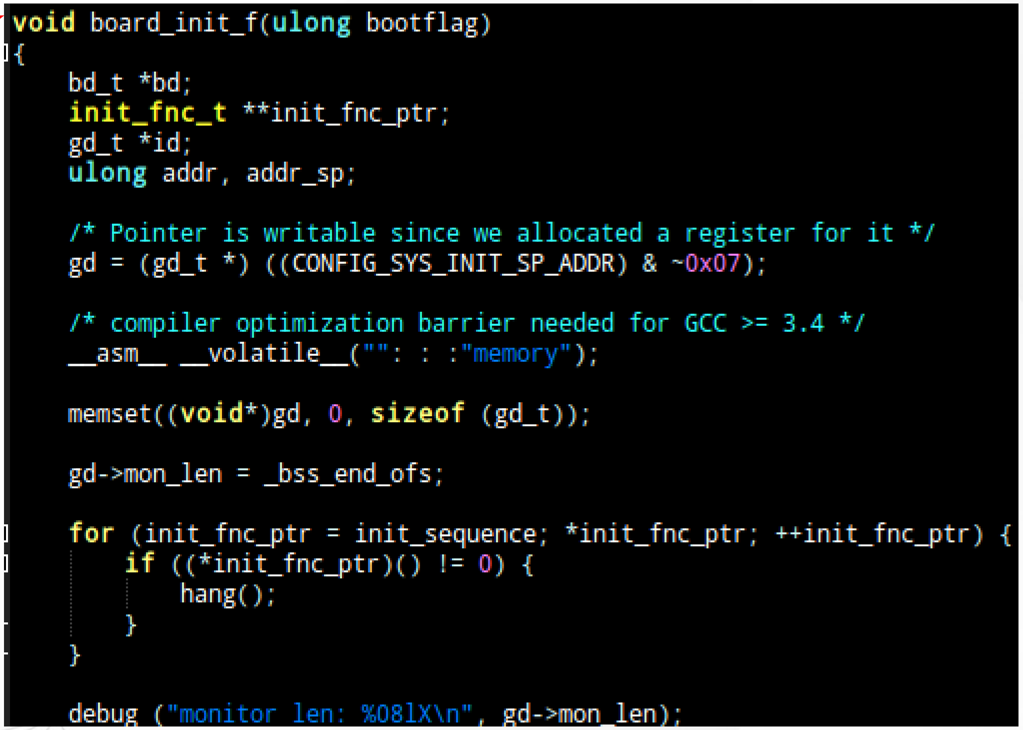
```
after_copy:

    /* led (GPM4_0~3) on */
    ldr r0, =0x110002E0
    ldr r1, =0x0c
    str r1, [r0, #0x04]

#ifdef CONFIG_SMDKC220
    /* set up C2C */
    ldr r0, =S5PV310_SYSREG_BASE
    ldr r2, =GENERAL_CTRL_C2C_OFFSET
    ldr r1, [r0, r2]
    ldr r3, =0x4000
    orr r1, r1, r3
    str r1, [r0, r2]
#endif

#ifdef CONFIG_ENABLE_MMU
    bl  enable_mmu
#endif

    /* store second boot information in u-boot C level variable */
    ldr r0, =CONFIG_PHY_UBOOT_BASE
    sub r0, r0, #8
    ldr r1, [r0]
    ldr r0, _second_boot_info
    str r1, [r0]

    /* Print 'K' */
    ldr r0, =S5PV310_UART_CONSOLE_BASE
    ldr r1, =0x4b4b4b4b
    str r1, [r0, #UTXH_OFFSET]

    ldr r0, _board_init_f
    mov pc, r0

_board_init_f:
    .word board_init_f
```

board_init_f

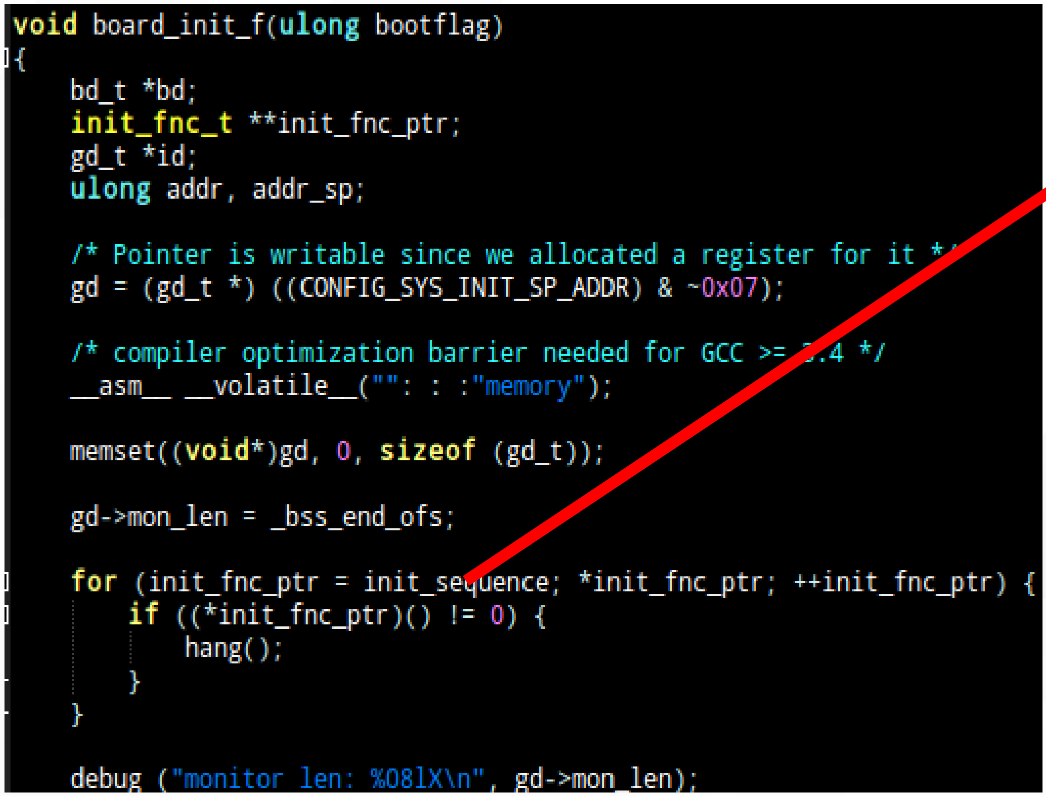

basic arch cpu dependent setup
env init
stage 1 init of console
Display banner
Print cpuinfo
display board info
configure available RAM banks
reserve TLB table
reserve memory for LCD display
reserve memory for U-Boot code, data & bss
reserve memory for malloc() arena
setup stackpointer for exeptions
Dram init banksize
Display dram config
relocate_code

uboot_tiny4412-master/arch/arm/lib

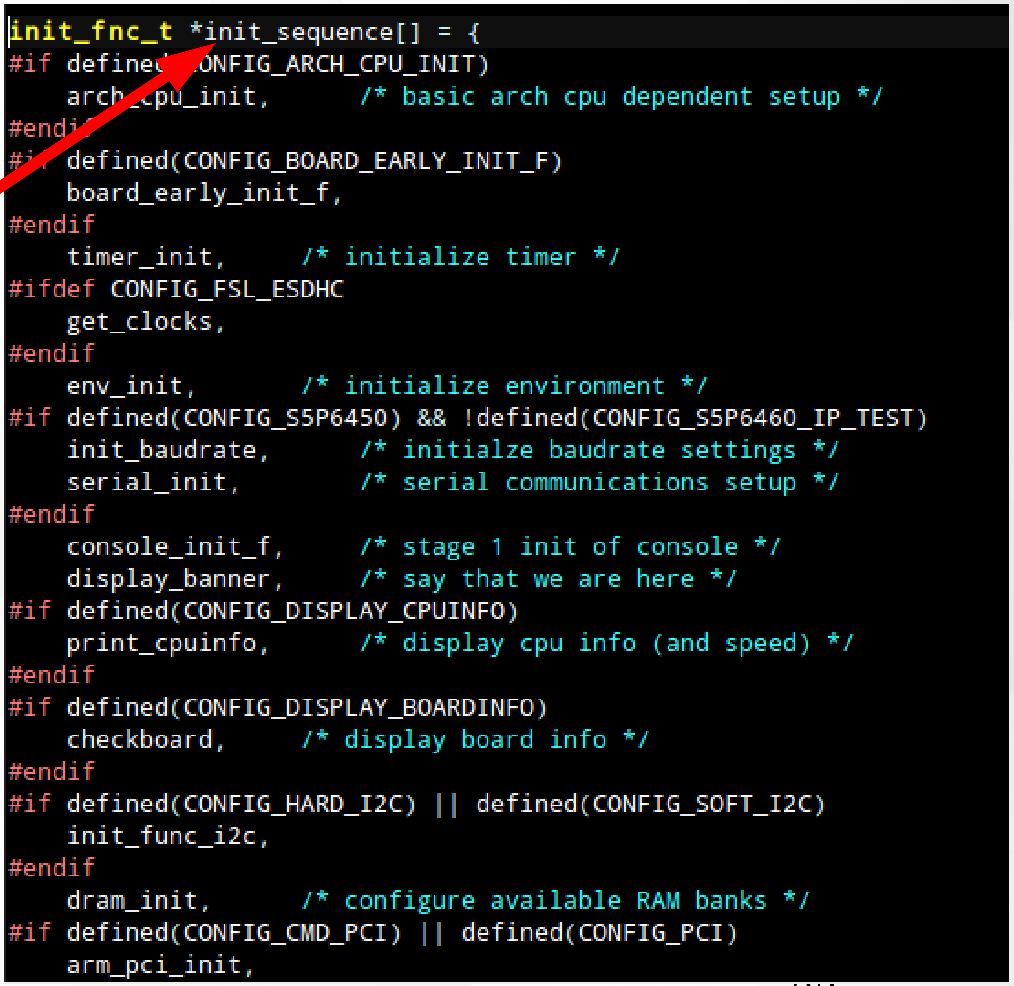
```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```



```
void board_init_f(ulong bootflag)
{
    bd_t *bd;
    init_fnc_t **init_fnc_ptr;
    gd_t *id;
    ulong addr, addr_sp;

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) ((CONFIG_SYS_INIT_SP_ADDR) & ~0x07);

    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("": : :"memory");

    memset((void*)gd, 0, sizeof (gd_t));

    gd->mon_len = _bss_end_ofs;

    for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {
        if ((*init_fnc_ptr)() != 0) {
            hang();
        }
    }

    debug ("monitor len: %08lX\n", gd->mon_len);
```



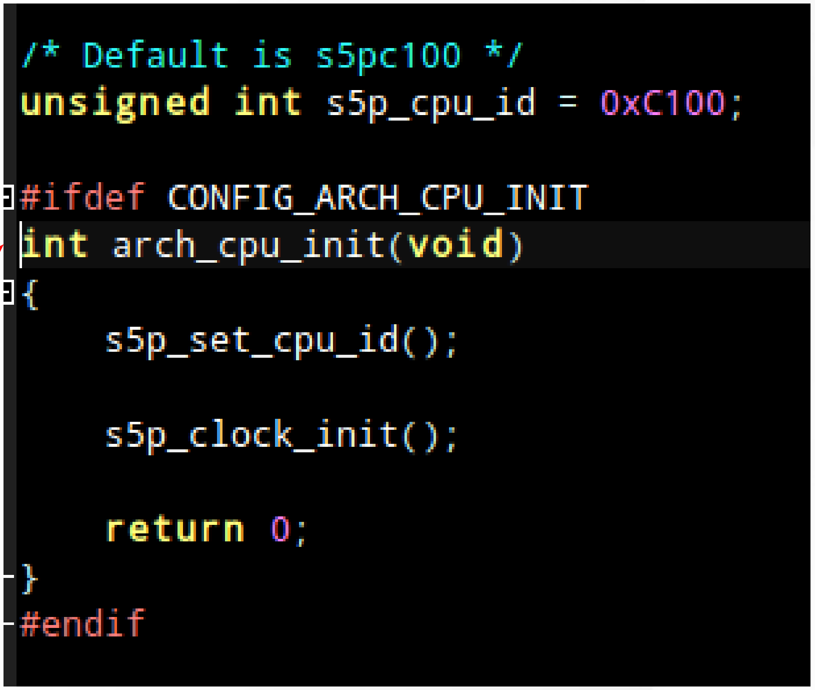
```c
void board_init_f(ulong bootflag)
{
    bd_t *bd;
    init_fnc_t **init_fnc_ptr;
    gd_t *id;
    ulong addr, addr_sp;

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) ((CONFIG_SYS_INIT_SP_ADDR) & ~0x07);

    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("": : :"memory");

    memset((void*)gd, 0, sizeof (gd_t));

    gd->mon_len = _bss_end_ofs;

    for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {
        if ((*init_fnc_ptr)() != 0) {
            hang();
        }
    }

    debug ("monitor len: %08lX\n", gd->mon_len);
```



```c
init_fnc_t *init_sequence[] = {
#if defined(CONFIG_ARCH_CPU_INIT)
    arch_cpu_init,      /* basic arch cpu dependent setup */
#endif
#if defined(CONFIG_BOARD_EARLY_INIT_F)
    board_early_init_f,
#endif
    timer_init,     /* initialize timer */
#ifdef CONFIG_FSL_ESDHC
    get_clocks,
#endif
    env_init,       /* initialize environment */
#if defined(CONFIG_S5P6450) && !defined(CONFIG_S5P6460_IP_TEST)
    init_baudrate,      /* initialze baudrate settings */
    serial_init,        /* serial communications setup */
#endif
    console_init_f,     /* stage 1 init of console */
    display_banner,     /* say that we are here */
#if defined(CONFIG_DISPLAY_CPUINFO)
    print_cpuinfo,      /* display cpu info (and speed) */
#endif
#if defined(CONFIG_DISPLAY_BOARDINFO)
    checkboard,     /* display board info */
#endif
#if defined(CONFIG_HARD_I2C) || defined(CONFIG_SOFT_I2C)
    init_func_i2c,
#endif
    dram_init,      /* configure available RAM banks */
#if defined(CONFIG_CMD_PCI) || defined(CONFIG_PCI)
    arm_pci_init,
```

basic arch cpu
dependent setup

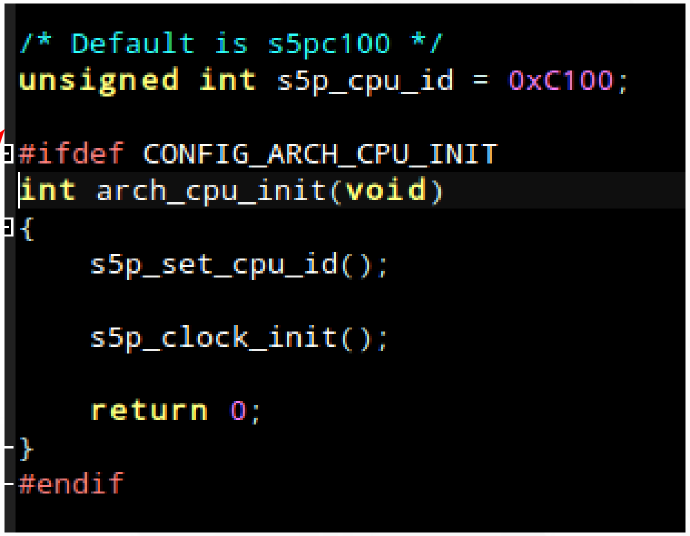
```
uboot_tiny4412-master/arch/arm/cpu/armv7/exynos
.
├── ace_sha1.c
├── clock.c
├── gpio.c
├── i2c.c
├── irom_copy.c
├── Makefile
├── movi_partition.c
├── nand.c
├── nand_cp.c
├── nand_write_bl.c
├── onenand_cp.c
├── pmic.c
├── reset.c
├── security_check.c
├── setup_hsmmc.c
├── sys_info.c
├── UBOOT_SB20_S5PC210S.c
└── UBOOT_SB20_S5PC210S.h
```



```
/* Default is s5pc100 */
unsigned int s5p_cpu_id = 0xC100;

#ifdef CONFIG_ARCH_CPU_INIT
int arch_cpu_init(void)
{
	s5p_set_cpu_id();

	s5p_clock_init();

	return 0;
}
#endif
```

uboot_tiny4412-master/arch/arm/cpu/armv7/exynos

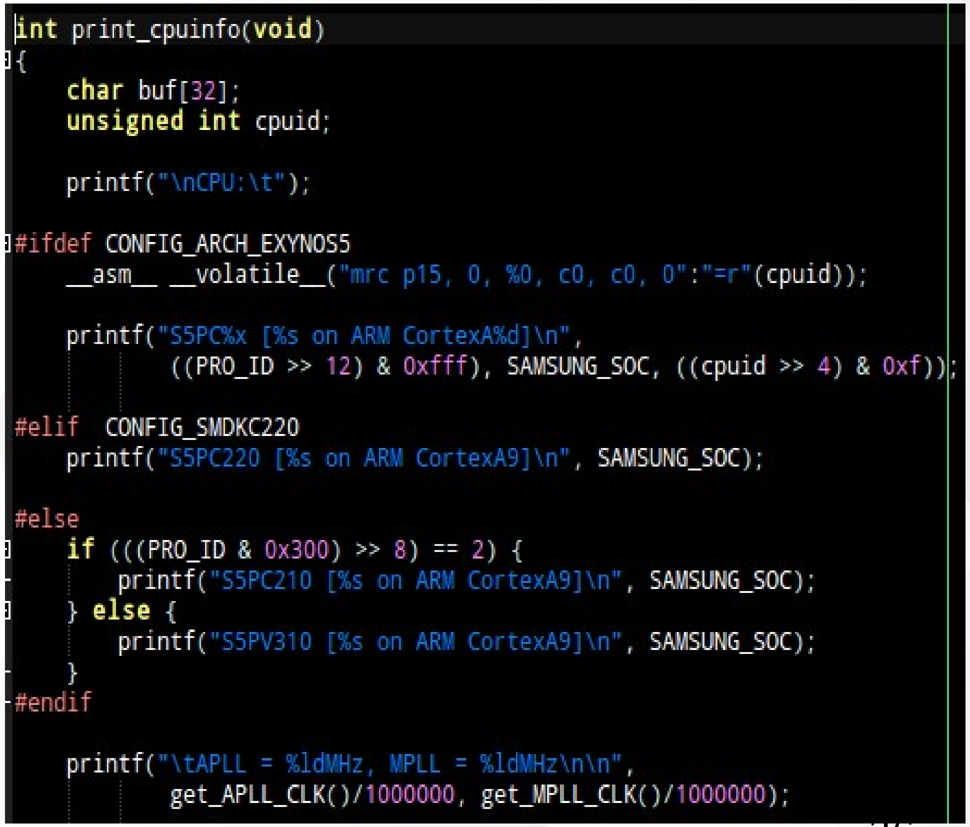
```
.
├── ace_sha1.c
├── clock.c
├── gpio.c
├── i2c.c
├── irom_copy.c
├── Makefile
├── movi_partition.c
├── nand.c
├── nand_cp.c
├── nand_write_bl.c
├── onenand_cp.c
├── pmic.c
├── reset.c
├── security_check.c
├── setup_hsmmc.c
├── sys_info.c
├── UBOOT_SB20_S5PC210S.c
└── UBOOT_SB20_S5PC210S.h
```



```
/* Default is s5pc100 */
unsigned int s5p_cpu_id = 0xC100;

#ifdef CONFIG_ARCH_CPU_INIT
int arch_cpu_init(void)
{
	s5p_set_cpu_id();

	s5p_clock_init();

	return 0;
}
#endif
```

uboot_tiny4412-master/arch/arm/cpu/armv7/exynos

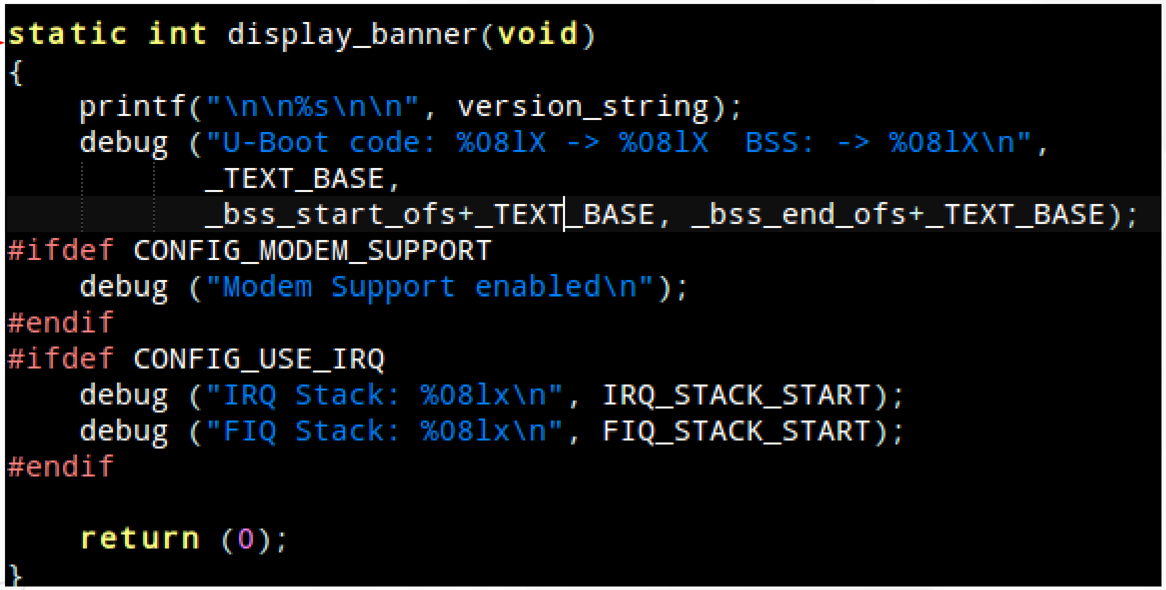
```
.
├── ace_sha1.c
├── clock.c
├── gpio.c
├── i2c.c
├── irom_copy.c
├── Makefile
├── movi_partition.c
├── nand.c
├── nand_cp.c
├── nand_write_bl.c
├── onenand_cp.c
├── pmic.c
├── reset.c
├── security_check.c
├── setup_hsmmc.c
├── sys_info.c
├── UBOOT_SB20_S5PC210S.c
└── UBOOT_SB20_S5PC210S.h
```



```
int print_cpuinfo(void)
{
    char buf[32];
    unsigned int cpuid;

    printf("\nCPU:\t");

#ifdef CONFIG_ARCH_EXYNOS5
    __asm__ __volatile__("mrc p15, 0, %0, c0, c0, 0":"=r"(cpuid));

    printf("S5PC%x [%s on ARM CortexA%d]\n",
            ((PRO_ID >> 12) & 0xfff), SAMSUNG_SOC, ((cpuid >> 4) & 0xf));

#elif  CONFIG_SMDKC220
    printf("S5PC220 [%s on ARM CortexA9]\n", SAMSUNG_SOC);

#else
    if (((PRO_ID & 0x300) >> 8) == 2) {
        printf("S5PC210 [%s on ARM CortexA9]\n", SAMSUNG_SOC);
    } else {
        printf("S5PV310 [%s on ARM CortexA9]\n", SAMSUNG_SOC);
    }
#endif

    printf("\tAPLL = %ldMHz, MPLL = %ldMHz\n\n",
            get_APLL_CLK()/1000000, get_MPLL_CLK()/1000000);
```

uboot_tiny4412-master/arch/arm/lib

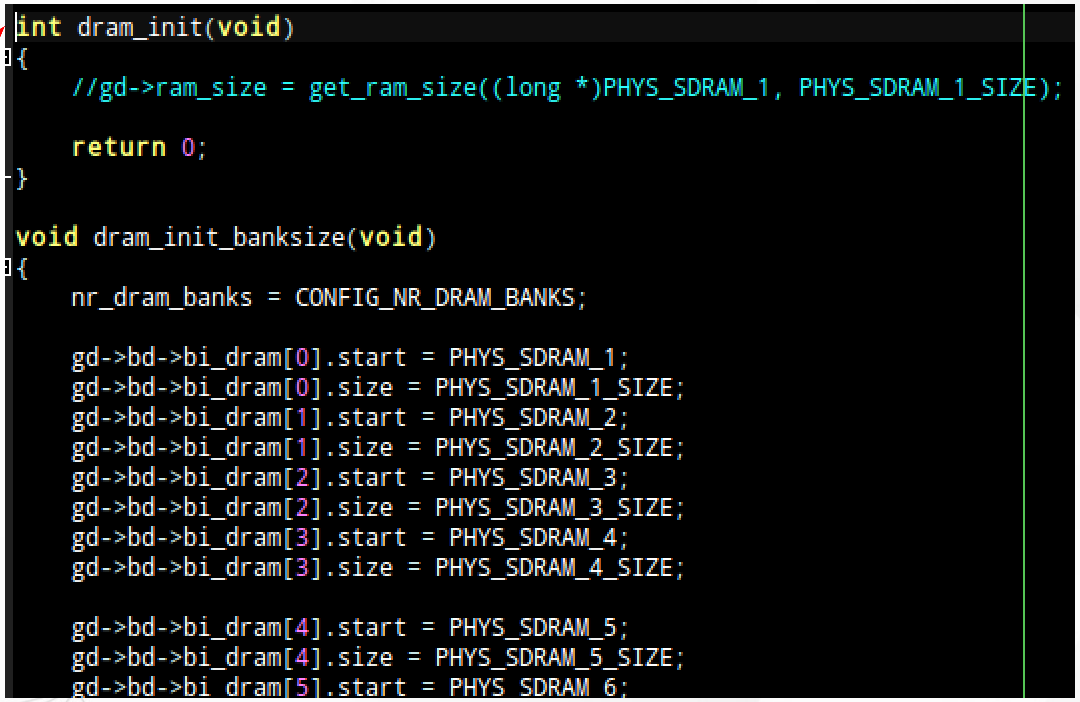
```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```



```
static int display_banner(void)
{
    printf("\n\n%s\n\n", version_string);
    debug ("U-Boot code: %08lX -> %08lX  BSS: -> %08lX\n",
           _TEXT_BASE,
           _bss_start_ofs+_TEXT_BASE, _bss_end_ofs+_TEXT_BASE);
#ifdef CONFIG_MODEM_SUPPORT
    debug ("Modem Support enabled\n");
#endif
#ifdef CONFIG_USE_IRQ
    debug ("IRQ Stack: %08lx\n", IRQ_STACK_START);
    debug ("FIQ Stack: %08lx\n", FIQ_STACK_START);
#endif

    return (0);
}
```

uboot_tiny4412-master/board/samsung/tiny4412

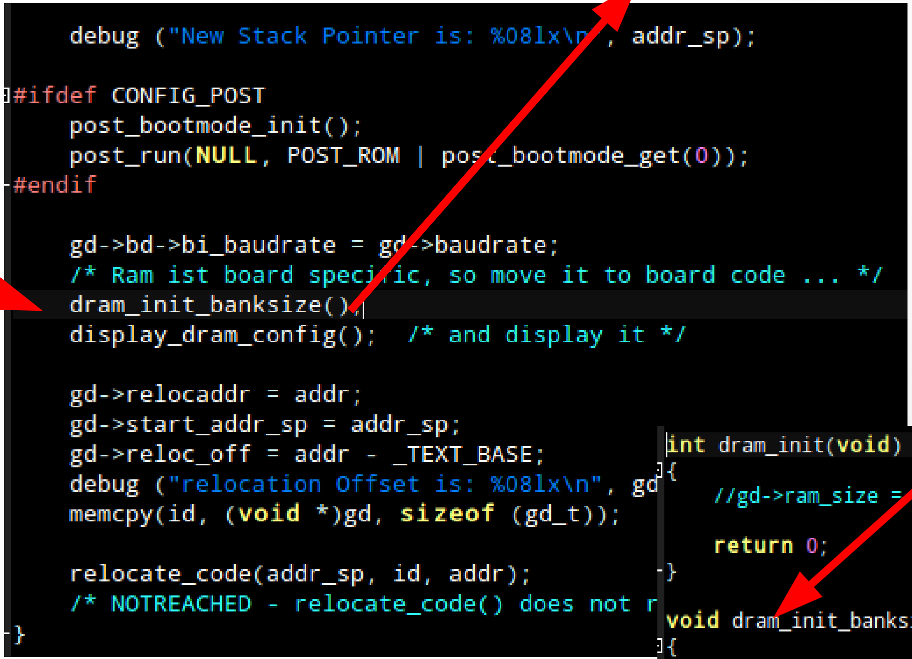
```
.
├── clock_init_tiny4412.S
├── config.mk
├── lowlevel_init.S
├── Makefile
├── mem_init_tiny4412_r1.S
├── mem_init_tiny4412_r2.S
├── mem_init_tiny4412.S
├── pmic.c
├── tiny4212_val.h
├── tiny4412.c
├── tiny4412_val.h
└── u-boot.lds
```



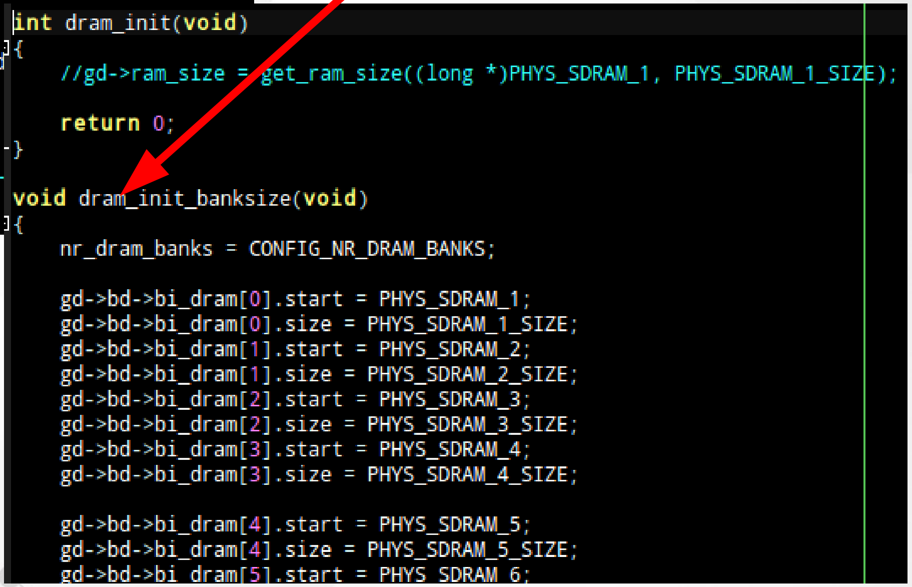
```
int dram_init(void)
{
    //gd->ram_size = get_ram_size((long *)PHYS_SDRAM_1, PHYS_SDRAM_1_SIZE);

    return 0;
}

void dram_init_banksize(void)
{
    nr_dram_banks = CONFIG_NR_DRAM_BANKS;

    gd->bd->bi_dram[0].start = PHYS_SDRAM_1;
    gd->bd->bi_dram[0].size = PHYS_SDRAM_1_SIZE;
    gd->bd->bi_dram[1].start = PHYS_SDRAM_2;
    gd->bd->bi_dram[1].size = PHYS_SDRAM_2_SIZE;
    gd->bd->bi_dram[2].start = PHYS_SDRAM_3;
    gd->bd->bi_dram[2].size = PHYS_SDRAM_3_SIZE;
    gd->bd->bi_dram[3].start = PHYS_SDRAM_4;
    gd->bd->bi_dram[3].size = PHYS_SDRAM_4_SIZE;

    gd->bd->bi_dram[4].start = PHYS_SDRAM_5;
    gd->bd->bi_dram[4].size = PHYS_SDRAM_5_SIZE;
    gd->bd->bi_dram[5].start = PHYS_SDRAM_6;
```

uboot_tiny4412-master/arch/arm/lib   master/board/samsung/tiny4412/tiny4412.c

```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```



```
	debug ("New Stack Pointer is: %08lx\n", addr_sp);

#ifdef CONFIG_POST
	post_bootmode_init();
	post_run(NULL, POST_ROM | post_bootmode_get(0));
#endif

	gd->bd->bi_baudrate = gd->baudrate;
	/* Ram ist board specific, so move it to board code ... */
	dram_init_banksize();
	display_dram_config();	/* and display it */

	gd->relocaddr = addr;
	gd->start_addr_sp = addr_sp;
	gd->reloc_off = addr - _TEXT_BASE;
	debug ("relocation Offset is: %08lx\n", gd
	memcpy(id, (void *)gd, sizeof (gd_t));

	relocate_code(addr_sp, id, addr);
	/* NOTREACHED - relocate_code() does not r
}
```

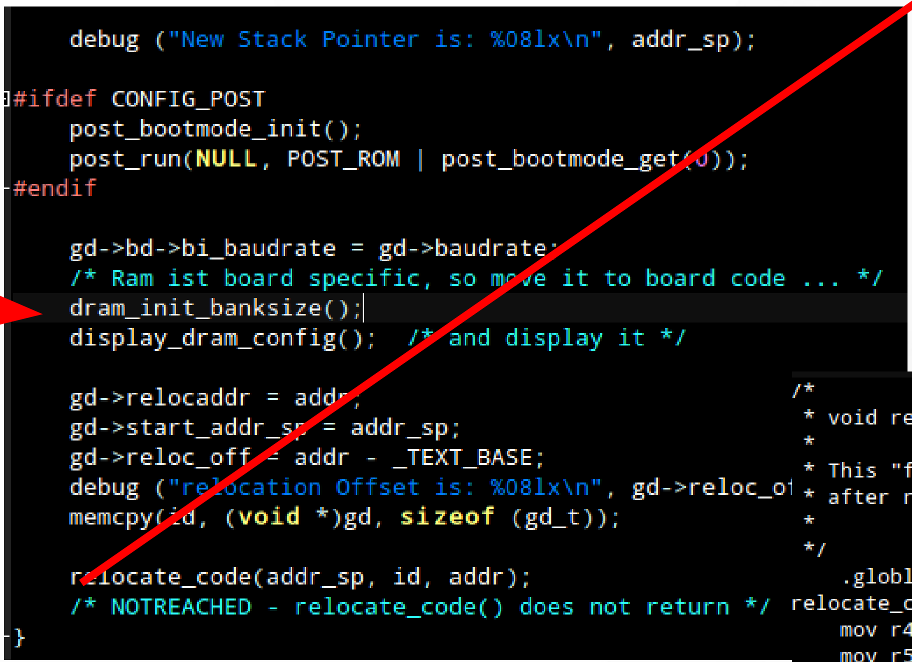


```
int dram_init(void)
{
	//gd->ram_size = get_ram_size((long *)PHYS_SDRAM_1, PHYS_SDRAM_1_SIZE);

	return 0;
}

void dram_init_banksize(void)
{
	nr_dram_banks = CONFIG_NR_DRAM_BANKS;

	gd->bd->bi_dram[0].start = PHYS_SDRAM_1;
	gd->bd->bi_dram[0].size = PHYS_SDRAM_1_SIZE;
	gd->bd->bi_dram[1].start = PHYS_SDRAM_2;
	gd->bd->bi_dram[1].size = PHYS_SDRAM_2_SIZE;
	gd->bd->bi_dram[2].start = PHYS_SDRAM_3;
	gd->bd->bi_dram[2].size = PHYS_SDRAM_3_SIZE;
	gd->bd->bi_dram[3].start = PHYS_SDRAM_4;
	gd->bd->bi_dram[3].size = PHYS_SDRAM_4_SIZE;

	gd->bd->bi_dram[4].start = PHYS_SDRAM_5;
	gd->bd->bi_dram[4].size = PHYS_SDRAM_5_SIZE;
	gd->bd->bi_dram[5].start = PHYS_SDRAM_6;
```

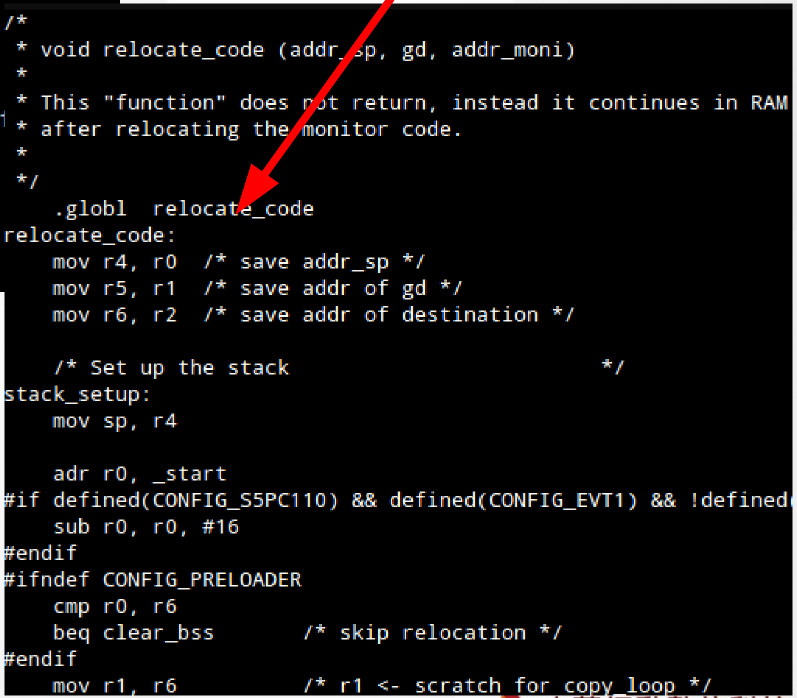
uboot_tiny4412-master/arch/arm/lib

arch/arm/cpu/armv7/start.S

```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```

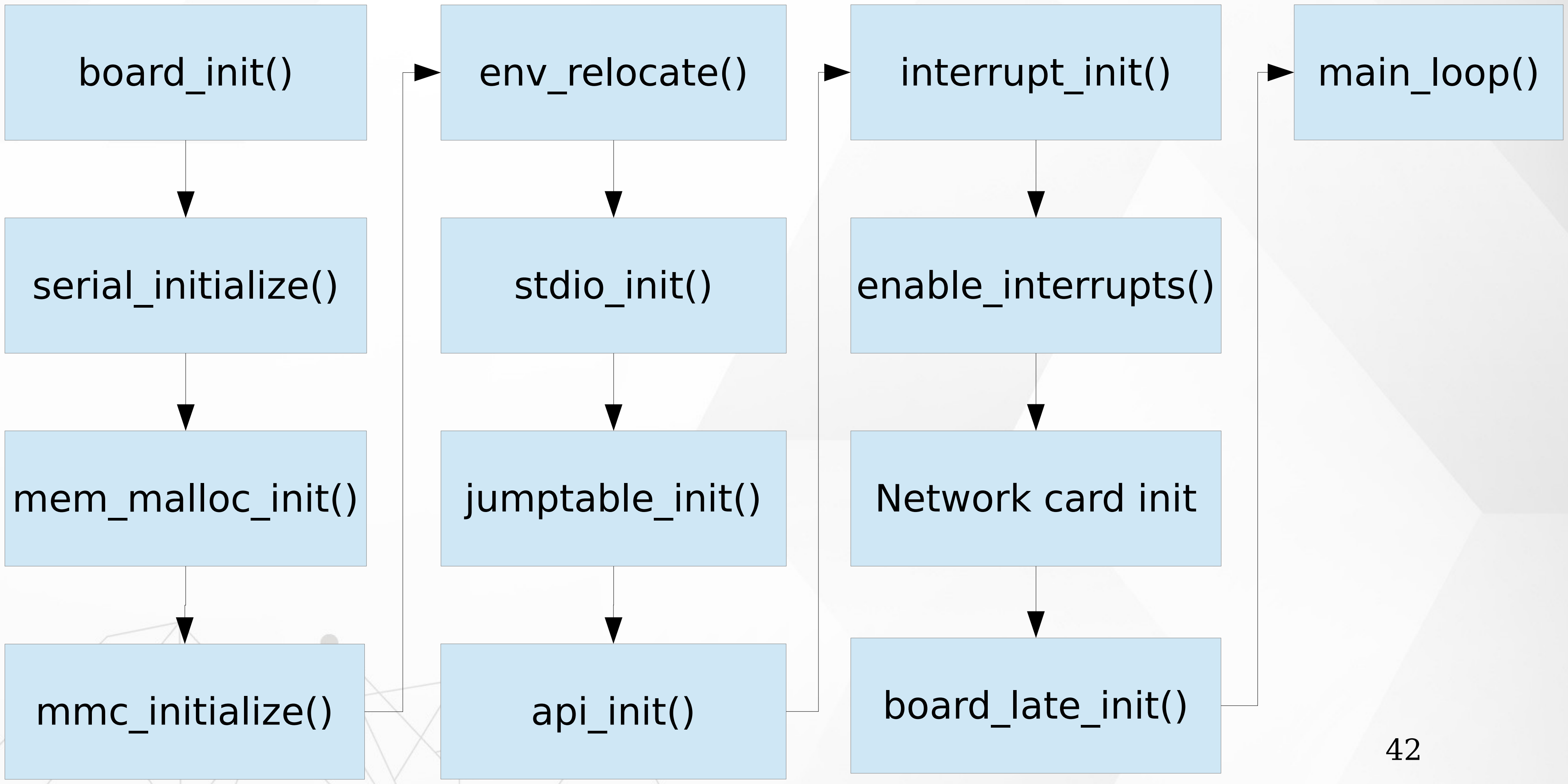


```
	debug ("New Stack Pointer is: %08lx\n", addr_sp);

#ifdef CONFIG_POST
	post_bootmode_init();
	post_run(NULL, POST_ROM | post_bootmode_get(0));
#endif

	gd->bd->bi_baudrate = gd->baudrate;
	/* Ram ist board specific, so move it to board code ... */
	dram_init_banksize();
	display_dram_config();	/* and display it */

	gd->relocaddr = addr;
	gd->start_addr_sp = addr_sp;
	gd->reloc_off = addr - _TEXT_BASE;
	debug ("relocation Offset is: %08lx\n", gd->reloc_of
	memcpy(id, (void *)gd, sizeof (gd_t));

	relocate_code(addr_sp, id, addr);
	/* NOTREACHED - relocate_code() does not return */
}
```



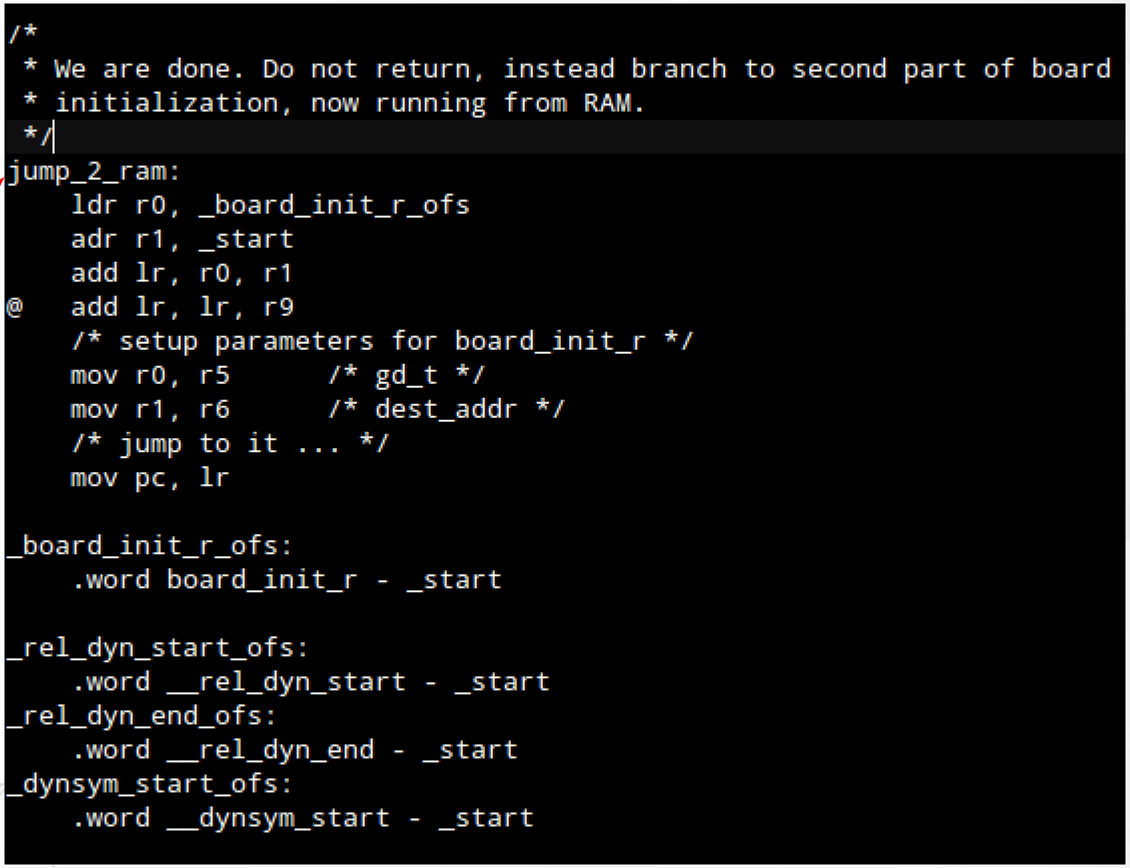
```
/*
 * void relocate_code (addr_sp, gd, addr_moni)
 *
 * This "function" does not return, instead it continues in RAM
 * after relocating the monitor code.
 *
 */
	.globl	relocate_code
relocate_code:
	mov	r4, r0	/* save addr_sp */
	mov	r5, r1	/* save addr of gd */
	mov	r6, r2	/* save addr of destination */

	/* Set up the stack						    */
stack_setup:
	mov	sp, r4

	adr	r0, _start
#if defined(CONFIG_S5PC110) && defined(CONFIG_EVT1) && !defined
	sub	r0, r0, #16
#endif
#ifndef CONFIG_PRELOADER
	cmp	r0, r6
	beq	clear_bss		/* skip relocation */
#endif
	mov	r1, r6			/* r1 <- scratch for copy_loop */
```

board_init_r


board_init()
serial_initialize()
mem_malloc_init()
mmc_initialize()
env_relocate()
stdio_init()
jumptable_init()
api_init()
interrupt_init()
enable_interrupts()
Network card init
board_late_init()
main_loop()

jump to ram
(board_init_r)

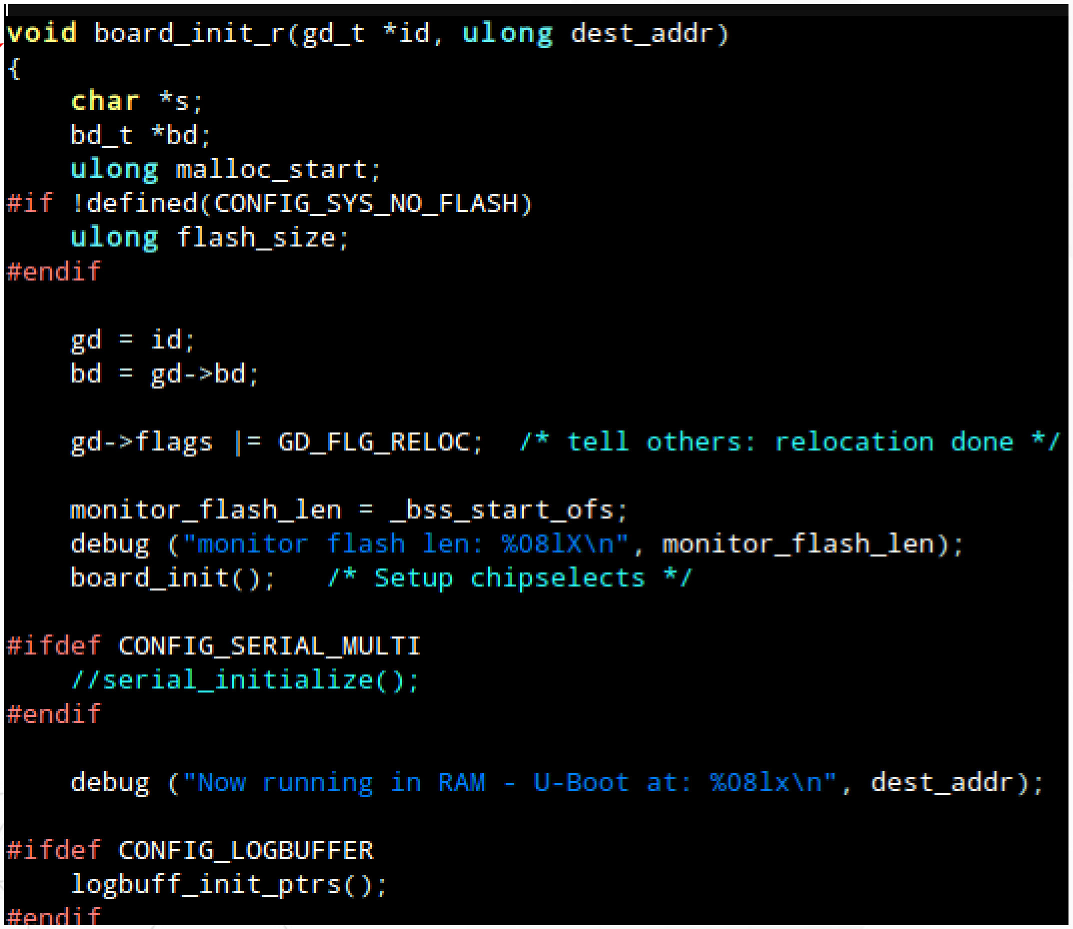
```
uboot_tiny4412-master/arch/arm/cpu/armv7
├── config.mk
├── cpu.c
├── exynos
│   ├── ace_sha1.c
│   ├── clock.c
│   ├── gpio.c
│   ├── i2c.c
│   ├── irom_copy.c
~~~~~~~~~~~~~~~~~~~~~~~~
│   ├── Makefile
│   ├── movi_partition.c
│   ├── nand.c
│   ├── nand_cp.c
│   └── UBOOT_SB20_S5PC210S.h
├── start.S
├── syslib.c
└── u-boot.lds
```



```
/*
 * We are done. Do not return, instead branch to second part of board
 * initialization, now running from RAM.
 */
jump_2_ram:
    ldr r0, _board_init_r_ofs
    adr r1, _start
    add lr, r0, r1
@   add lr, lr, r9
    /* setup parameters for board_init_r */
    mov r0, r5      /* gd_t */
    mov r1, r6      /* dest_addr */
    /* jump to it ... */
    mov pc, lr

_board_init_r_ofs:
    .word board_init_r - _start

_rel_dyn_start_ofs:
    .word __rel_dyn_start - _start
_rel_dyn_end_ofs:
    .word __rel_dyn_end - _start
_dynsym_start_ofs:
    .word __dynsym_start - _start
```

uboot_tiny4412-master/arch/arm/lib

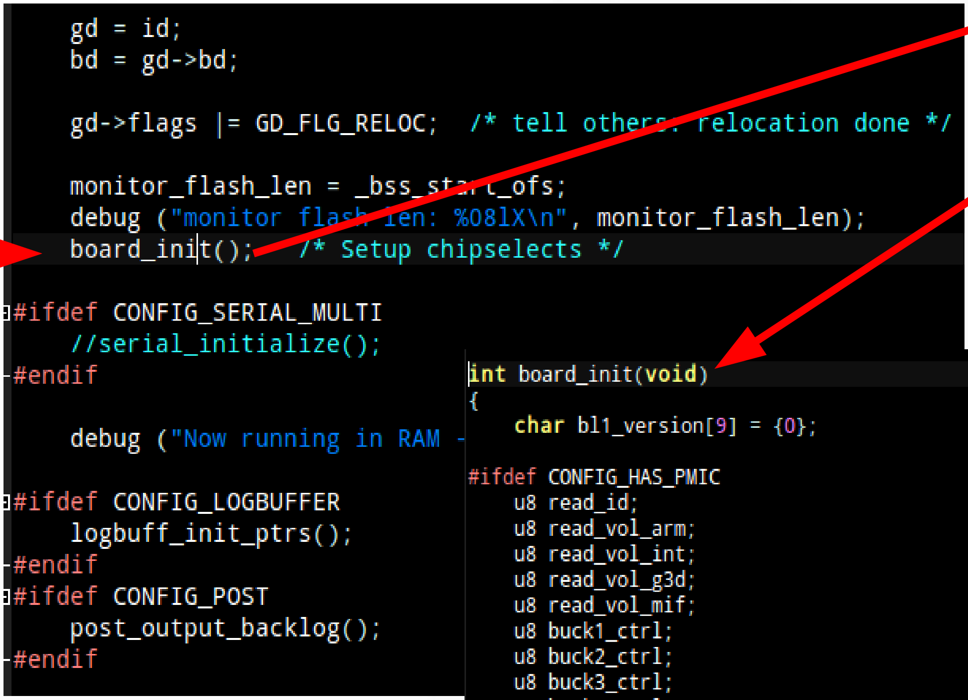
```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```



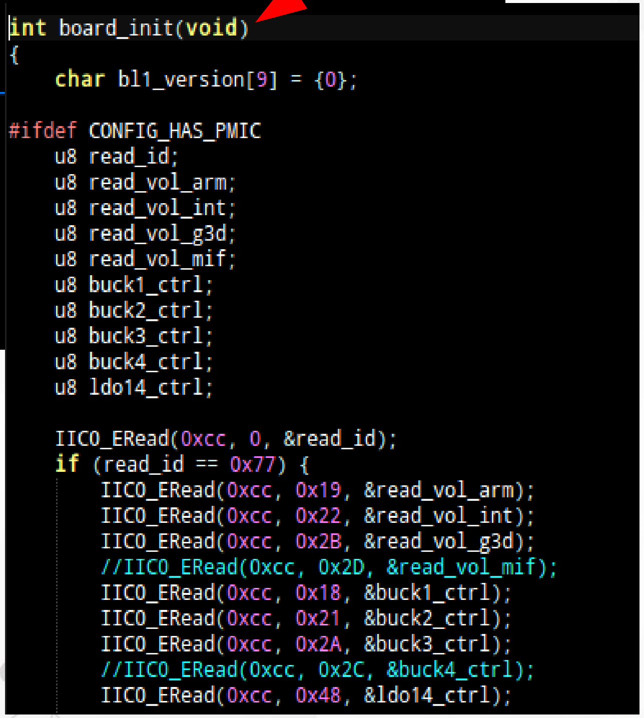
```
void board_init_r(gd_t *id, ulong dest_addr)
{
    char *s;
    bd_t *bd;
    ulong malloc_start;
#if !defined(CONFIG_SYS_NO_FLASH)
    ulong flash_size;
#endif

    gd = id;
    bd = gd->bd;

    gd->flags |= GD_FLG_RELOC;  /* tell others: relocation done */

    monitor_flash_len = _bss_start_ofs;
    debug ("monitor flash len: %08lX\n", monitor_flash_len);
    board_init();   /* Setup chipselects */

#ifdef CONFIG_SERIAL_MULTI
    //serial_initialize();
#endif

    debug ("Now running in RAM - U-Boot at: %08lx\n", dest_addr);

#ifdef CONFIG_LOGBUFFER
    logbuff_init_ptrs();
#endif
```

uboot_tiny4412-master/arch/arm/lib   master/board/samsung/tiny4412/tiny4412.c

```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```



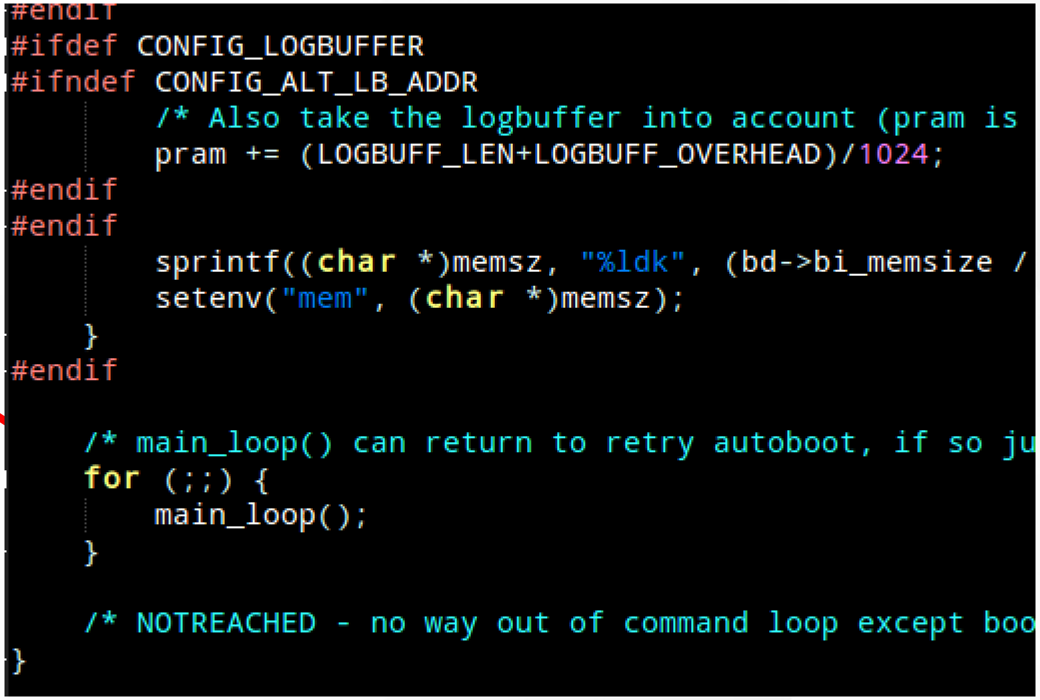
```
	gd = id;
	bd = gd->bd;

	gd->flags |= GD_FLG_RELOC;	/* tell others: relocation done */

	monitor_flash_len = _bss_start_ofs;
	debug ("monitor flash len: %08lX\n", monitor_flash_len);
	board_init();	/* Setup chipselects */

#ifdef CONFIG_SERIAL_MULTI
	//serial_initialize();
#endif

	debug ("Now running in RAM -

#ifdef CONFIG_LOGBUFFER
	logbuff_init_ptrs();
#endif
#ifdef CONFIG_POST
	post_output_backlog();
#endif
```



```
int board_init(void)
{
	char bl1_version[9] = {0};

#ifdef CONFIG_HAS_PMIC
	u8 read_id;
	u8 read_vol_arm;
	u8 read_vol_int;
	u8 read_vol_g3d;
	u8 read_vol_mif;
	u8 buck1_ctrl;
	u8 buck2_ctrl;
	u8 buck3_ctrl;
	u8 buck4_ctrl;
	u8 ldo14_ctrl;

	IIC0_ERead(0xcc, 0, &read_id);
	if (read_id == 0x77) {
		IIC0_ERead(0xcc, 0x19, &read_vol_arm);
		IIC0_ERead(0xcc, 0x22, &read_vol_int);
		IIC0_ERead(0xcc, 0x2B, &read_vol_g3d);
		//IIC0_ERead(0xcc, 0x2D, &read_vol_mif);
		IIC0_ERead(0xcc, 0x18, &buck1_ctrl);
		IIC0_ERead(0xcc, 0x21, &buck2_ctrl);
		IIC0_ERead(0xcc, 0x2A, &buck3_ctrl);
		//IIC0_ERead(0xcc, 0x2C, &buck4_ctrl);
		IIC0_ERead(0xcc, 0x48, &ldo14_ctrl);
```

uboot_tiny4412-master/arch/arm/lib

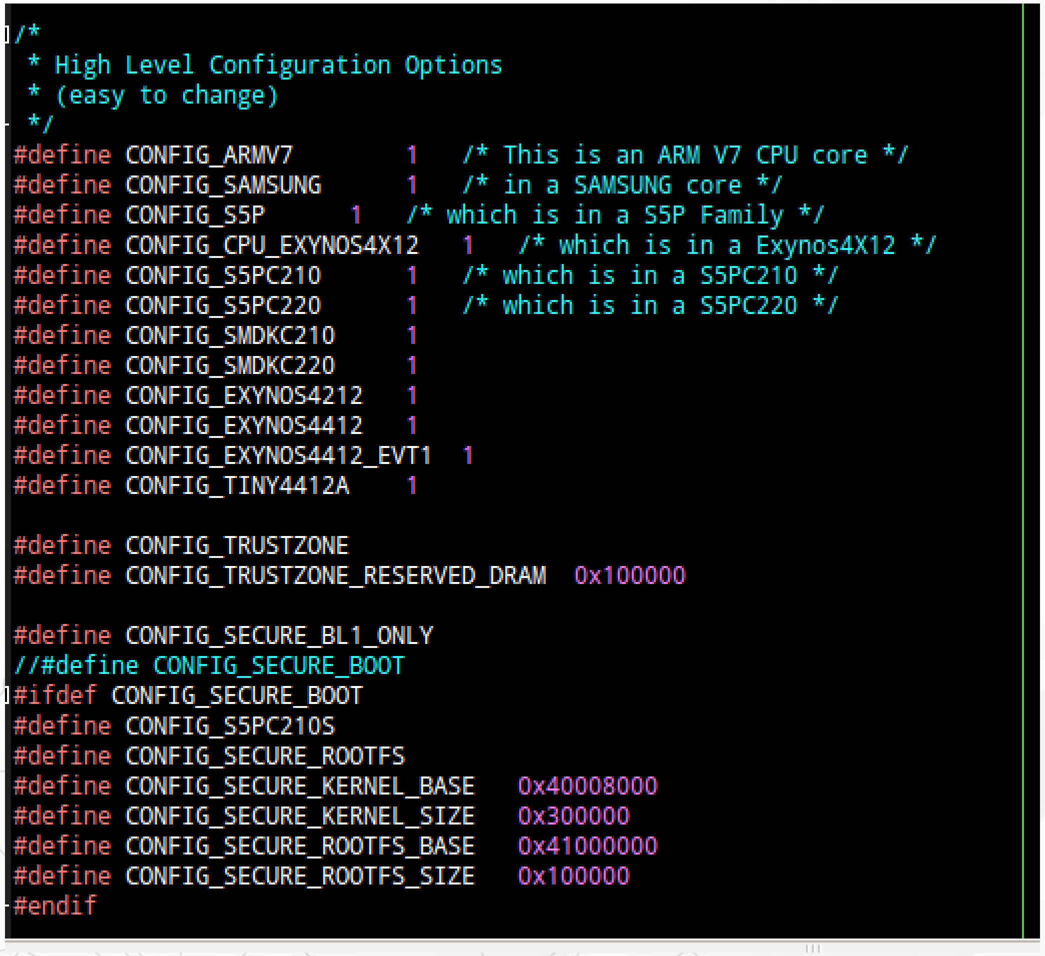
```
.
├── _ashldi3.S
├── _ashrdi3.S
├── board.c
├── bootm.c
├── cache.c
├── cache-cp15.c
├── div0.c
├── _divsi3.S
├── eabi_compat.c
├── interrupts.c
├── _lshrdi3.S
├── Makefile
├── _modsi3.S
├── reset.c
├── _udivsi3.S
└── _umodsi3.S
```



```
#endif
#ifdef CONFIG_LOGBUFFER
#ifndef CONFIG_ALT_LB_ADDR
		/* Also take the logbuffer into account (pram is
		pram += (LOGBUFF_LEN+LOGBUFF_OVERHEAD)/1024;
#endif
#endif
		sprintf((char *)memsz, "%ldk", (bd->bi_memsize /
		setenv("mem", (char *)memsz);
	}
#endif

	/* main_loop() can return to retry autoboot, if so ju
	for (;;) {
		main_loop();
	}

	/* NOTREACHED - no way out of command loop except boo
}
```

Configure

uboot_tiny4412-master/include/configs/tiny4412.h



```
/*
 * High Level Configuration Options
 * (easy to change)
 */
#define CONFIG_ARMV7        1   /* This is an ARM V7 CPU core */
#define CONFIG_SAMSUNG      1   /* in a SAMSUNG core */
#define CONFIG_S5P      1   /* which is in a S5P Family */
#define CONFIG_CPU_EXYNOS4X12   1   /* which is in a Exynos4X12 */
#define CONFIG_S5PC210      1   /* which is in a S5PC210 */
#define CONFIG_S5PC220      1   /* which is in a S5PC220 */
#define CONFIG_SMDKC210     1
#define CONFIG_SMDKC220     1
#define CONFIG_EXYNOS4212   1
#define CONFIG_EXYNOS4412   1
#define CONFIG_EXYNOS4412_EVT1  1
#define CONFIG_TINY4412A    1

#define CONFIG_TRUSTZONE
#define CONFIG_TRUSTZONE_RESERVED_DRAM  0x100000

#define CONFIG_SECURE_BL1_ONLY
//#define CONFIG_SECURE_BOOT
#ifdef CONFIG_SECURE_BOOT
#define CONFIG_S5PC210S
#define CONFIG_SECURE_ROOTFS
#define CONFIG_SECURE_KERNEL_BASE   0x40008000
#define CONFIG_SECURE_KERNEL_SIZE   0x300000
#define CONFIG_SECURE_ROOTFS_BASE   0x41000000
#define CONFIG_SECURE_ROOTFS_SIZE   0x100000
#endif
```

Boot Linux kernel

Boot Linux kernel

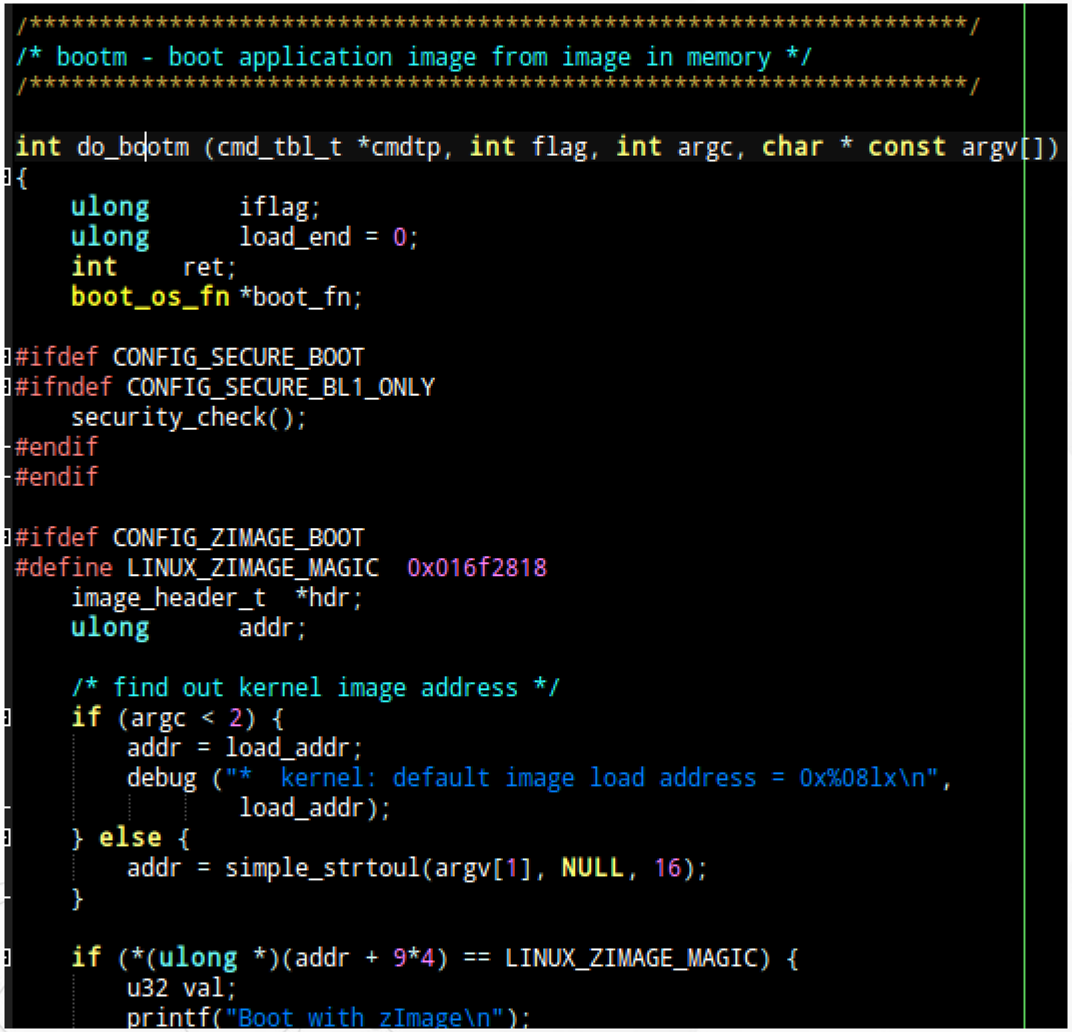
- Boot command
  - bootm
  - common/cmd_boot.c
- uImage
  - Tool : mkimage
- Linux kernel ATAG

How to jump to kernel

- Use boot command
  - common/cmd_bootm.c
  - command entry
    - int do_bootm (…)
    - boot_os_fn   *boot_fn;
- do_bootm_linux(...)
  - arch/arm/lib/bootm.c
- kernel_entry(0, machid, bd->bi_boot_params);
  - **0**
  - **Mach ID**
  - **atag**

uboot_tiny4412-master/common

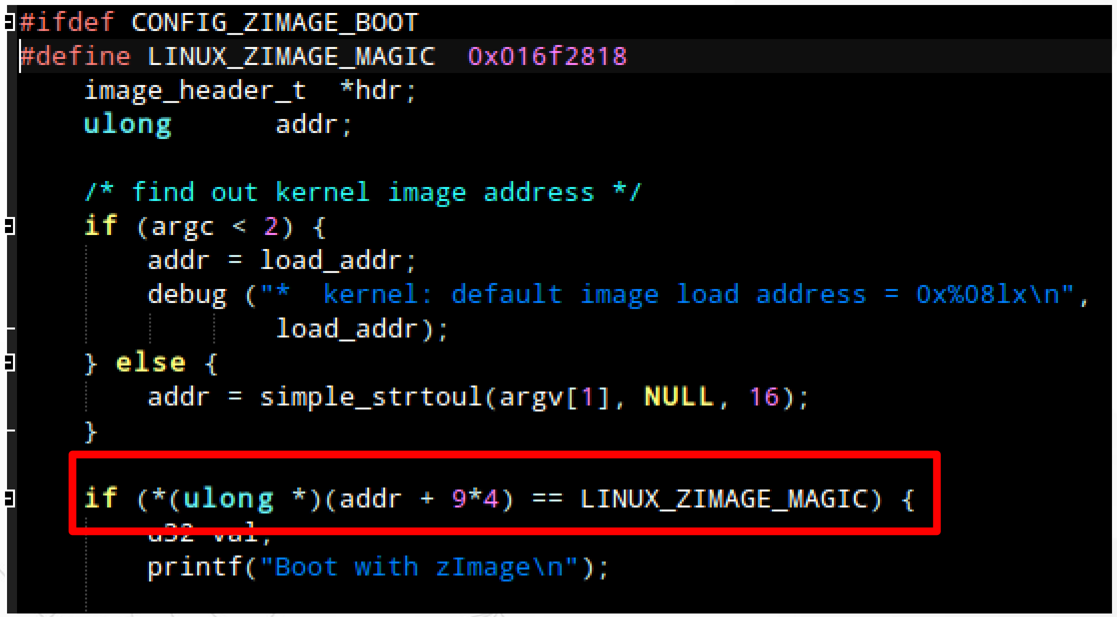
```
.
├── bedbug.c
├── cmd_ambapp.c
├── cmd_bdinfo.c
├── cmd_bedbug.c
├── cmd_bmp.c
├── cmd_boot.c
├── cmd_bootldr.c
├── cmd_bootm.c
```



```
/*******************************************************************/
/* bootm - boot application image from image in memory */
/*******************************************************************/

int do_bootm (cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
	ulong		iflag;
	ulong		load_end = 0;
	int		ret;
	boot_os_fn	*boot_fn;

#ifdef CONFIG_SECURE_BOOT
#ifndef CONFIG_SECURE_BL1_ONLY
	security_check();
#endif
#endif

#ifdef CONFIG_ZIMAGE_BOOT
#define LINUX_ZIMAGE_MAGIC	0x016f2818
	image_header_t	*hdr;
	ulong		addr;

	/* find out kernel image address */
	if (argc < 2) {
		addr = load_addr;
		debug ("*  kernel: default image load address = 0x%08lx\n",
				load_addr);
	} else {
		addr = simple_strtoul(argv[1], NULL, 16);
	}

	if (*(ulong *)(addr + 9*4) == LINUX_ZIMAGE_MAGIC) {
		u32 val;
		printf("Boot with zImage\n");
```

Check zImage format

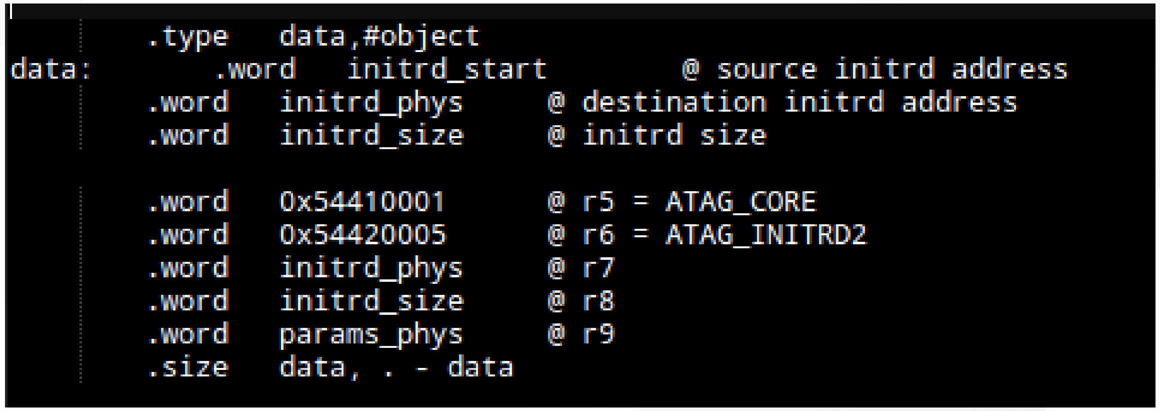
uboot_tiny4412-master/common/cmd_bootm.c



```
#ifdef CONFIG_ZIMAGE_BOOT
#define LINUX_ZIMAGE_MAGIC  0x016f2818
    image_header_t  *hdr;
    ulong       addr;

    /* find out kernel image address */
    if (argc < 2) {
        addr = load_addr;
        debug ("*  kernel: default image load address = 0x%08lx\n",
                load_addr);
    } else {
        addr = simple_strtoul(argv[1], NULL, 16);
    }

    if (*(ulong *)(addr + 9*4) == LINUX_ZIMAGE_MAGIC) {
        u32 val;
        printf("Boot with zImage\n");
```

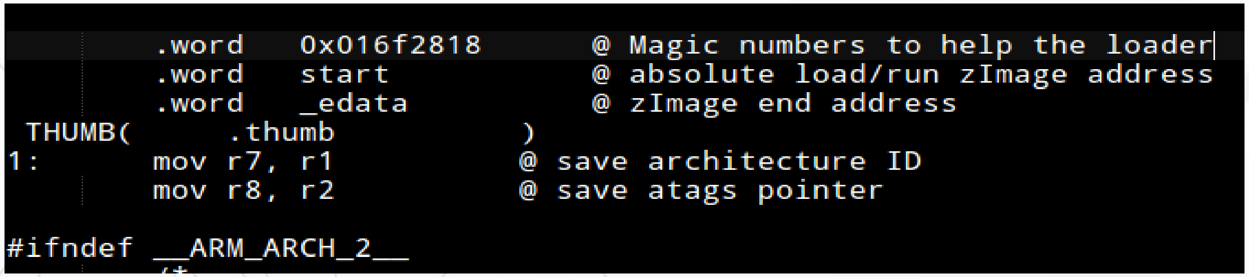
Check zImage format

tiny4412/arch/arm/boot/bootp/init.S



```
        .type   data,#object
data:       .word   initrd_start        @ source initrd address
        .word   initrd_phys     @ destination initrd address
        .word   initrd_size     @ initrd size

        .word   0x54410001      @ r5 = ATAG_CORE
        .word   0x54420005      @ r6 = ATAG_INITRD2
        .word   initrd_phys     @ r7
        .word   initrd_size     @ r8
        .word   params_phys     @ r9
        .size   data, . - data
```

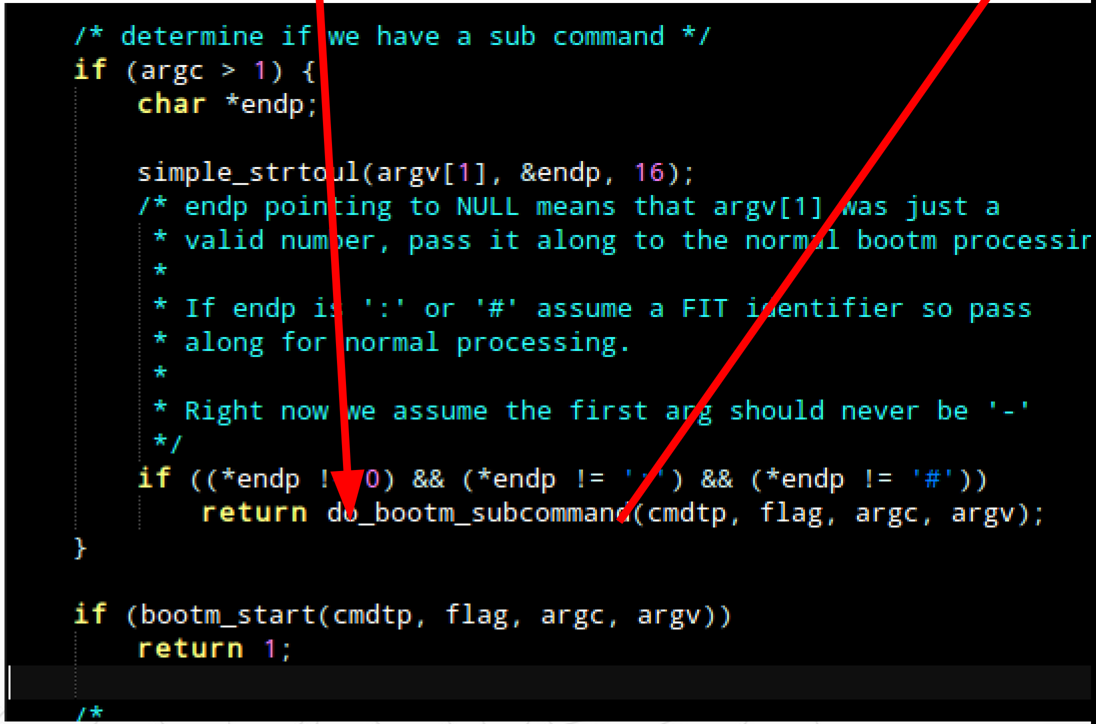
linux_3.5.0_tiny4412/arch/arm/boot/compressed/head.S



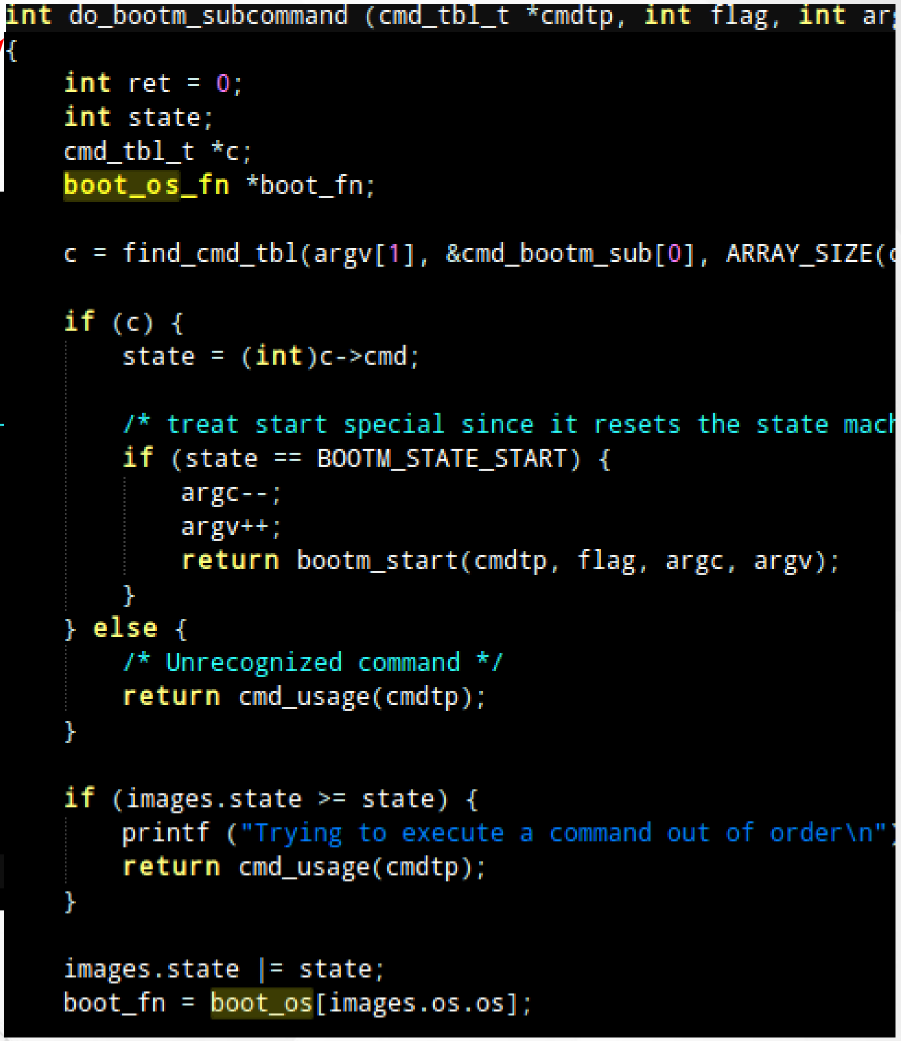
```
        .word   0x016f2818      @ Magic numbers to help the loader
        .word   start           @ absolute load/run zImage address
        .word   _edata          @ zImage end address
 THUMB(     .thumb          )
1:      mov r7, r1          @ save architecture ID
        mov r8, r2          @ save atags pointer

#ifndef __ARM_ARCH_2__
```

Prepare Jmp to Linux kernel

cmd_bootm.c



```
    /* determine if we have a sub command */
    if (argc > 1) {
        char *endp;

        simple_strtoul(argv[1], &endp, 16);
        /* endp pointing to NULL means that argv[1] was just a
         * valid number, pass it along to the normal bootm processin
         *
         * If endp is ':' or '#' assume a FIT identifier so pass
         * along for normal processing.
         *
         * Right now we assume the first arg should never be '-'
         */
        if ((*endp != 0) && (*endp != ':') && (*endp != '#'))
            return do_bootm_subcommand(cmdtp, flag, argc, argv);
    }

    if (bootm_start(cmdtp, flag, argc, argv))
        return 1;

    /*
```

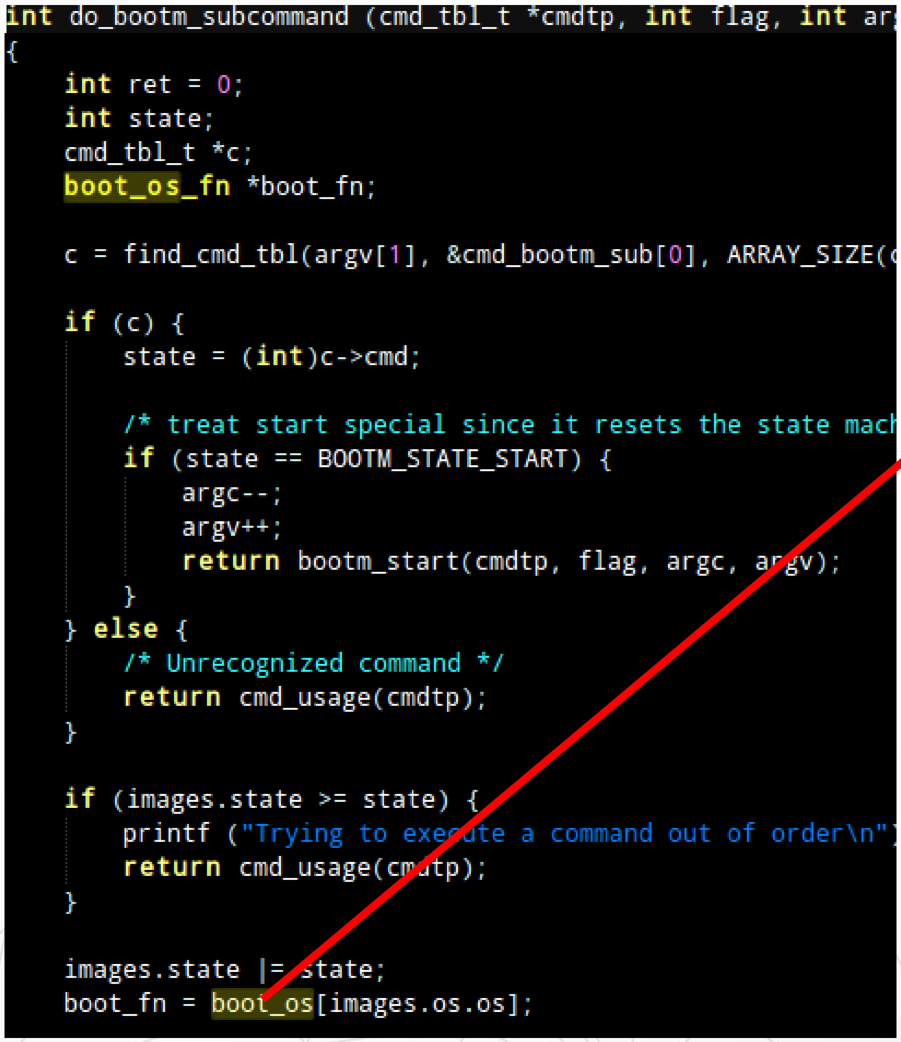


```
int do_bootm_subcommand (cmd_tbl_t *cmdtp, int flag, int ar
{
    int ret = 0;
    int state;
    cmd_tbl_t *c;
    boot_os_fn *boot_fn;

    c = find_cmd_tbl(argv[1], &cmd_bootm_sub[0], ARRAY_SIZE(c

    if (c) {
        state = (int)c->cmd;

        /* treat start special since it resets the state mach
        if (state == BOOTM_STATE_START) {
            argc--;
            argv++;
            return bootm_start(cmdtp, flag, argc, argv);
        }
    } else {
        /* Unrecognized command */
        return cmd_usage(cmdtp);
    }

    if (images.state >= state) {
        printf ("Trying to execute a command out of order\n")
        return cmd_usage(cmdtp);
    }

    images.state |= state;
    boot_fn = boot_os[images.os.os];
```

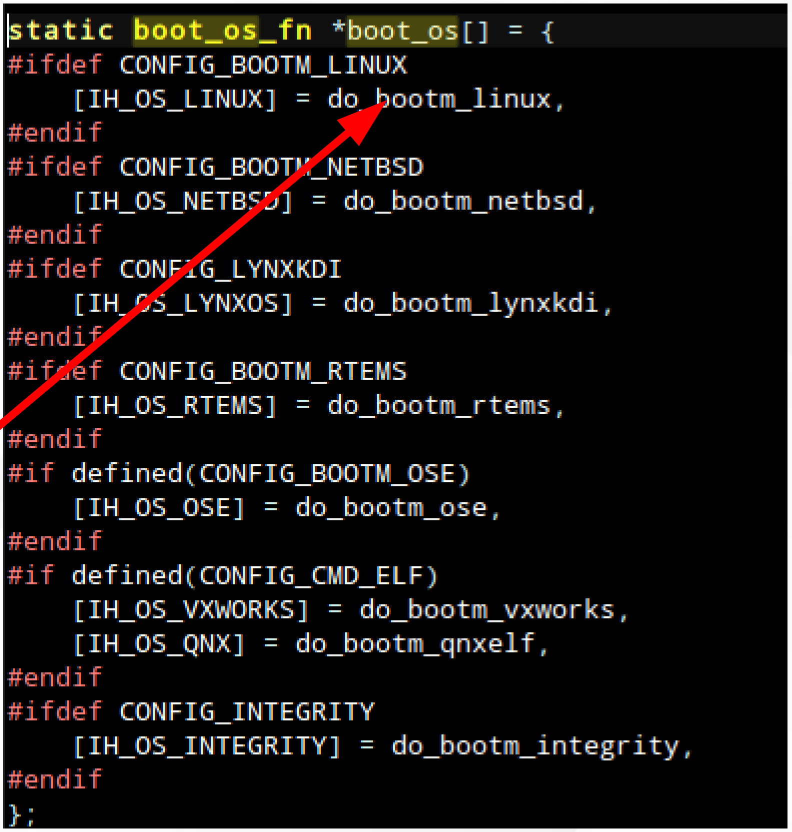
中華行動數位科技
Chinese Action Digital Technology

Prepare Jmp to Linux kernel

cmd_bootm.c



```
int do_bootm_subcommand (cmd_tbl_t *cmdtp, int flag, int ar
{
    int ret = 0;
    int state;
    cmd_tbl_t *c;
    boot_os_fn *boot_fn;

    c = find_cmd_tbl(argv[1], &cmd_bootm_sub[0], ARRAY_SIZE(

    if (c) {
        state = (int)c->cmd;

        /* treat start special since it resets the state mac
        if (state == BOOTM_STATE_START) {
            argc--;
            argv++;
            return bootm_start(cmdtp, flag, argc, argv);
        }
    } else {
        /* Unrecognized command */
        return cmd_usage(cmdtp);
    }

    if (images.state >= state) {
        printf ("Trying to execute a command out of order\n"
        return cmd_usage(cmdtp);
    }

    images.state |= state;
    boot_fn = boot_os[images.os.os];
```

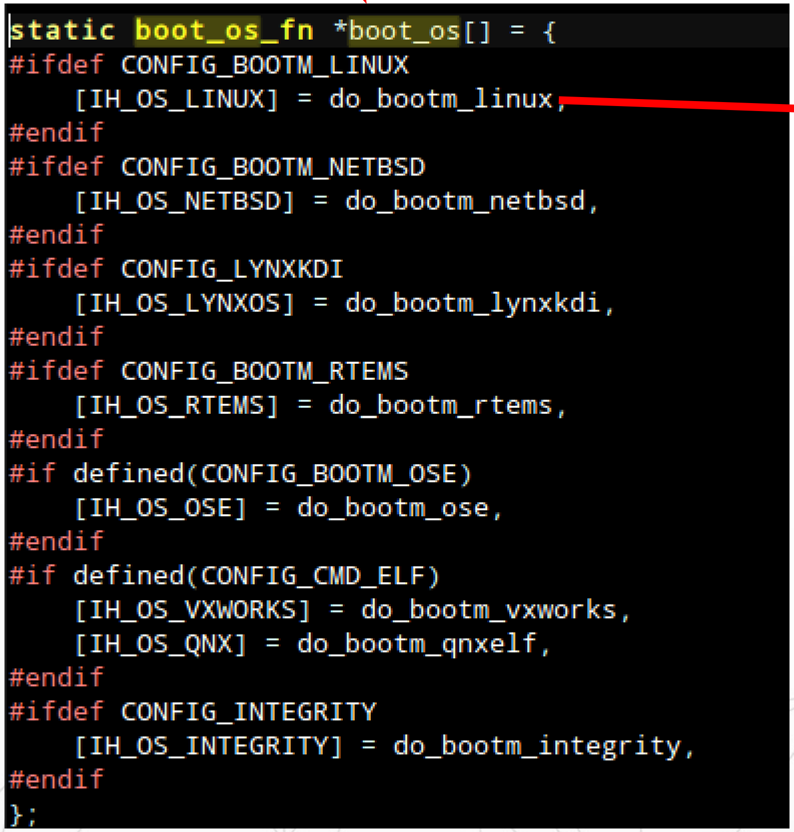


```
static boot_os_fn *boot_os[] = {
#ifdef CONFIG_BOOTM_LINUX
    [IH_OS_LINUX] = do_bootm_linux,
#endif
#ifdef CONFIG_BOOTM_NETBSD
    [IH_OS_NETBSD] = do_bootm_netbsd,
#endif
#ifdef CONFIG_LYNXKDI
    [IH_OS_LYNXOS] = do_bootm_lynxkdi,
#endif
#ifdef CONFIG_BOOTM_RTEMS
    [IH_OS_RTEMS] = do_bootm_rtems,
#endif
#if defined(CONFIG_BOOTM_OSE)
    [IH_OS_OSE] = do_bootm_ose,
#endif
#if defined(CONFIG_CMD_ELF)
    [IH_OS_VXWORKS] = do_bootm_vxworks,
    [IH_OS_QNX] = do_bootm_qnxelf,
#endif
#ifdef CONFIG_INTEGRITY
    [IH_OS_INTEGRITY] = do_bootm_integrity,
#endif
};
```

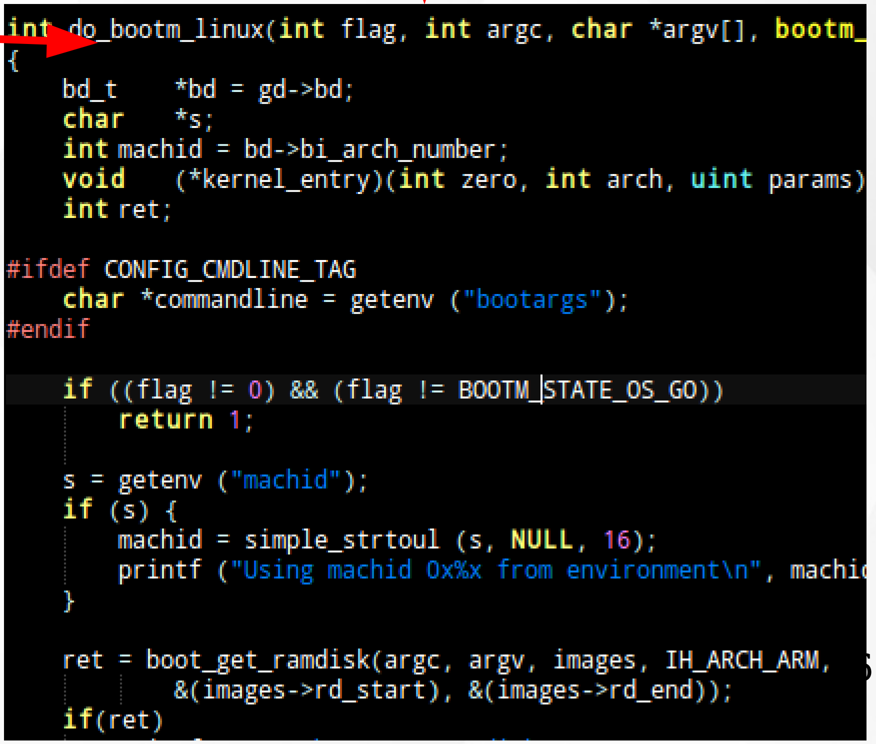
Prepare Jmp to Linux kernel

uboot_tiny4412-master/common/cmd_bootm.c

uboot_tiny4412-master/arch/arm/lib



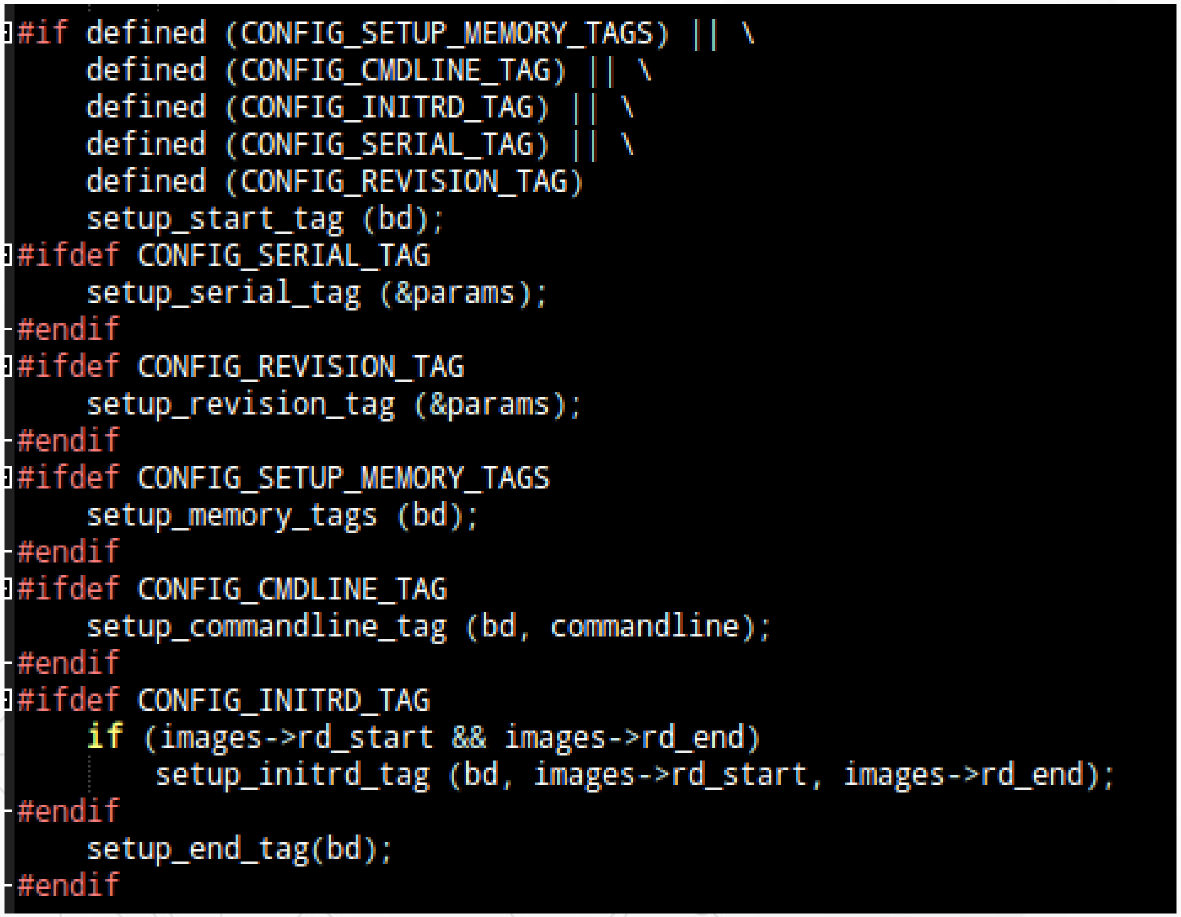
```
static boot_os_fn *boot_os[] = {
#ifdef CONFIG_BOOTM_LINUX
    [IH_OS_LINUX] = do_bootm_linux,
#endif
#ifdef CONFIG_BOOTM_NETBSD
    [IH_OS_NETBSD] = do_bootm_netbsd,
#endif
#ifdef CONFIG_LYNXKDI
    [IH_OS_LYNXOS] = do_bootm_lynxkdi,
#endif
#ifdef CONFIG_BOOTM_RTEMS
    [IH_OS_RTEMS] = do_bootm_rtems,
#endif
#if defined(CONFIG_BOOTM_OSE)
    [IH_OS_OSE] = do_bootm_ose,
#endif
#if defined(CONFIG_CMD_ELF)
    [IH_OS_VXWORKS] = do_bootm_vxworks,
    [IH_OS_QNX] = do_bootm_qnxelf,
#endif
#ifdef CONFIG_INTEGRITY
    [IH_OS_INTEGRITY] = do_bootm_integrity,
#endif
};
```



```
int do_bootm_linux(int flag, int argc, char *argv[], bootm_
{
    bd_t    *bd = gd->bd;
    char    *s;
    int machid = bd->bi_arch_number;
    void    (*kernel_entry)(int zero, int arch, uint params)
    int ret;

#ifdef CONFIG_CMDLINE_TAG
    char *commandline = getenv ("bootargs");
#endif

    if ((flag != 0) && (flag != BOOTM_STATE_OS_GO))
        return 1;

    s = getenv ("machid");
    if (s) {
        machid = simple_strtoul (s, NULL, 16);
        printf ("Using machid 0x%x from environment\n", machic
    }

    ret = boot_get_ramdisk(argc, argv, images, IH_ARCH_ARM,
            &(images->rd_start), &(images->rd_end));
    if(ret)
```

Setup ATAG

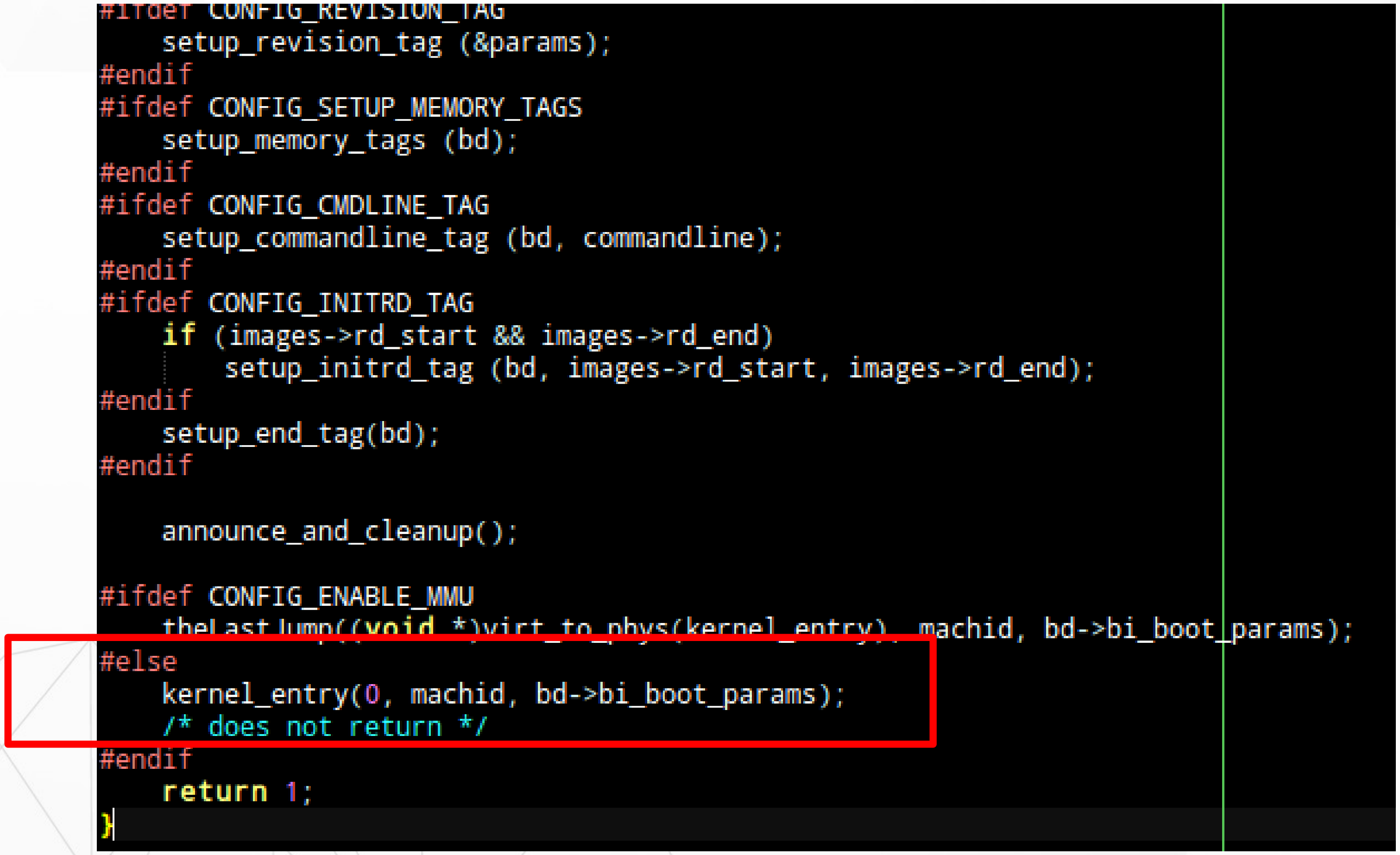
uboot_tiny4412-master/arch/arm/lib/bootm.c



```
#if defined (CONFIG_SETUP_MEMORY_TAGS) || \
    defined (CONFIG_CMDLINE_TAG) || \
    defined (CONFIG_INITRD_TAG) || \
    defined (CONFIG_SERIAL_TAG) || \
    defined (CONFIG_REVISION_TAG)
    setup_start_tag (bd);
#ifdef CONFIG_SERIAL_TAG
    setup_serial_tag (&params);
#endif
#ifdef CONFIG_REVISION_TAG
    setup_revision_tag (&params);
#endif
#ifdef CONFIG_SETUP_MEMORY_TAGS
    setup_memory_tags (bd);
#endif
#ifdef CONFIG_CMDLINE_TAG
    setup_commandline_tag (bd, commandline);
#endif
#ifdef CONFIG_INITRD_TAG
    if (images->rd_start && images->rd_end)
        setup_initrd_tag (bd, images->rd_start, images->rd_end);
#endif
    setup_end_tag(bd);
#endif
```

Jmp to Linux kernel

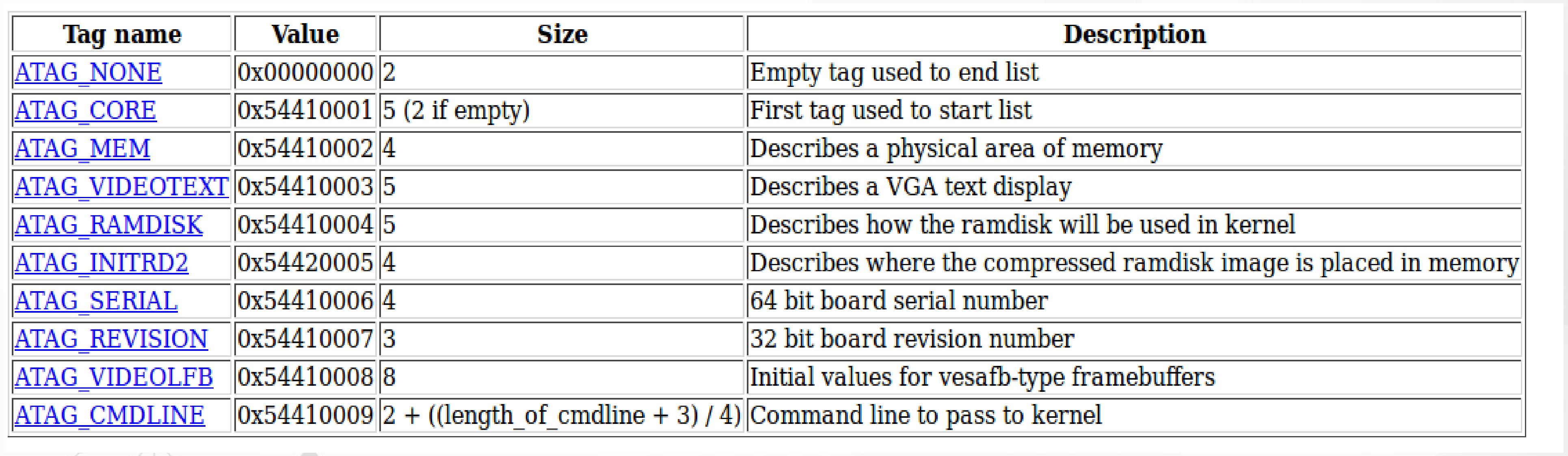
uboot_tiny4412-master/arch/arm/lib/bootm.c



```
#ifdef CONFIG_REVISION_TAG
    setup_revision_tag (&params);
#endif
#ifdef CONFIG_SETUP_MEMORY_TAGS
    setup_memory_tags (bd);
#endif
#ifdef CONFIG_CMDLINE_TAG
    setup_commandline_tag (bd, commandline);
#endif
#ifdef CONFIG_INITRD_TAG
    if (images->rd_start && images->rd_end)
        setup_initrd_tag (bd, images->rd_start, images->rd_end);
#endif
    setup_end_tag(bd);
#endif

    announce_and_cleanup();

#ifdef CONFIG_ENABLE_MMU
    theLastJump((void *)virt_to_phys(kernel_entry), machid, bd->bi_boot_params);
#else
    kernel_entry(0, machid, bd->bi_boot_params);
    /* does not return */
#endif
    return 1;
}
```

Linux Atag

- Kernel parameters

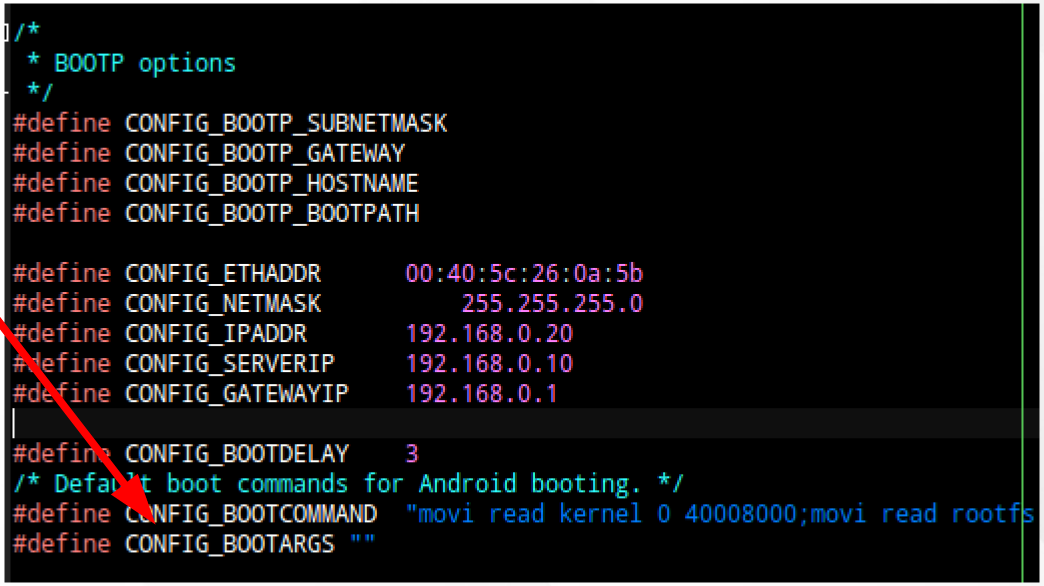


Tag name	Value	Size	Description
ATAG_NONE	0x00000000	2	Empty tag used to end list
ATAG_CORE	0x54410001	5 (2 if empty)	First tag used to start list
ATAG_MEM	0x54410002	4	Describes a physical area of memory
ATAG_VIDEOTEXT	0x54410003	5	Describes a VGA text display
ATAG_RAMDISK	0x54410004	5	Describes how the ramdisk will be used in kernel
ATAG_INITRD2	0x54420005	4	Describes where the compressed ramdisk image is placed in memory
ATAG_SERIAL	0x54410006	4	64 bit board serial number
ATAG_REVISION	0x54410007	3	32 bit board revision number
ATAG_VIDEOLFB	0x54410008	8	Initial values for vesafb-type framebuffers
ATAG_CMDLINE	0x54410009	2 + ((length_of_cmdline + 3) / 4)	Command line to pass to kernel

Boot Args

/include/configs/tiny4412.h

#define CONFIG_BOOTARGS



```
/*
 * BOOTP options
 */
#define CONFIG_BOOTP_SUBNETMASK
#define CONFIG_BOOTP_GATEWAY
#define CONFIG_BOOTP_HOSTNAME
#define CONFIG_BOOTP_BOOTPATH

#define CONFIG_ETHADDR        00:40:5c:26:0a:5b
#define CONFIG_NETMASK            255.255.255.0
#define CONFIG_IPADDR         192.168.0.20
#define CONFIG_SERVERIP       192.168.0.10
#define CONFIG_GATEWAYIP      192.168.0.1

#define CONFIG_BOOTDELAY      3
/* Default boot commands for Android booting. */
#define CONFIG_BOOTCOMMAND    "movi read kernel 0 40008000;movi read rootfs
#define CONFIG_BOOTARGS ""
```

bootargs :
noinitrd init=/linuxrc root=/dev/nfs ip=192.168.0.20:192.168.0.10:192.168.0.1:255.255.255.0::eth0:on nfsroot=192.168.0.10:/home/cadtc/tiny4412/experiment/root_mkfs, ip=192.168.0.20 console=ttySAC0 lcd=S70

Boot Args

noinitrd
No use RamDisk

kernel into rootfs will excuse /.linuxrc application first
init=/linuxrc

NFS ip setting
ip=192.168.0.20:192.168.0.10:192.168.0.1:255.255.255.0

Host	IP	->	192.168.0.20
Device	IP	->	192.168.0.10:
Getway	IP	->	192.168.0.1:
IP Mask		->	255.255.255.0

rootfs path in the network
nfsroot=192.168.0.10:/home/cadtc/tiny4412/experiment/root_mkfs

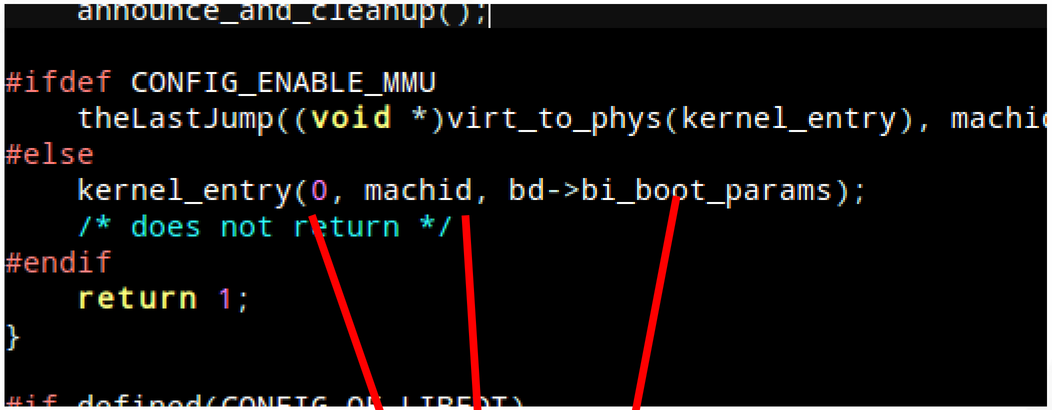
Setting console interface is ttySAC0
console=ttySAC0

Setting panel type
lcd=S70

Add Feature

- Add command
    - common/
- Add driver
    - driver
- Add application
    - example
- boards.cfg

Linux Enter Point



```
    announce_and_cleanup();

#ifdef CONFIG_ENABLE_MMU
    theLastJump((void *)virt_to_phys(kernel_entry), machi
#else
    kernel_entry(0, machid, bd->bi_boot_params);
    /* does not return */
#endif
    return 1;
}

#if defined(CONFIG_OF_LIBFDT)
```

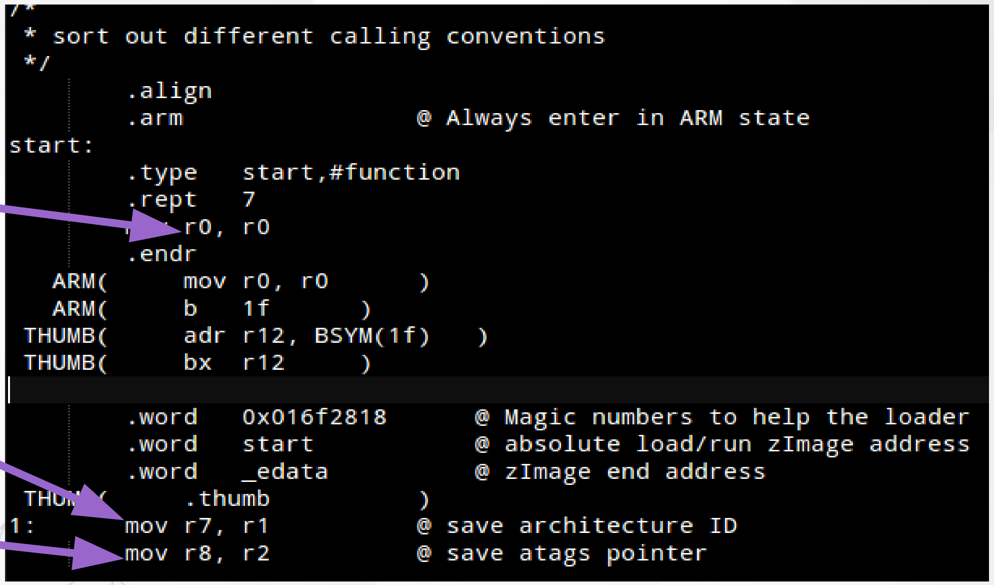
$uboot_folder/arch/arm/lib/bootm.c

R0 : 0

R1 : mach ID

R2 : ATAG

tiny4412/arch/arm/boot/compressed/head.S



```
/*
 * sort out different calling conventions
 */
        .align
        .arm                    @ Always enter in ARM state
start:
        .type   start,#function
        .rept   7
        mov     r0, r0
        .endr
   ARM(     mov r0, r0       )
   ARM(     b   1f           )
 THUMB(     adr r12, BSYM(1f)    )
 THUMB(     bx  r12          )

        .word   0x016f2818      @ Magic numbers to help the loader
        .word   start           @ absolute load/run zImage address
        .word   _edata          @ zImage end address
 THUMB(     .thumb             )
1:      mov r7, r1              @ save architecture ID
        mov r8, r2              @ save atags pointer
```

Add Command

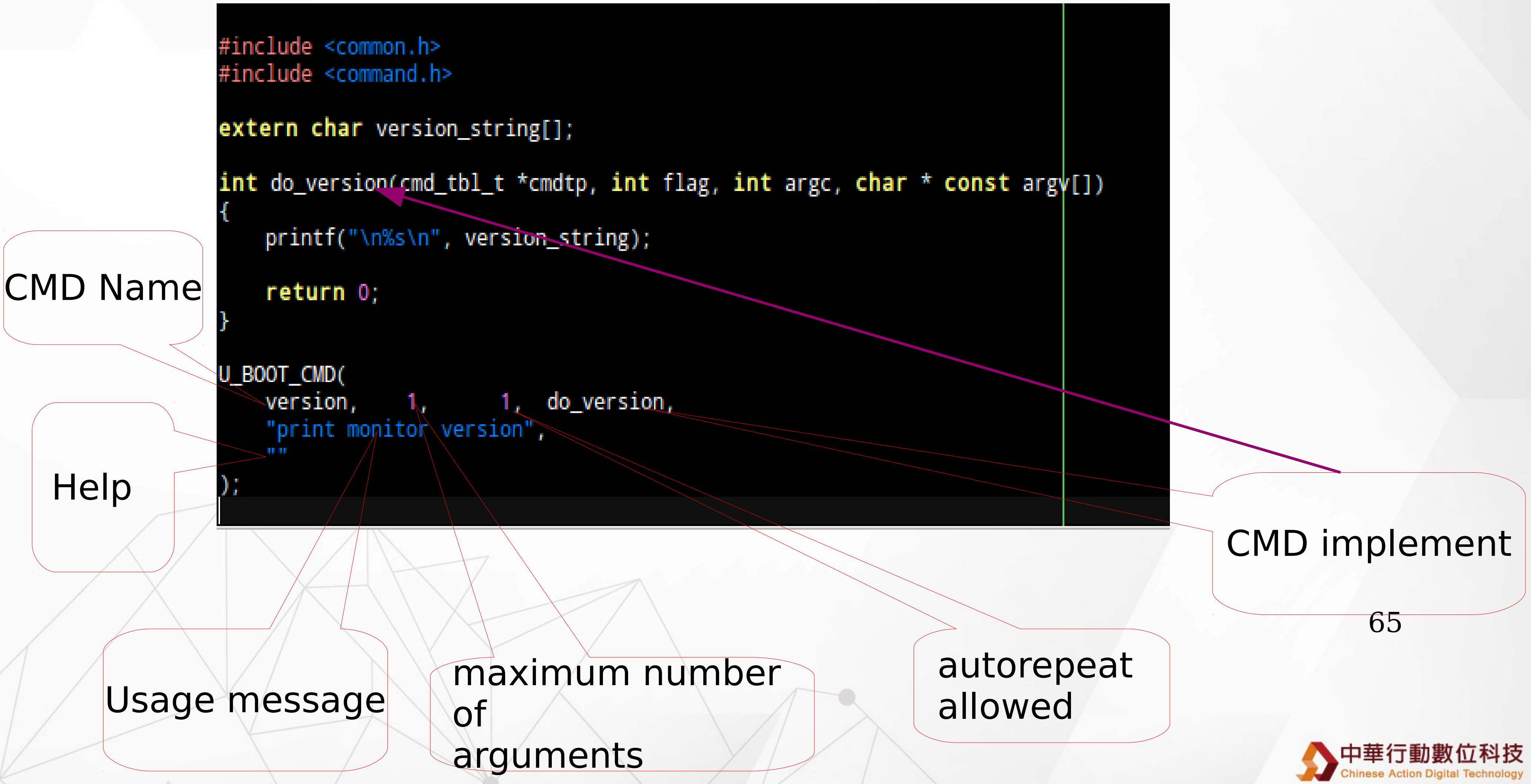
- How to create a command ?

- Directory
  - common/
    - cmd_mmc.c, cmd_bootm.c, cmd_help.c

- U_BOOT_CMD(name,maxargs,rep,cmd,usage,help)
  - include/command.h

How to Command

uboot_tiny4412-master/common/cmd_version.c



```
#include <common.h>
#include <command.h>

extern char version_string[];

int do_version(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    printf("\n%s\n", version_string);

    return 0;
}

U_BOOT_CMD(
    version,    1,      1,  do_version,
    "print monitor version",
    ""
);
```

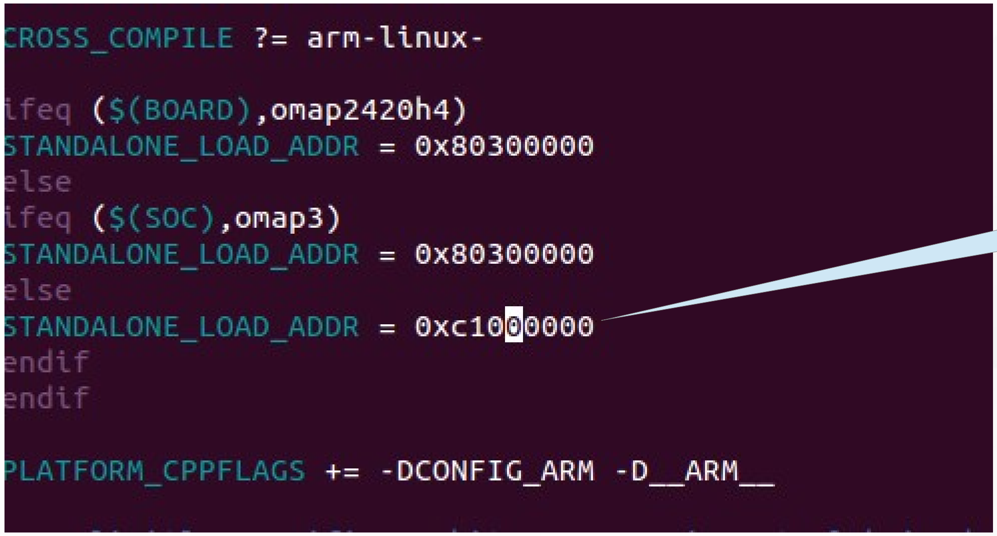
中華行動數位科技
Chinese Action Digital Technology

Add application

- How to add a application
- Directory
  - $(UBBOT)/example/standalone
  - arch/arm/config.mk
    - STANDALONE_LOAD_ADDR
- Example
  - LED sample
    - $(UBBOT)/example/standalone/led_sample.c
    - drivers/gpio/s5p_gpio.c

arch/arm/config.mk

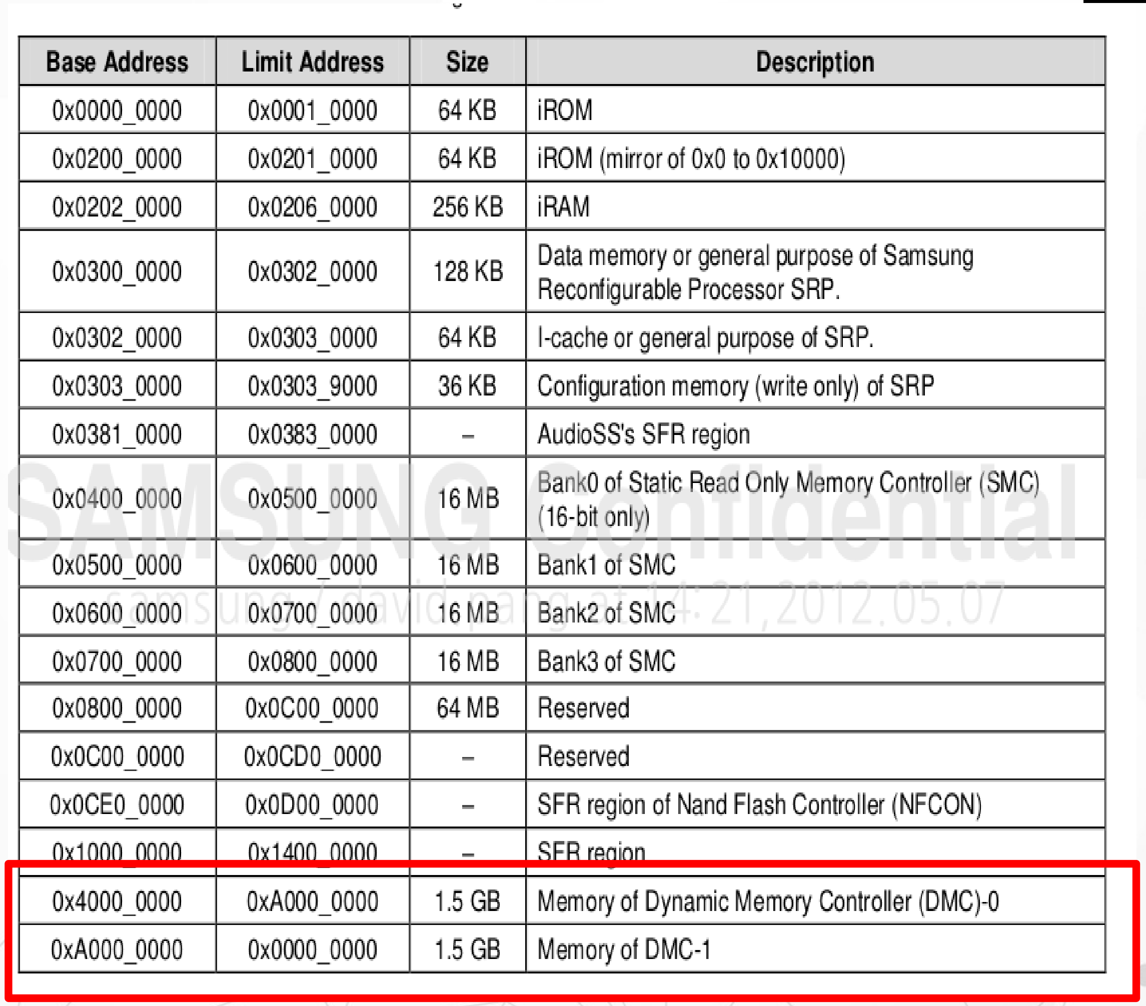
$(uboot)arch/arm/config.mk



```
CROSS_COMPILE ?= arm-linux-

ifeq ($(BOARD),omap2420h4)
STANDALONE_LOAD_ADDR = 0x80300000
else
ifeq ($(SOC),omap3)
STANDALONE_LOAD_ADDR = 0x80300000
else
STANDALONE_LOAD_ADDR = 0xc1000000
endif
endif

PLATFORM_CPPFLAGS += -DCONFIG_ARM -D__ARM__
```

STANDALONE LOAD ADDR

EVB DDR PHY ADDRESS & SYSTEM_MAP



Base Address	Limit Address	Size	Description
0x0000_0000	0x0001_0000	64 KB	iROM
0x0200_0000	0x0201_0000	64 KB	iROM (mirror of 0x0 to 0x10000)
0x0202_0000	0x0206_0000	256 KB	iRAM
0x0300_0000	0x0302_0000	128 KB	Data memory or general purpose of Samsung Reconfigurable Processor SRP.
0x0302_0000	0x0303_0000	64 KB	I-cache or general purpose of SRP.
0x0303_0000	0x0303_9000	36 KB	Configuration memory (write only) of SRP
0x0381_0000	0x0383_0000	–	AudioSS's SFR region
0x0400_0000	0x0500_0000	16 MB	Bank0 of Static Read Only Memory Controller (SMC) (16-bit only)
0x0500_0000	0x0600_0000	16 MB	Bank1 of SMC
0x0600_0000	0x0700_0000	16 MB	Bank2 of SMC
0x0700_0000	0x0800_0000	16 MB	Bank3 of SMC
0x0800_0000	0x0C00_0000	64 MB	Reserved
0x0C00_0000	0x0CD0_0000	–	Reserved
0x0CE0_0000	0x0D00_0000	–	SFR region of Nand Flash Controller (NFCON)
0x1000_0000	0x1400_0000	–	SFR region
0x4000_0000	0xA000_0000	1.5 GB	Memory of Dynamic Memory Controller (DMC)-0
0xA000_0000	0x0000_0000	1.5 GB	Memory of DMC-1