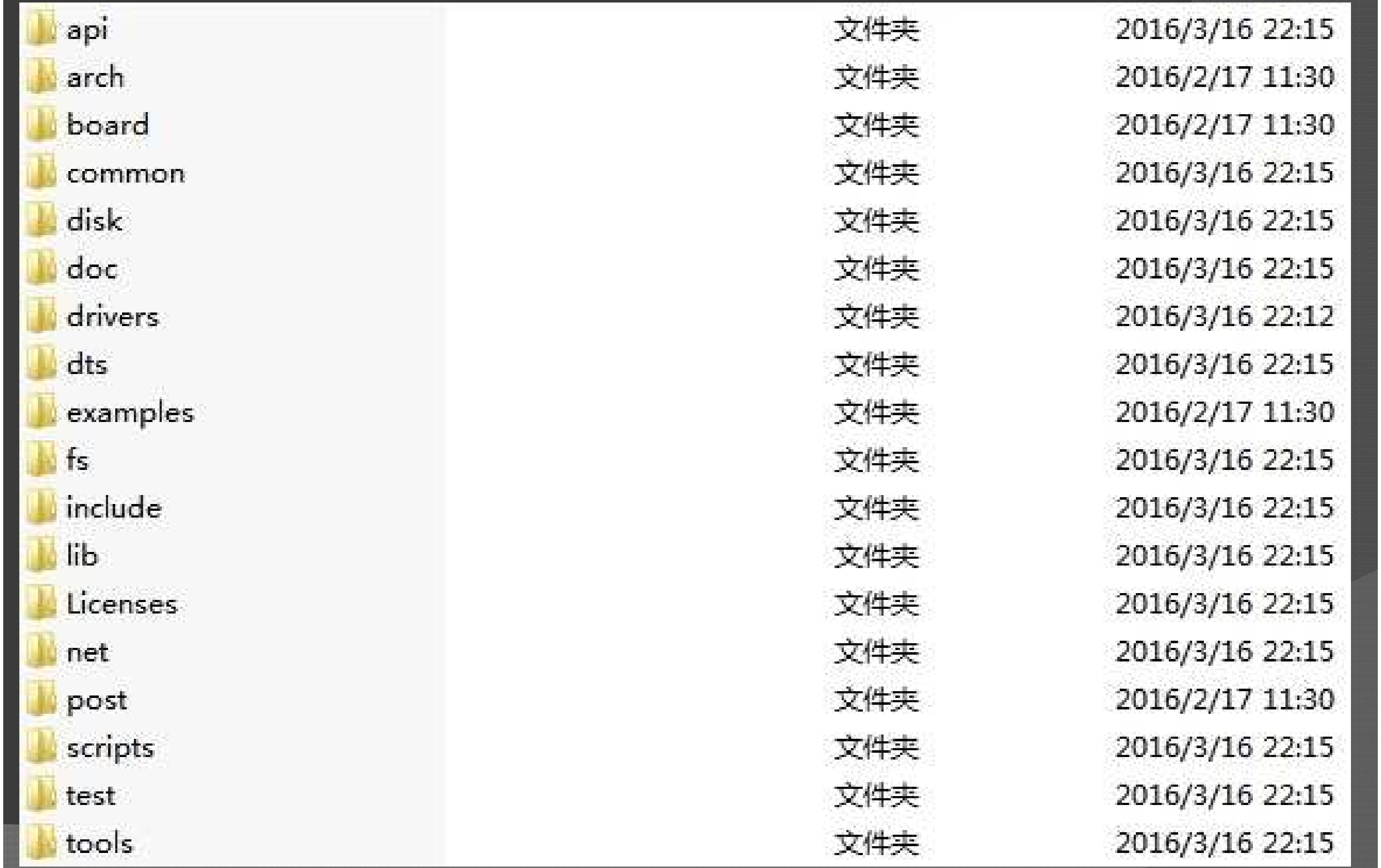
Uboot　2014.04 启动流程分析

廖荣辉

➢ Uboot目录结构



api	文件夹	2016/3/16 22:15
arch	文件夹	2016/2/17 11:30
board	文件夹	2016/2/17 11:30
common	文件夹	2016/3/16 22:15
disk	文件夹	2016/3/16 22:15
doc	文件夹	2016/3/16 22:15
drivers	文件夹	2016/3/16 22:12
dts	文件夹	2016/3/16 22:15
examples	文件夹	2016/2/17 11:30
fs	文件夹	2016/3/16 22:15
include	文件夹	2016/3/16 22:15
lib	文件夹	2016/3/16 22:15
Licenses	文件夹	2016/3/16 22:15
net	文件夹	2016/3/16 22:15
post	文件夹	2016/2/17 11:30
scripts	文件夹	2016/3/16 22:15
test	文件夹	2016/3/16 22:15
tools	文件夹	2016/3/16 22:15

➢ Uboot目录结构

- api 此目录下存放u-boot向外提供的接口函数
- arch 与体系结构相关的代码。
- board 是根据不同的具体开发板而定制的代码。
- common 通用代码，涵盖各个方面，以命令行处理为主
- disk 磁盘分区相关代码
- doc 常见功能和问题的说明文档，一堆README开头的文件
- drivers 常用的设备驱动程序，每个类型的设备驱动占用一个子目录
- examples 示例程序
- fs 文件系统，支持嵌入式开发常见fs(cramfs,ext2,ext3,jffs2,etc)
- include 全局需要的头文件定义在这儿
- lib 通用库文件
- net 网络相关的代码，小型的协议栈
- post Power On Self Test，上点自检 程序
- Tools 辅助程序，用于编译和检查uboot目标文件

➢ Uboot目录结构

- boards.cfg ： 所有支持的板子的基本信息
- config.mk ,Makefile:顶层目录下的Makefile文件
- mkconfig: shell 脚本文件

➢ Uboot 编译过程-make xxx_config

- ◉ make xxx_config
- ✓ xxx ：开发板名字
- ✓ 匹配 Makefile 中的目标

```
%_config:: outputmakefile
	@$(MKCONFIG) -A $(@:_config =)
```

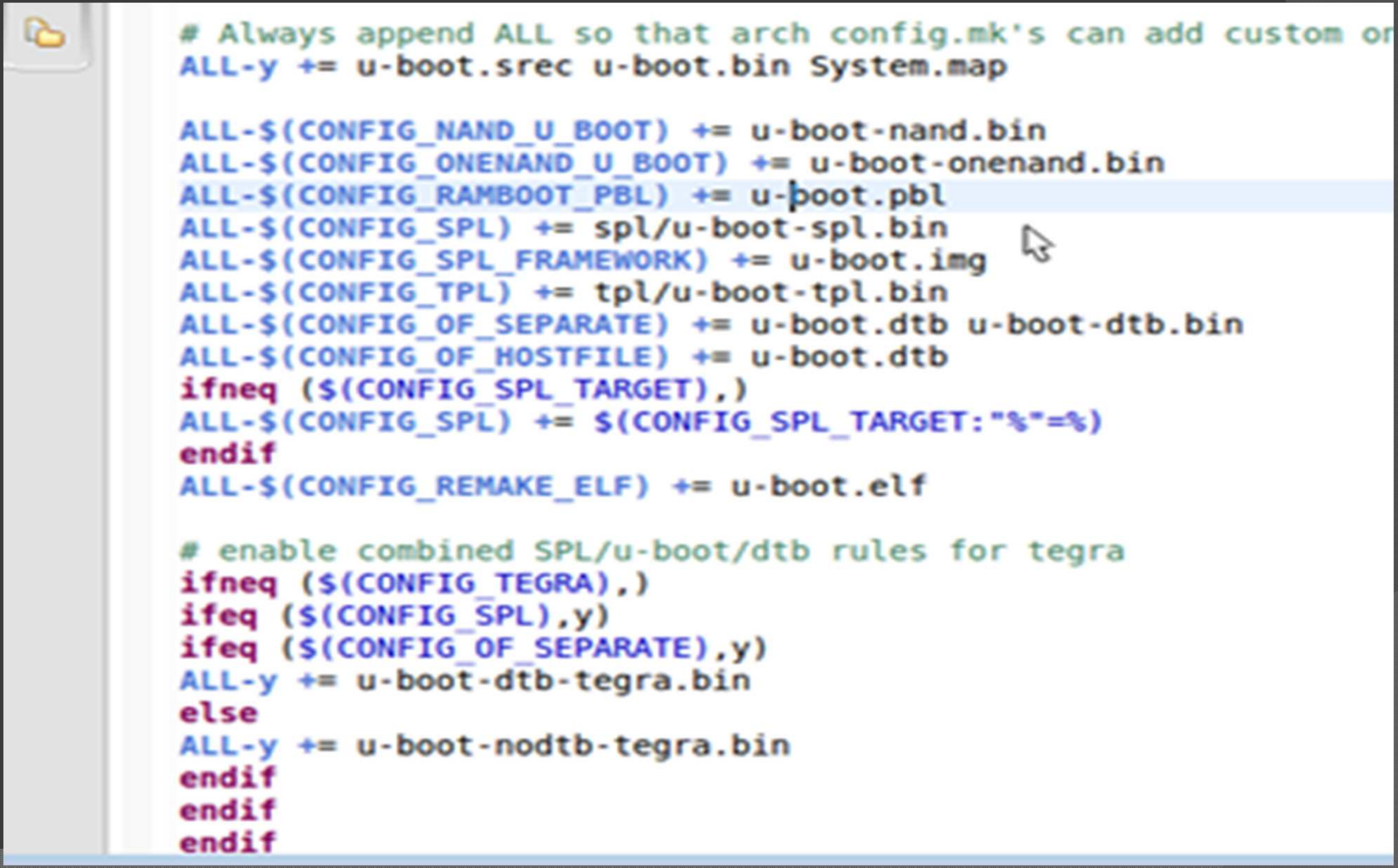
- ✓ 通过 mkconfig shell脚本 从boards.cfg 中获取开发板基本信息到 include/config.mk,并生成 config.h

➢ Uboot 编译过程-make all

- make (all) CROSS_COMPILE=
- ✓ CROSS_COMPILE:设置交叉编译工具
- ✓ 匹配 Makefile 中的目标

  all:$(ALL-y)

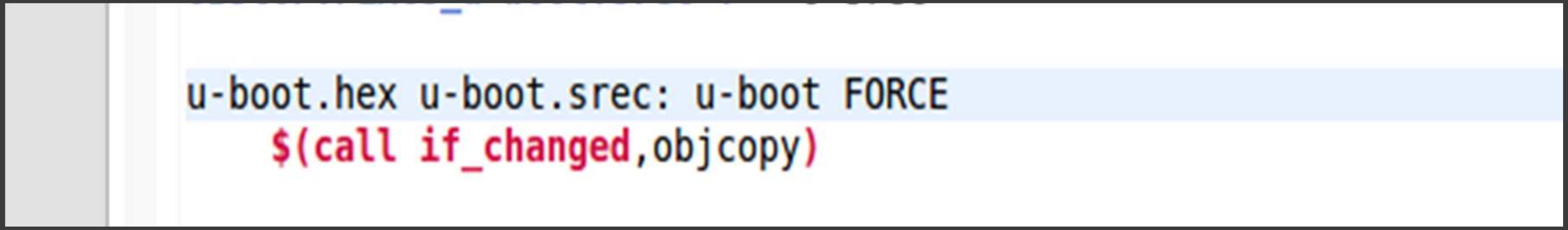
➢ Uboot 编译过程-make all



```
# Always append ALL so that arch config.mk's can add custom or
ALL-y += u-boot.srec u-boot.bin System.map

ALL-$(CONFIG_NAND_U_BOOT) += u-boot-nand.bin
ALL-$(CONFIG_ONENAND_U_BOOT) += u-boot-onenand.bin
ALL-$(CONFIG_RAMBOOT_PBL) += u-boot.pbl
ALL-$(CONFIG_SPL) += spl/u-boot-spl.bin
ALL-$(CONFIG_SPL_FRAMEWORK) += u-boot.img
ALL-$(CONFIG_TPL) += tpl/u-boot-tpl.bin
ALL-$(CONFIG_OF_SEPARATE) += u-boot.dtb u-boot-dtb.bin
ALL-$(CONFIG_OF_HOSTFILE) += u-boot.dtb
ifneq ($(CONFIG_SPL_TARGET),)
ALL-$(CONFIG_SPL) += $(CONFIG_SPL_TARGET:"%"=%)
endif
ALL-$(CONFIG_REMAKE_ELF) += u-boot.elf

# enable combined SPL/u-boot/dtb rules for tegra
ifneq ($(CONFIG_TEGRA),)
ifeq ($(CONFIG_SPL),y)
ifeq ($(CONFIG_OF_SEPARATE),y)
ALL-y += u-boot-dtb-tegra.bin
else
ALL-y += u-boot-nodtb-tegra.bin
endif
endif
endif
```

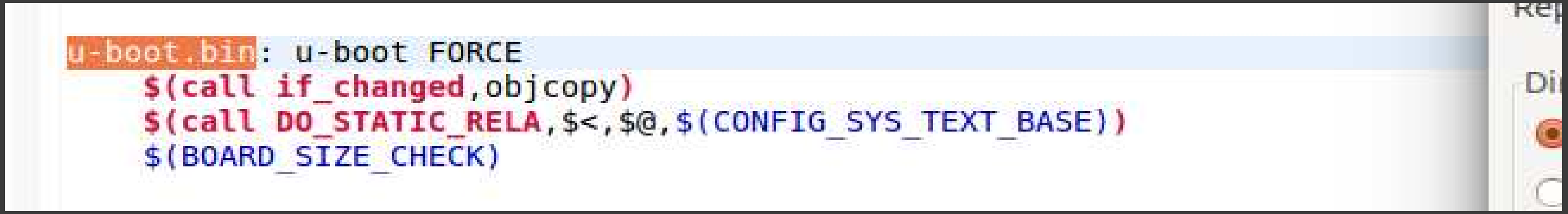
➢ Uboot 编译过程-make all

- ✓ u-boot.srec ： u-boot



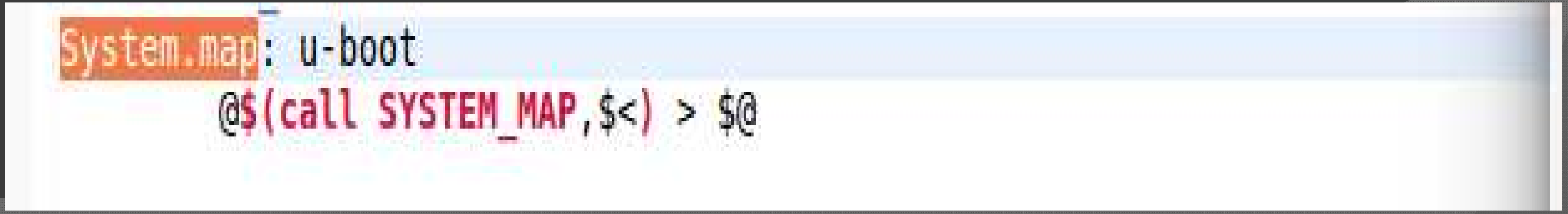
```
u-boot.hex u-boot.srec: u-boot FORCE
	$(call if_changed,objcopy)
```

- ✓ u-boot.bin ： u-boot



```
u-boot.bin: u-boot FORCE
	$(call if_changed,objcopy)
	$(call DO_STATIC_RELA,$<,$@,$(CONFIG_SYS_TEXT_BASE))
	$(BOARD_SIZE_CHECK)
```

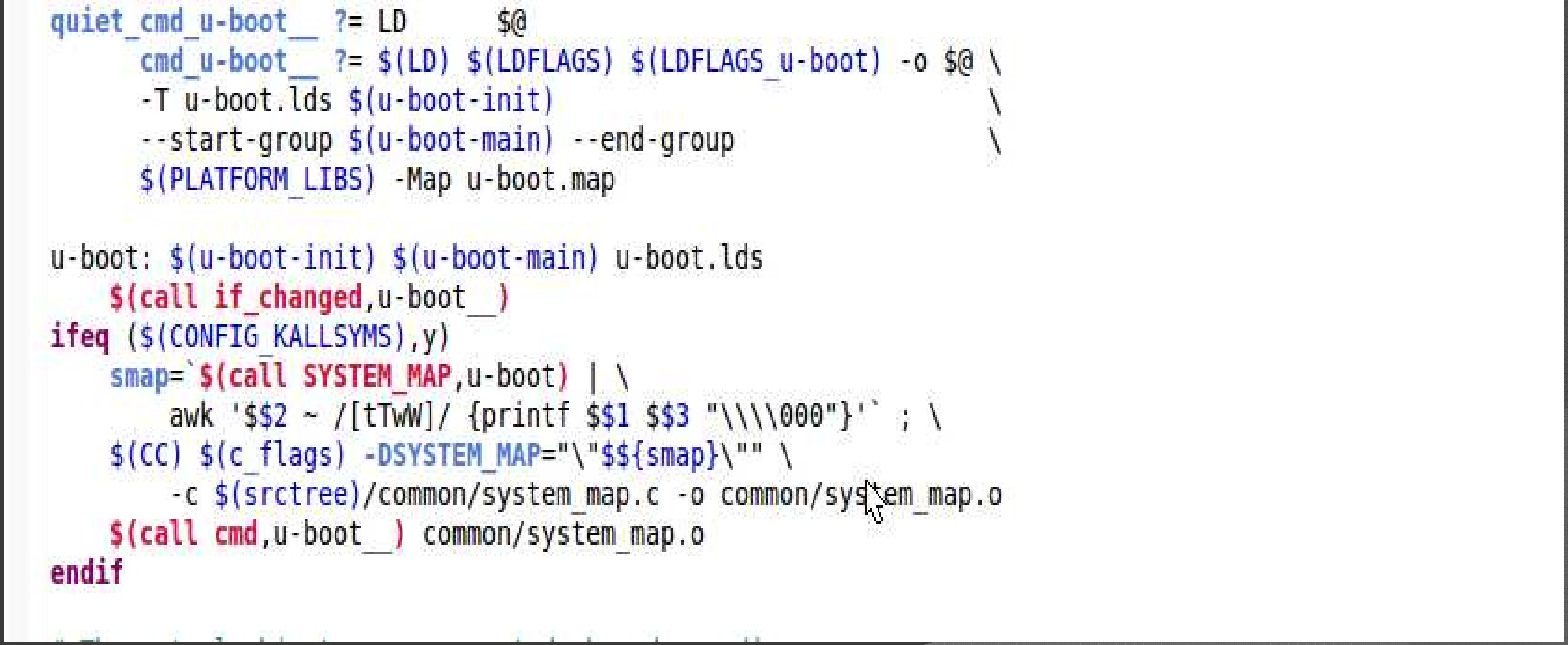
- ✓ System.map ： u-boot



```
System.map: u-boot
	@$(call SYSTEM_MAP,$<) > $@
```

➢ Uboot 编译过程-make all

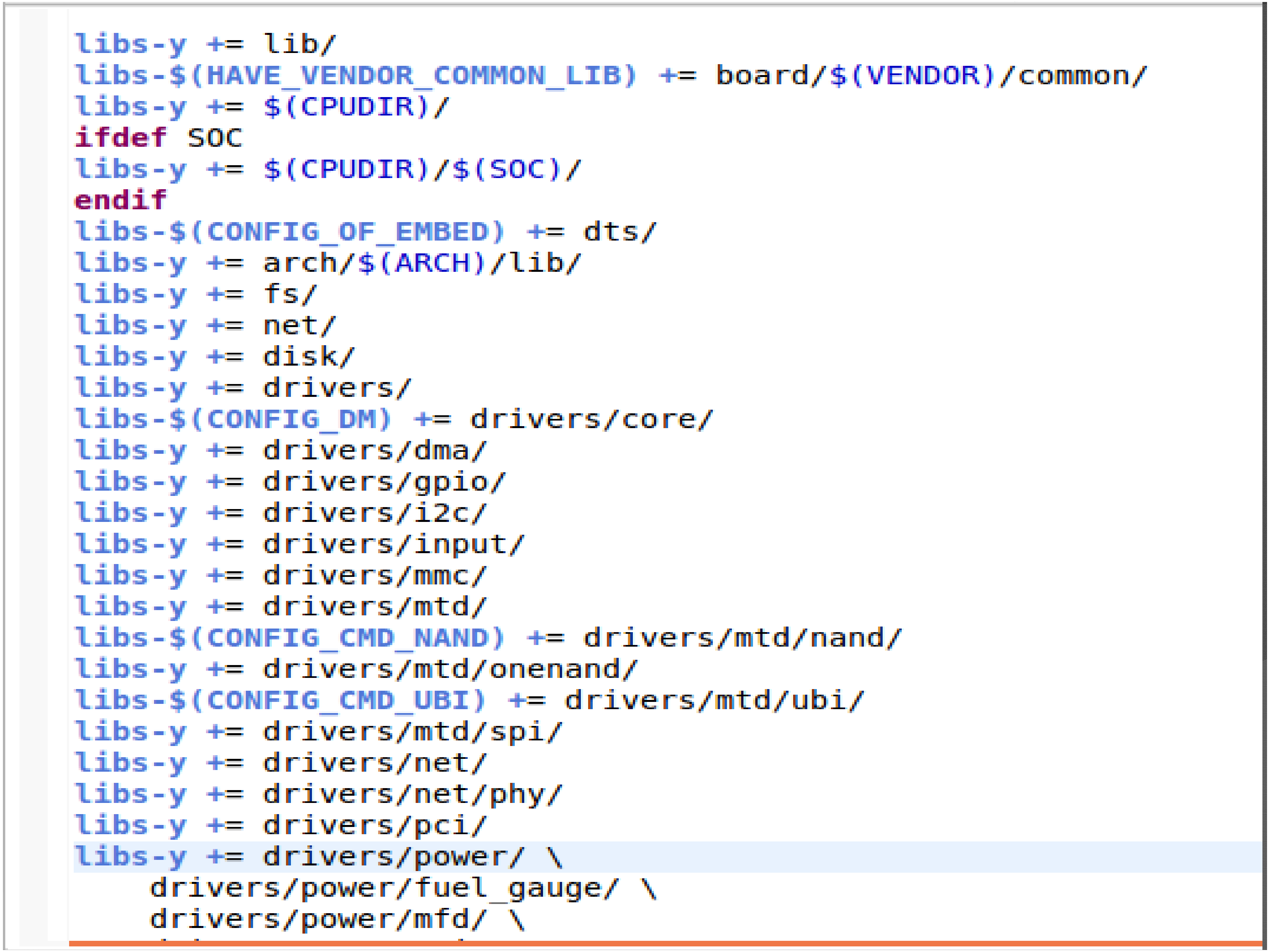
- u-boot:$(u-boot-init) $(u-boot-main) u-boot.lds



```
quiet_cmd_u-boot__ ?= LD      $@
      cmd_u-boot__ ?= $(LD) $(LDFLAGS) $(LDFLAGS_u-boot) -o $@ \
      -T u-boot.lds $(u-boot-init)                             \
      --start-group $(u-boot-main) --end-group                 \
      $(PLATFORM_LIBS) -Map u-boot.map

u-boot: $(u-boot-init) $(u-boot-main) u-boot.lds
    $(call if_changed,u-boot__)
ifeq ($(CONFIG_KALLSYMS),y)
    smap=`$(call SYSTEM_MAP,u-boot) | \
        awk '$$2 ~ /[tTwW]/ {printf $$1 $$3 "\\\\000"}'` ; \
    $(CC) $(c_flags) -DSYSTEM_MAP="\"$${smap}\"" \
        -c $(srctree)/common/system_map.c -o common/system_map.o
    $(call cmd,u-boot__) common/system_map.o
endif
```

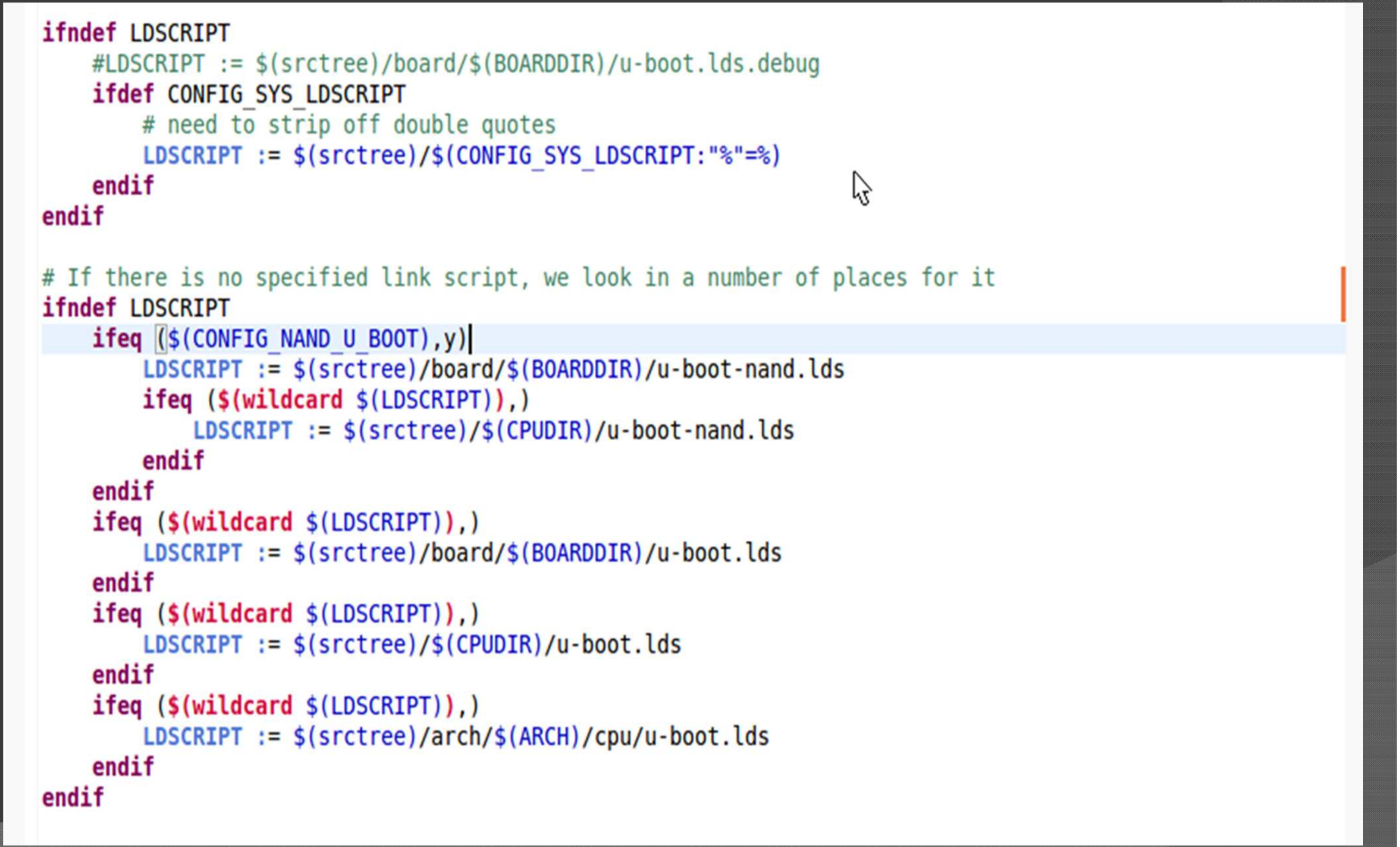
➢ Uboot 编译过程-make all

- ✓ u-boot-init := $(head-y)
- ✓ head-y := $(CPUDIR)/start.o
- ✓ u-boot-main := $(libs-y)
- ✓ u-boot.lds: $(LDSCRIPT) prepare FORCE
  $(call if_changed_dep,cpp_lds)



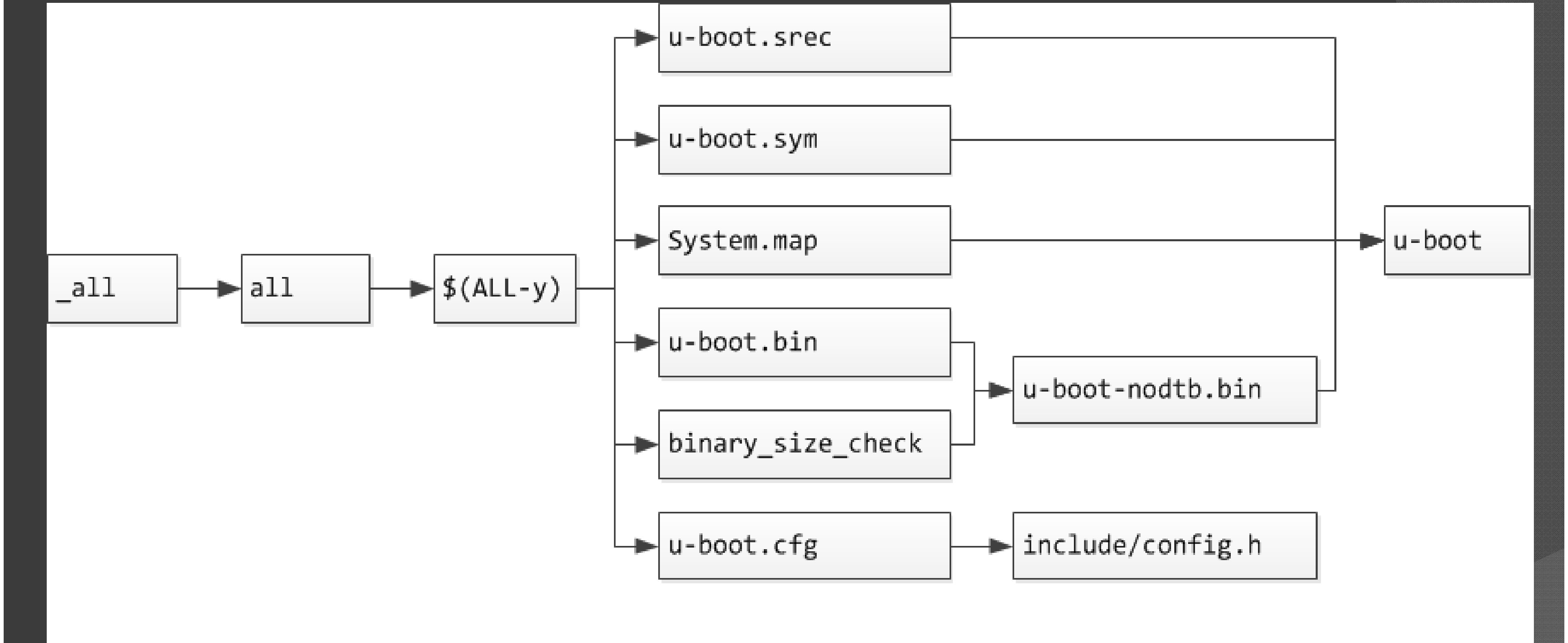
```
libs-y += lib/
libs-$(HAVE_VENDOR_COMMON_LIB) += board/$(VENDOR)/common/
libs-y += $(CPUDIR)/
ifdef SOC
libs-y += $(CPUDIR)/$(SOC)/
endif
libs-$(CONFIG_OF_EMBED) += dts/
libs-y += arch/$(ARCH)/lib/
libs-y += fs/
libs-y += net/
libs-y += disk/
libs-y += drivers/
libs-$(CONFIG_DM) += drivers/core/
libs-y += drivers/dma/
libs-y += drivers/gpio/
libs-y += drivers/i2c/
libs-y += drivers/input/
libs-y += drivers/mmc/
libs-y += drivers/mtd/
libs-$(CONFIG_CMD_NAND) += drivers/mtd/nand/
libs-y += drivers/mtd/onenand/
libs-$(CONFIG_CMD_UBI) += drivers/mtd/ubi/
libs-y += drivers/mtd/spi/
libs-y += drivers/net/
libs-y += drivers/net/phy/
libs-y += drivers/pci/
libs-y += drivers/power/ \
	drivers/power/fuel_gauge/ \
	drivers/power/mfd/ \
```

➢ Uboot 编译过程-make all



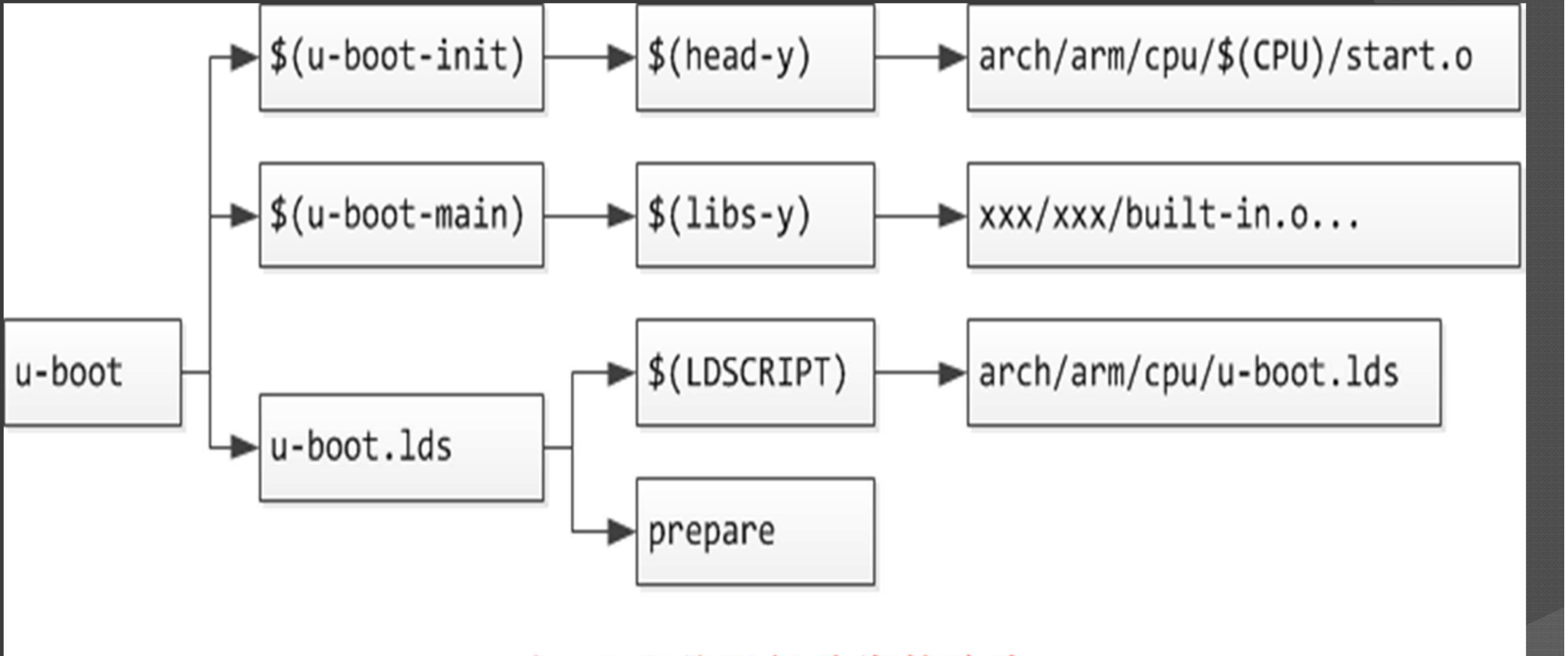
```
ifndef LDSCRIPT
    #LDSCRIPT := $(srctree)/board/$(BOARDDIR)/u-boot.lds.debug
    ifdef CONFIG_SYS_LDSCRIPT
        # need to strip off double quotes
        LDSCRIPT := $(srctree)/$(CONFIG_SYS_LDSCRIPT:"%"=%)
    endif
endif

# If there is no specified link script, we look in a number of places for it
ifndef LDSCRIPT
    ifeq ($(CONFIG_NAND_U_BOOT),y)
        LDSCRIPT := $(srctree)/board/$(BOARDDIR)/u-boot-nand.lds
        ifeq ($(wildcard $(LDSCRIPT)),)
            LDSCRIPT := $(srctree)/$(CPUDIR)/u-boot-nand.lds
        endif
    endif
    ifeq ($(wildcard $(LDSCRIPT)),)
        LDSCRIPT := $(srctree)/board/$(BOARDDIR)/u-boot.lds
    endif
    ifeq ($(wildcard $(LDSCRIPT)),)
        LDSCRIPT := $(srctree)/$(CPUDIR)/u-boot.lds
    endif
    ifeq ($(wildcard $(LDSCRIPT)),)
        LDSCRIPT := $(srctree)/arch/$(ARCH)/cpu/u-boot.lds
    endif
endif
```

➢ Uboot 编译过程-文件依赖关系



u-boot顶层目标的依赖关系

➢ Uboot 编译过程-文件依赖关系



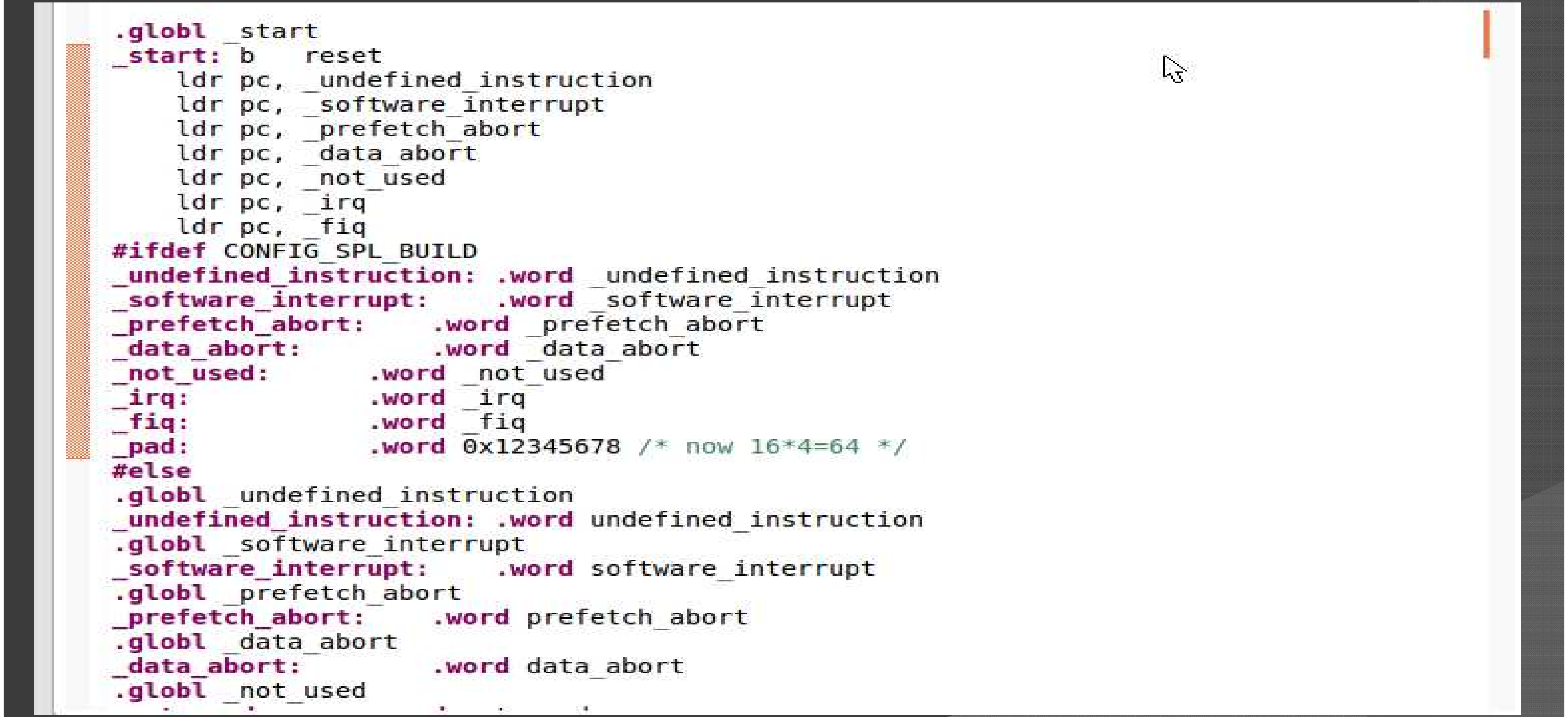
u-boot文件目标的依赖关系

➢ Uboot 启动流程分析

- _start,
- _main,
- board_init_f,
- relocate_code,
- board_init_r

➢ Uboot 启动流程分析-_start

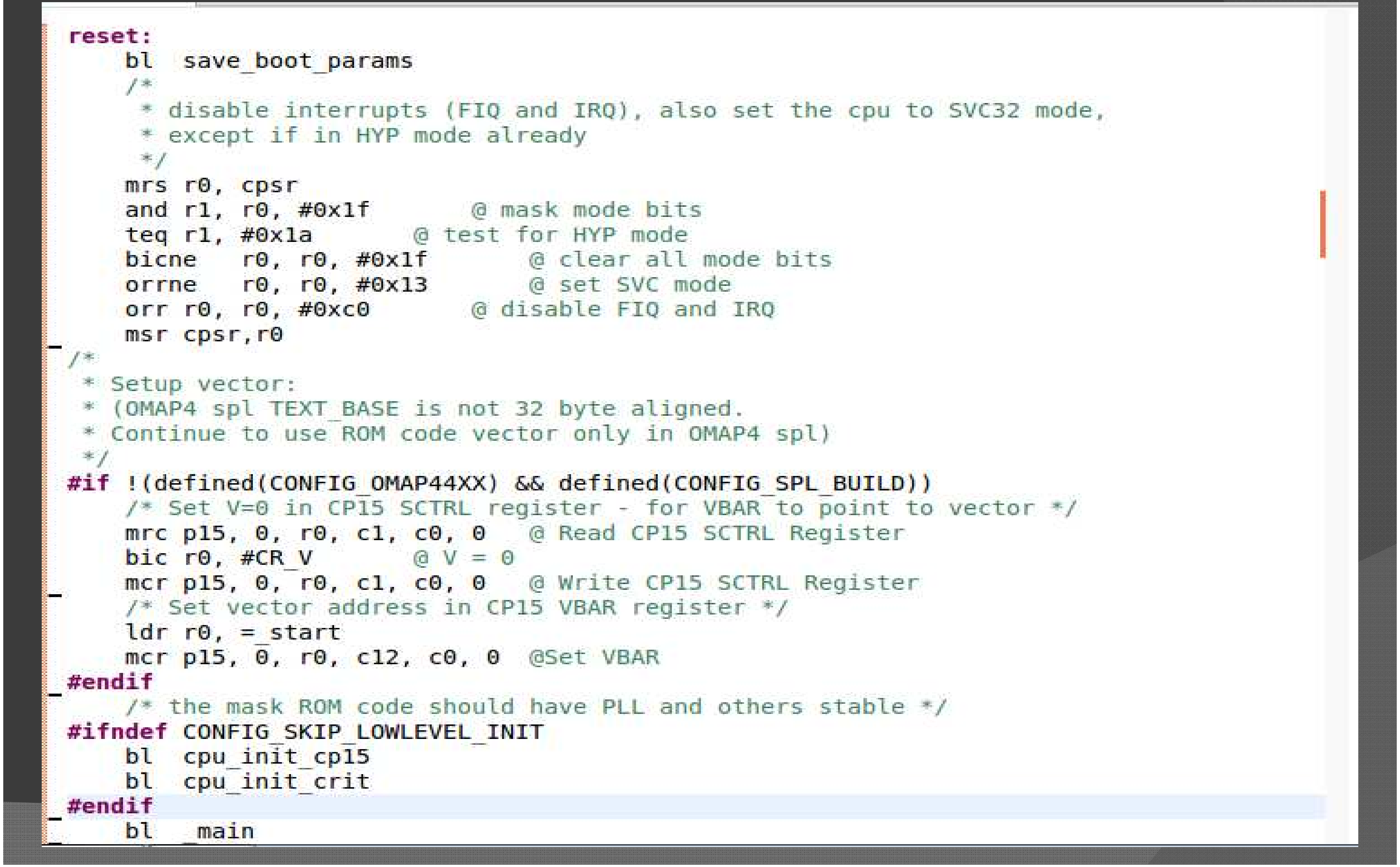
✓ 定义异常向量入口



```
.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
#ifdef CONFIG_SPL_BUILD
_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
_fiq:           .word _fiq
_pad:           .word 0x12345678 /* now 16*4=64 */
#else
.globl _undefined_instruction
_undefined_instruction: .word undefined_instruction
.globl _software_interrupt
_software_interrupt:    .word software_interrupt
.globl _prefetch_abort
_prefetch_abort:    .word prefetch_abort
.globl _data_abort
_data_abort:        .word data_abort
.globl _not_used
```

✓ reset向量 执行 b reset

➢ Uboot 启动流程分析-reset



```
reset:
    bl  save_boot_params
    /*
     * disable interrupts (FIQ and IRQ), also set the cpu to SVC32 mode,
     * except if in HYP mode already
     */
    mrs r0, cpsr
    and r1, r0, #0x1f       @ mask mode bits
    teq r1, #0x1a       @ test for HYP mode
    bicne   r0, r0, #0x1f       @ clear all mode bits
    orrne   r0, r0, #0x13       @ set SVC mode
    orr r0, r0, #0xc0       @ disable FIQ and IRQ
    msr cpsr,r0
/*
 * Setup vector:
 * (OMAP4 spl TEXT_BASE is not 32 byte aligned.
 * Continue to use ROM code vector only in OMAP4 spl)
 */
#if !(defined(CONFIG_OMAP44XX) && defined(CONFIG_SPL_BUILD))
    /* Set V=0 in CP15 SCTRL register - for VBAR to point to vector */
    mrc p15, 0, r0, c1, c0, 0   @ Read CP15 SCTRL Register
    bic r0, #CR_V       @ V = 0
    mcr p15, 0, r0, c1, c0, 0   @ Write CP15 SCTRL Register
    /* Set vector address in CP15 VBAR register */
    ldr r0, =_start
    mcr p15, 0, r0, c12, c0, 0  @Set VBAR
#endif
    /* the mask ROM code should have PLL and others stable */
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
    bl  cpu_init_cp15
    bl  cpu_init_crit
#endif
    bl  _main
```

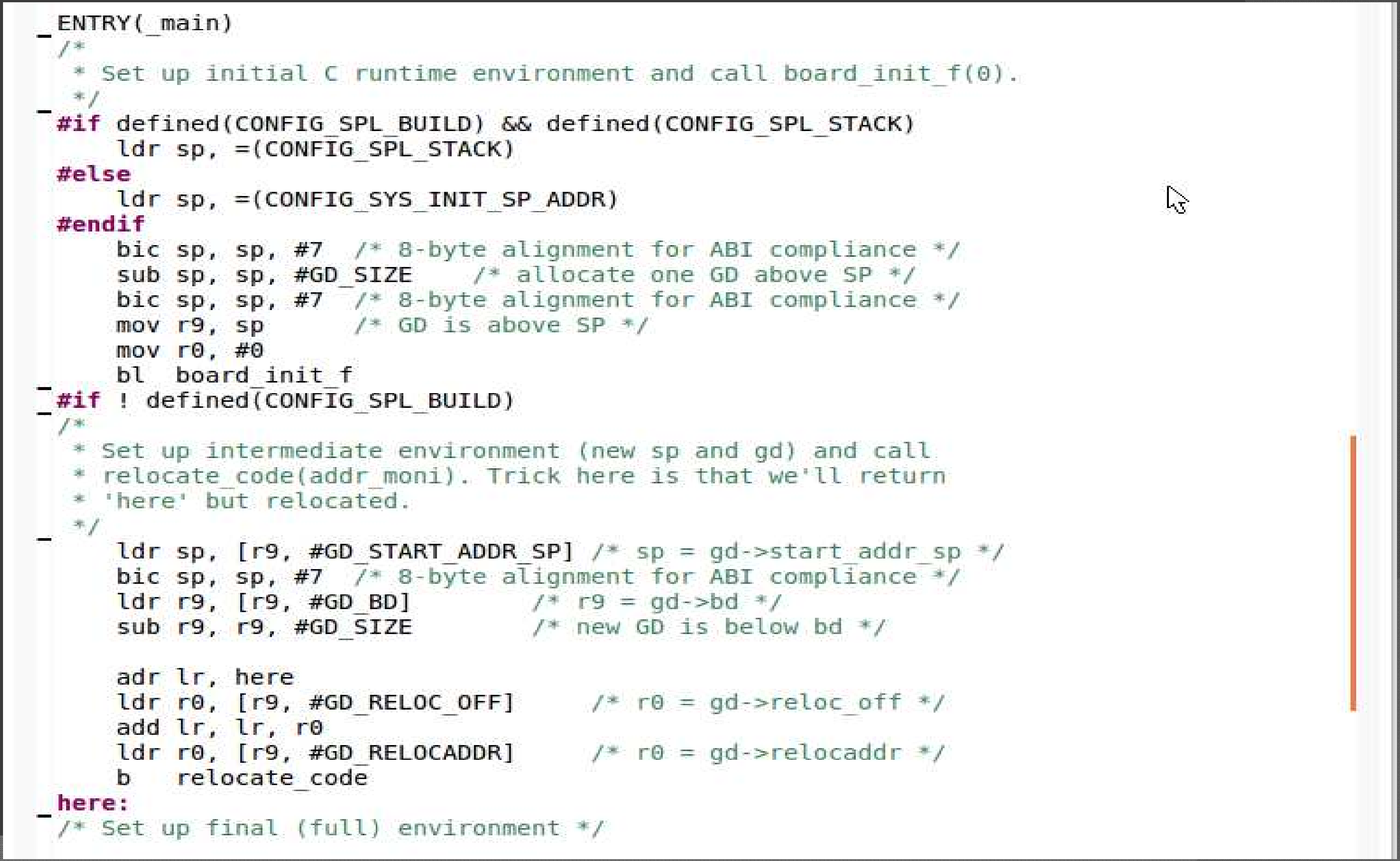
➢ Uboot 启动流程分析-reset

- 设置cpu进入svc32模式
- 关闭中断
- cpu_init_cp15

  设置cp15相关寄存器来设置处理器的MMU, cache以及tlb
- cpu_init_crit

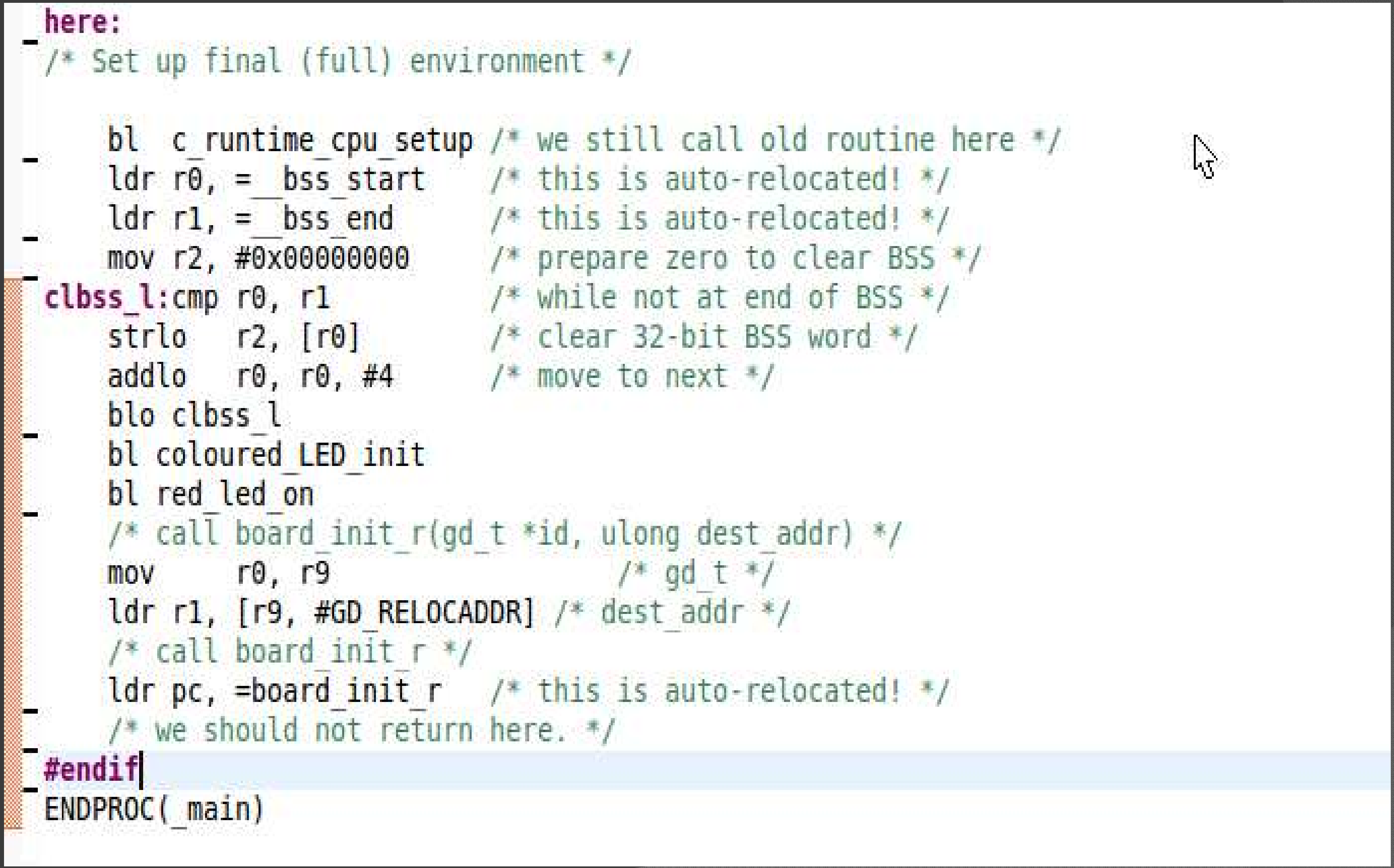
  lowlevel_init: 初始化 pll,memory

➢ Uboot 启动流程分析-_main



```
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)
#endif
    bic sp, sp, #7  /* 8-byte alignment for ABI compliance */
    sub sp, sp, #GD_SIZE    /* allocate one GD above SP */
    bic sp, sp, #7  /* 8-byte alignment for ABI compliance */
    mov r9, sp      /* GD is above SP */
    mov r0, #0
    bl  board_init_f
#if ! defined(CONFIG_SPL_BUILD)
/*
 * Set up intermediate environment (new sp and gd) and call
 * relocate_code(addr_moni). Trick here is that we'll return
 * 'here' but relocated.
 */
    ldr sp, [r9, #GD_START_ADDR_SP] /* sp = gd->start_addr_sp */
    bic sp, sp, #7  /* 8-byte alignment for ABI compliance */
    ldr r9, [r9, #GD_BD]        /* r9 = gd->bd */
    sub r9, r9, #GD_SIZE        /* new GD is below bd */

    adr lr, here
    ldr r0, [r9, #GD_RELOC_OFF]     /* r0 = gd->reloc_off */
    add lr, lr, r0
    ldr r0, [r9, #GD_RELOCADDR]     /* r0 = gd->relocaddr */
    b   relocate_code
here:
/* Set up final (full) environment */
```

➢ Uboot 启动流程分析-_main



```
here:
/* Set up final (full) environment */

    bl  c_runtime_cpu_setup /* we still call old routine here */
    ldr r0, =__bss_start    /* this is auto-relocated! */
    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 */
    strlo   r2, [r0]        /* clear 32-bit BSS word */
    addlo   r0, r0, #4      /* move to next */
    blo clbss_l
    bl coloured_LED_init
    bl red_led_on
    /* 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 */
    ldr pc, =board_init_r   /* this is auto-relocated! */
    /* we should not return here. */
#endif
ENDPROC(_main)
```

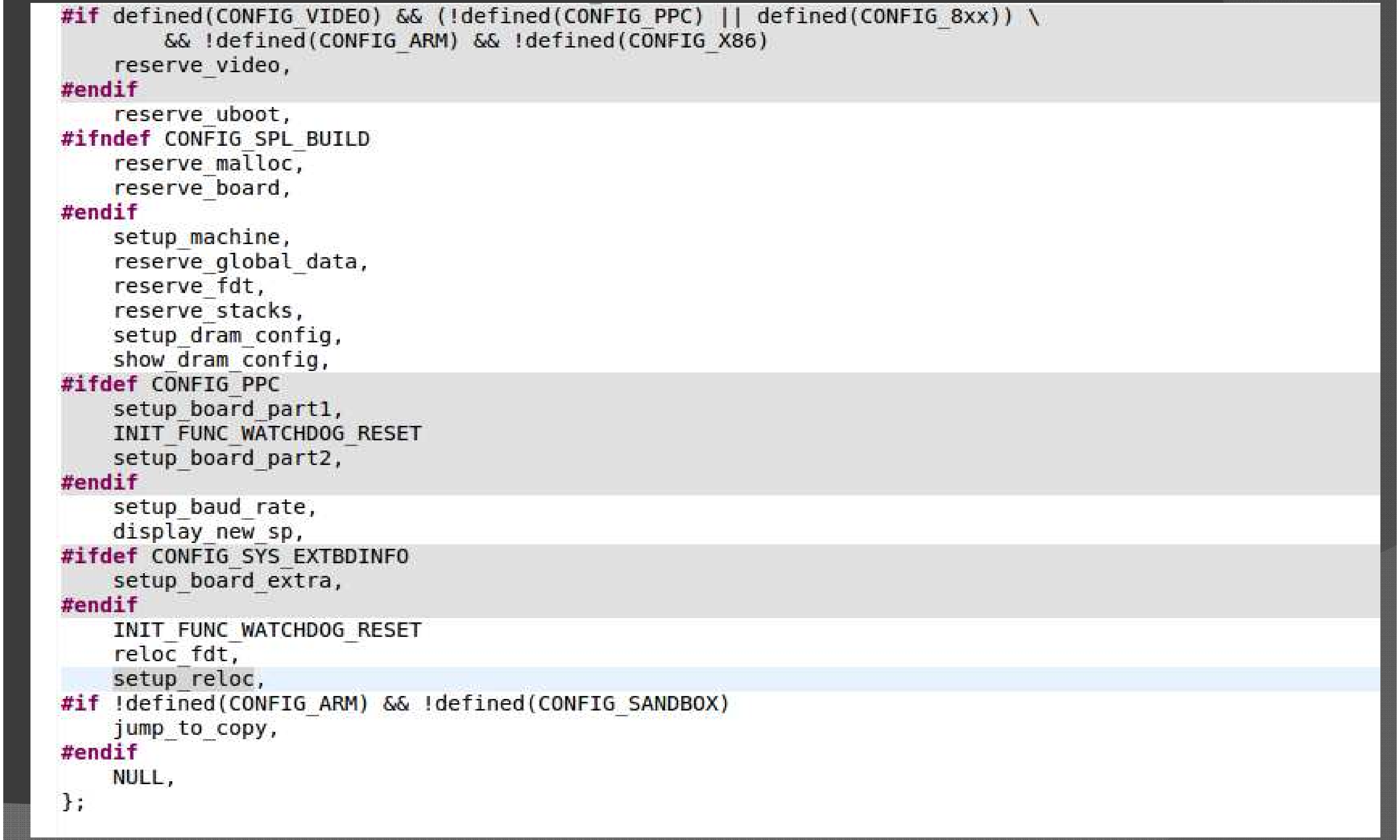
➢ Uboot 启动流程分析-_main

- 设置栈指针sp
- board_init_f : init_sequence_f
- relocate_code
- board_init_r : init_sequence_r

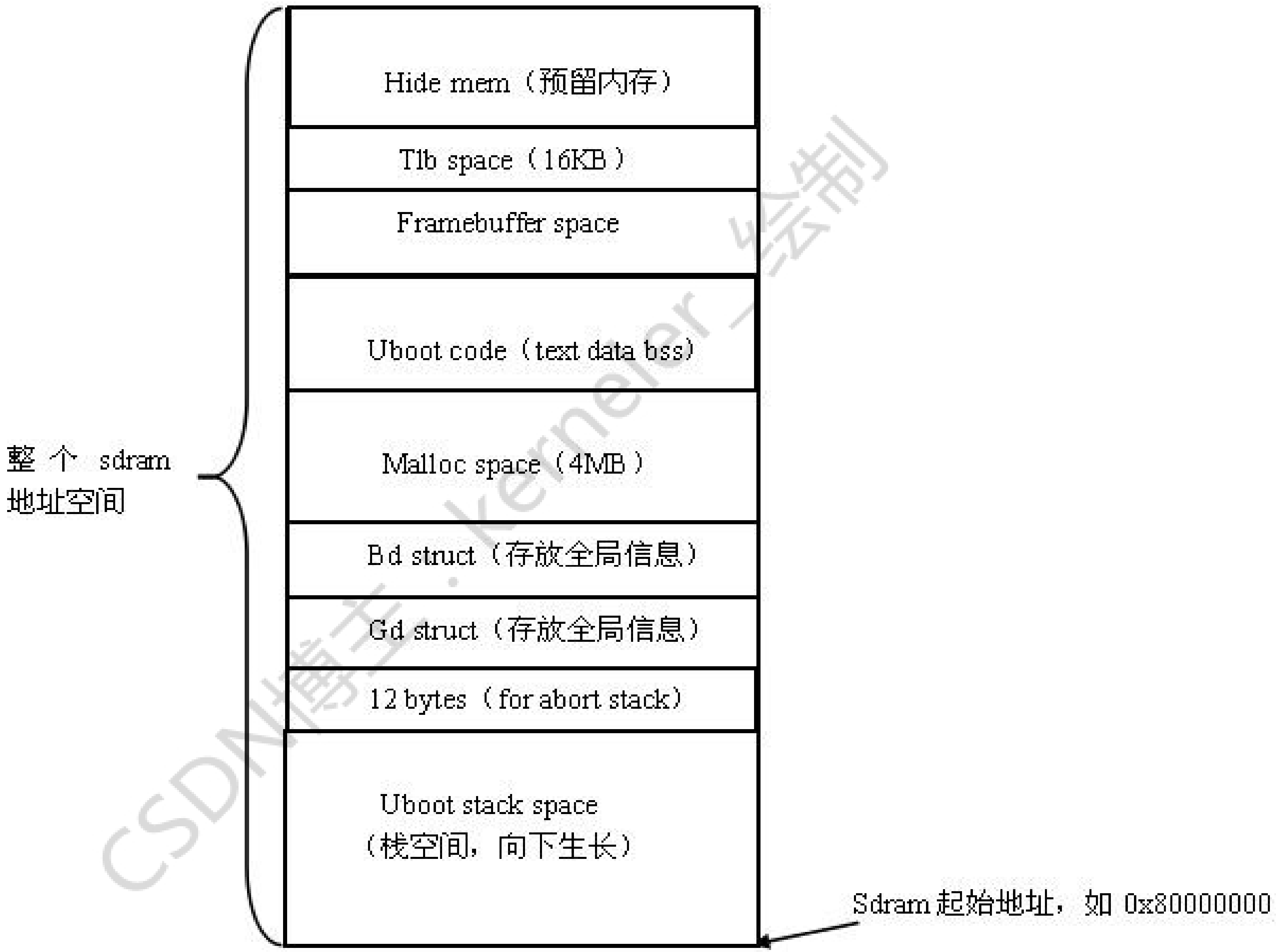
➢ Uboot 启动流程分析- board_init_f

- setup_mon_len:获取uboot code长度(初始化gd->mon_len)
- setup_fdt : 获取fdt地址 (初始化gd->fdt_blob)
- mark_bootstage : 记录启动阶段
- env_init : 初始化环境来源 (gd->env_addr,gd->env_valid)
- init_baud_rate:从环境变量中获取baudrate (gd->baudrate)
- serial_init : 串口初始化
- console_init_f: (gd->have_console=1)
- display_options:打印uboot版本信息
- display_text_info:打印调试信息 (CONFIG_SYS_TEXT_BASE,bss_start,bss_end)
- print_cpuinfo:打印cpu相关信息
- dram_init : dram初始化 (gd->ram_size)

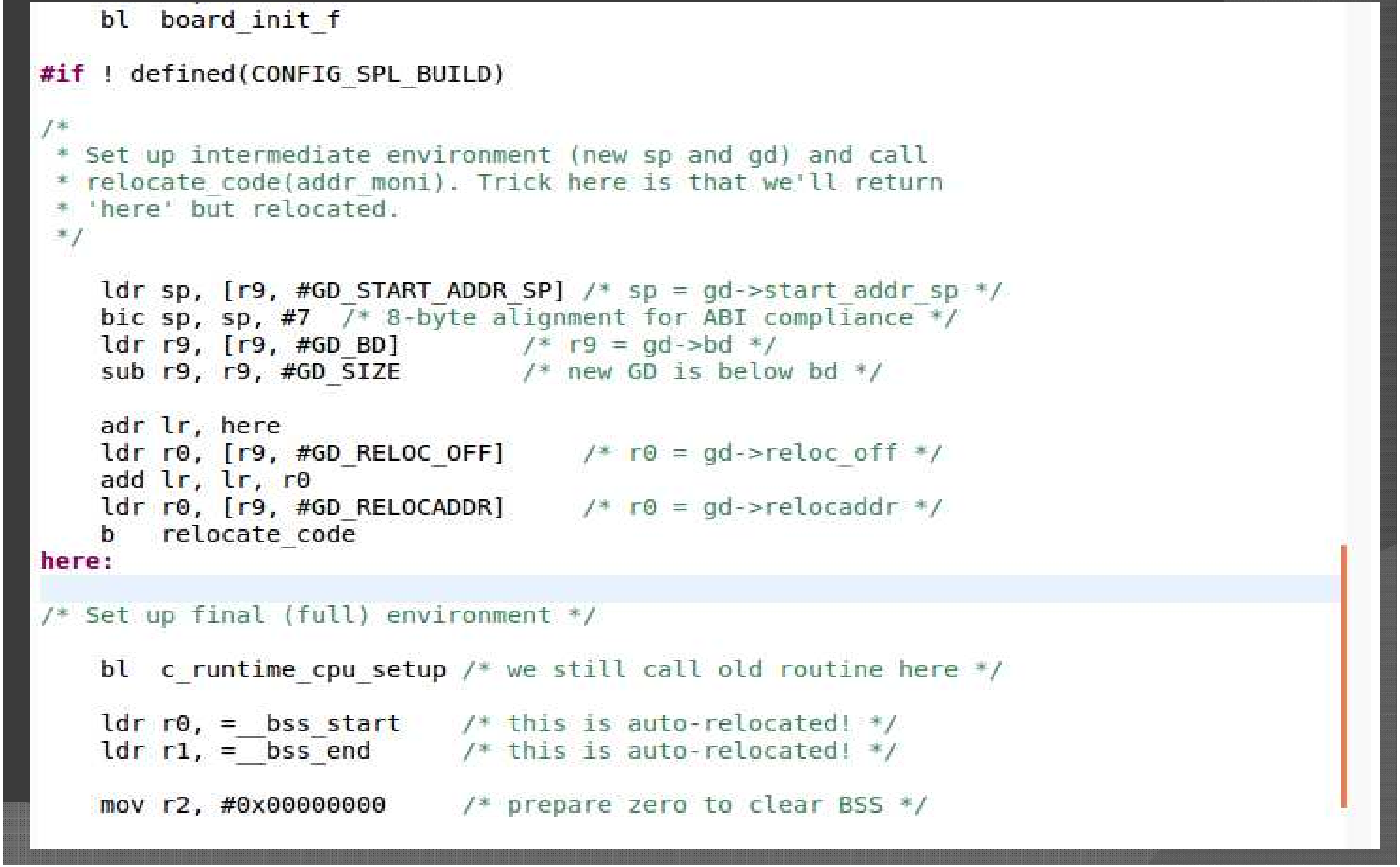
➢ Uboot 启动流程分析- board_init_f



```
#if defined(CONFIG_VIDEO) && (!defined(CONFIG_PPC) || defined(CONFIG_8xx)) \
        && !defined(CONFIG_ARM) && !defined(CONFIG_X86)
    reserve_video,
#endif
    reserve_uboot,
#ifndef CONFIG_SPL_BUILD
    reserve_malloc,
    reserve_board,
#endif
    setup_machine,
    reserve_global_data,
    reserve_fdt,
    reserve_stacks,
    setup_dram_config,
    show_dram_config,
#ifdef CONFIG_PPC
    setup_board_part1,
    INIT_FUNC_WATCHDOG_RESET
    setup_board_part2,
#endif
    setup_baud_rate,
    display_new_sp,
#ifdef CONFIG_SYS_EXTBDINFO
    setup_board_extra,
#endif
    INIT_FUNC_WATCHDOG_RESET
    reloc_fdt,
    setup_reloc,
#if !defined(CONFIG_ARM) && !defined(CONFIG_SANDBOX)
    jump_to_copy,
#endif
    NULL,
};
```


Hide mem（预留内存）
Tlb space（16KB）
Framebuffer space
Uboot code（text data bss）
Malloc space（4MB）
Bd struct（存放全局信息）
Gd struct（存放全局信息）
12 bytes（for abort stack）
Uboot stack space
（栈空间，向下生长）
整 个 sdram
地址空间
Sdram 起始地址，如 0x80000000

➢ Uboot 启动流程分析- relocate_code

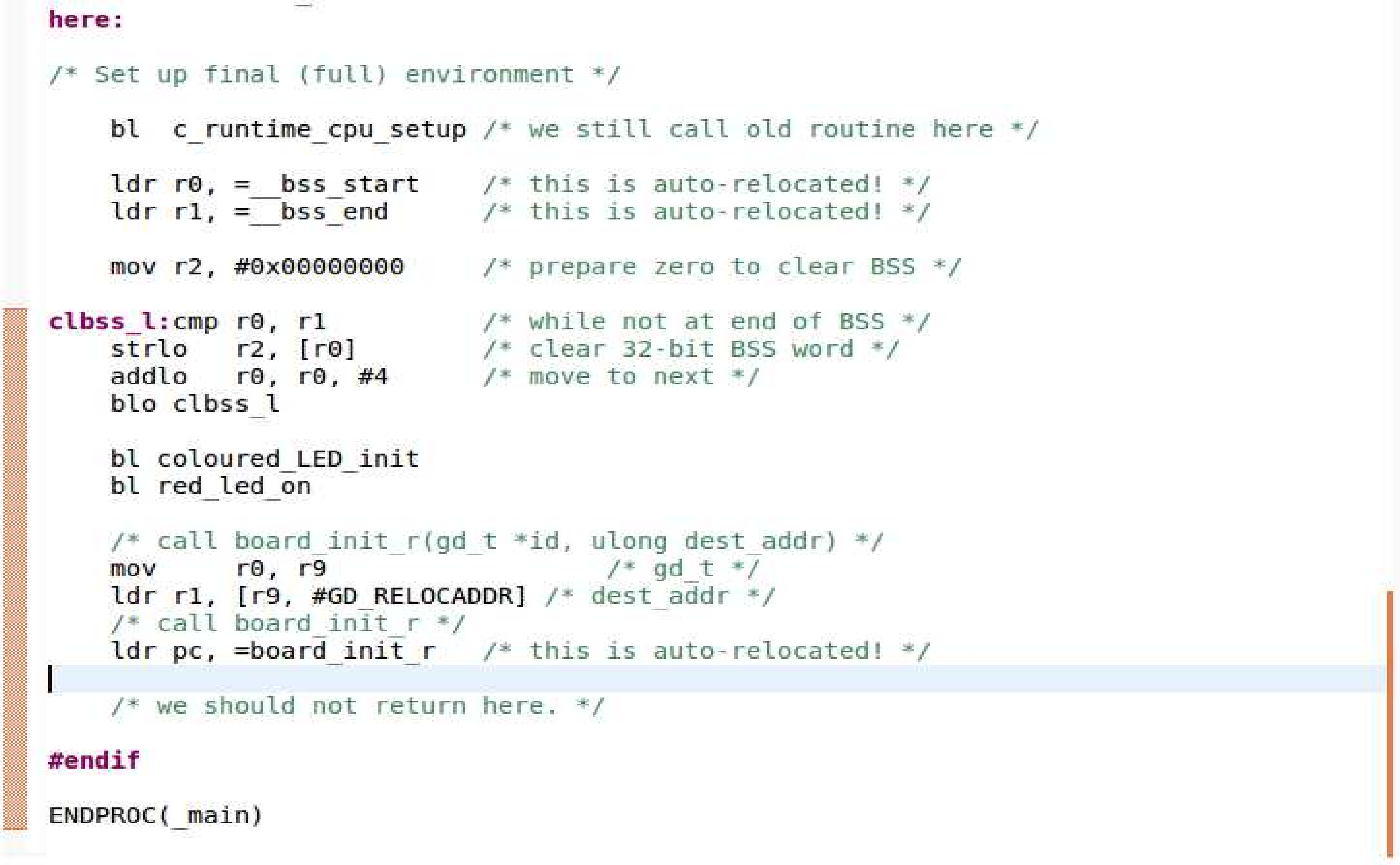


```
    bl  board_init_f

#if ! defined(CONFIG_SPL_BUILD)

/*
 * Set up intermediate environment (new sp and gd) and call
 * relocate_code(addr_moni). Trick here is that we'll return
 * 'here' but relocated.
 */

    ldr sp, [r9, #GD_START_ADDR_SP] /* sp = gd->start_addr_sp */
    bic sp, sp, #7  /* 8-byte alignment for ABI compliance */
    ldr r9, [r9, #GD_BD]          /* r9 = gd->bd */
    sub r9, r9, #GD_SIZE          /* new GD is below bd */

    adr lr, here
    ldr r0, [r9, #GD_RELOC_OFF]       /* r0 = gd->reloc_off */
    add lr, lr, r0
    ldr r0, [r9, #GD_RELOCADDR]       /* r0 = gd->relocaddr */
    b   relocate_code
here:

/* Set up final (full) environment */

    bl  c_runtime_cpu_setup /* we still call old routine here */

    ldr r0, =__bss_start    /* this is auto-relocated! */
    ldr r1, =__bss_end      /* this is auto-relocated! */

    mov r2, #0x00000000     /* prepare zero to clear BSS */
```

➢ Uboot 启动流程分析- relocate_code

- 更新栈指针sp
- 更新全局结构体gd地址
- 保存here搬移后在ram中的位置
- relocate_code
- relocate_code后进入ram空间

➢ Uboot 启动流程分析- board_init_r



```
here:

/* Set up final (full) environment */

    bl  c_runtime_cpu_setup /* we still call old routine here */

    ldr r0, =__bss_start    /* this is auto-relocated! */
    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 */
    strlo   r2, [r0]        /* clear 32-bit BSS word */
    addlo   r0, r0, #4      /* move to next */
    blo clbss_l

    bl coloured_LED_init
    bl red_led_on

    /* 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 */
    ldr pc, =board_init_r   /* this is auto-relocated! */

    /* we should not return here. */

#endif

ENDPROC(_main)
```

➢ Uboot 启动流程分析- board_init_r

- initr_reloc:设置gd->flag,记录启动阶段
- set_cpu_clk_info:设置时钟相关的操作
- initr_serial: 注册当前串口设备到全局串口链表中(将串口更新到ram)
- initr_malloc:初始化 malloc 区域(清零)
- set_cpu_clk_info:设置cpu时钟信息
- stdio_init:将所有串口注册到stdio设备中
- initr_jumptable : 设置gd->jt
- console_init_r: 设置标准输入输出和出错设备
- run_main_loop:进入uboot命令模式

谢谢观赏！