



# **Melis4.0 RTOS 系统开发指南**

**版本号: 0.1**  
**发布日期: 2021.05.12**

## 版本历史

版本号	日期	制/修订人	内容描述
0.1	2021.05.12	PDC-PD1	创建



# 目 录

<b>1 前言</b>	<b>1</b>
1.1 编写目的 . . . . .	1
1.2 使用范围 . . . . .	1
1.3 相关人员 . . . . .	1
<b>2 系统组件</b>	<b>2</b>
<b>3 内核启动</b>	<b>4</b>
3.1 head_s.S . . . . .	4
3.2 init.c . . . . .	4
<b>4 异常处理</b>	<b>7</b>
<b>5 操作系统 OS</b>	<b>8</b>
<b>6 驱动层</b>	<b>9</b>
<b>7 文件系统</b>	<b>10</b>
<b>8 subsys 子系统</b>	<b>11</b>
<b>9 legacy 系统封装</b>	<b>12</b>

# 1 前言

---

## 1.1 编写目的

介绍 melis 4.0 的开发环境和系统架构。

## 1.2 使用范围

Allwinner 软件平台 Melis4.0 以上版本。

## 1.3 相关人员

使用 Melis4.0 以上版本进行产品开发的开发者。

## 2 系统组件



图 2-1: kernel component

系统泛指内核，如上图所示包含了各种组件，软件将在启动运行过程中，逐步对各组件进行初始化及调用。内核代码主要在 sdk 的 source/ekernel 文件夹。

```
ekernel
├── arch
├── core
├── drivers
├── filesystem
├── legacy
└── subsys
```

- arch：存放架构相关的代码；

- core：存放 os 组件的相关代码；
- drivers：存放驱动 hal 层相关的代码；
- filesystem：存放设备管理器、分区管理器、文件系统相关的代码；
- legacy：存放系统接口，给 emodules、应用等非内核模块调用的函数接口；
- subsys：存放 finsh\_cli 控制台、adb 等第三方组件相关的代码；



## 3 内核启动

内核启动在架构文件夹`ekernel/arch/riscv/`，架构目录结构如下图所示；

```
riscv
├── atomic
├── common
├── csp
├── lds
├── plic
├── rv64gc
├── sbi
├── standby
└── sunxi
```

### 3.1 head\_s.S

首先看 `riscv/rv64gc/head_s.S` 文件，主要完成了如下功能：

- bss 段的清零；
- sp 栈指针的初始化；
- mmu 初始化和页表基地址赋值；
- 异常统一入口赋值；
- 跳转至 `start_kernel` 函数；

```
221    la    t0, start_kernel
```

### 3.2 init.c

`start_kernel`在`riscv/sunxi/init.c`文件中实现，开始以 C 语言为主要编程语言对各组件进行初始化。本段主要介绍一下初始化过程中调用的`do_initcalls()`函数。

`do_initcalls()`函数通过层层调用，调用了`initcall_levels`数组定义的函数地址标识，这些地址标识在`riscv/lds/kernel.lds`中定义，表示代码段名称为`initcallxx.init`类的代码，C 文件通过`__attribute__((__section__("text")))`，指定函数或变量在链接时存放的代码段名称。这个由`source/include/melis/init.h`文件中的宏定义`__define_initcall(fn, id, .initcall##id)`来声明实现。

例如：subsys\_initcall(drv\_dma\_init);将函数指定到代码段.initcall4.init，将会在调用\_\_initcall4\_start时，把\_\_initcall4\_start到\_\_initcall5\_start代码段地址区间的函数接口执行一遍。

```
static initcall_entry_t *initcall_levels[] =
{
    __initcall0_start,
    __initcall1_start,
    __initcall2_start,
    __initcall3_start,
    __initcall4_start,
    __initcall5_start,
    __initcall6_start,
    __initcall7_start,
    __initcall_end,
};
```

```
.dram_seg.initcall ADDR(.dram_seg.rodata) + SIZEOF(.dram_seg.rodata) : AT(LOADADDR(.
dram_seg.rodata) + SIZEOF(.dram_seg.rodata))
{
    . = ALIGN(8);
    __initcall_start = .;
    KEEP(*(.initcallearly.init))
    __initcall0_start = .;
    KEEP(*(.initcall0.init))
    KEEP(*(.initcall0s.init))
    __initcall1_start = .;
    KEEP(*(.initcall1.init))
    KEEP(*(.initcall1s.init))
    __initcall2_start = .;
    KEEP(*(.initcall2.init))
    KEEP(*(.initcall2s.init))
    __initcall3_start = .;
    KEEP(*(.initcall3.init))
    KEEP(*(.initcall3s.init))
    __initcall4_start = .;
    KEEP(*(.initcall4.init))
    KEEP(*(.initcall4s.init))
    __initcall5_start = .;
    KEEP(*(.initcall5.init))
    KEEP(*(.initcall5s.init))
    __initcallrootfs_start = .;
    KEEP(*(.initcallrootfs.init))
    KEEP(*(.initcallrootfss.init))
    __initcall6_start = .;
    KEEP(*(.initcall6.init))
    KEEP(*(.initcall6s.init))
    __initcall7_start = .;
    KEEP(*(.initcall7.init))
    KEEP(*(.initcall7s.init))
    __initcall_end = .;
    __con_initcall_start = .;
    KEEP(*(.con_initcall.init))
    __con_initcall_end = .;
    . = ALIGN(8);
} > DRAM_SEG_KRN :rodata =0x9002
```



```
#define __define_initcall(fn, id, __sec) \
    static initcall_t __initcall_##fn##id __used \
        __attribute__((__section__(#__sec ".init"))) = fn;

#define __define_initcall(fn, id) __define_initcall(fn, id, .initcall##id)
#define pure_initcall(fn) __define_initcall(fn, 0)
#define core_initcall(fn) __define_initcall(fn, 1)
#define core_initcall_sync(fn) __define_initcall(fn, 1s)
#define postcore_initcall(fn) __define_initcall(fn, 2)
#define postcore_initcall_sync(fn) __define_initcall(fn, 2s)
#define arch_initcall(fn) __define_initcall(fn, 3)
#define arch_initcall_sync(fn) __define_initcall(fn, 3s)
#define subsys_initcall(fn) __define_initcall(fn, 4)
#define subsys_initcall_sync(fn) __define_initcall(fn, 4s)
#define fs_initcall(fn) __define_initcall(fn, 5)
#define fs_initcall_sync(fn) __define_initcall(fn, 5s)
#define rootfs_initcall(fn) __define_initcall(fn, rootfs)
#define device_initcall(fn) __define_initcall(fn, 6)
#define device_initcall_sync(fn) __define_initcall(fn, 6s)
#define late_initcall(fn) __define_initcall(fn, 7)
#define late_initcall_sync(fn) __define_initcall(fn, 7s)
#define __initcall(fn) device_initcall(fn)
#define __exitcall(fn) \
    static exitcall_t __exitcall_##fn __exit_call = fn

#define console_initcall(fn) __define_initcall(fn, , .con_initcall)
```

## 4 异常处理

异常处理是针对异常事件 `exception` 和中断事件 `interrupt` 的处理，`ekernel/arch/riscv/rv64gc/exception_s.S` 文件的 `handle_exception` 为总入口函数，根据 `scause` 的状态为由 `riscv_cpu_handle_interrupt` 和 `riscv_cpu_handle_exception` 分发至中断处理函数和异常处理函数。

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	<i>Reserved for future standard use</i>
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt <sup>③</sup>
1	6	<i>Reserved for future standard use</i>
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	<i>Reserved for future standard use</i>
1	11	Machine external interrupt
1	12–15	<i>Reserved for future standard use</i>
1	≥16	<i>Reserved for platform use</i>
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	<i>Reserved</i>
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	<i>Reserved for future standard use</i>
0	15	Store/AMO page fault
0	16–23	<i>Reserved for future standard use</i>
0	24–31	<i>Reserved for custom use</i>
0	32–47	<i>Reserved for future standard use</i>
0	48–63	<i>Reserved for custom use</i>
0	≥64	<i>Reserved for future standard use</i>

图 4-1: mcausereg

## 5 操作系统 OS

操作系统使用的 rt-thread，其源代码主要在 ekernel/core/rt-thread 文件夹，包含线程创建 (thread.c)、系统调度 (scheduler.c)、信号量互斥量 (ipc.c)、临界区保护 (irq.c)、内存管理 (slab.c)、设备管理 (device.c)、系统软件定时器 (timer.c)、系统时钟节拍 (clock.c) 等组件。

```
core
├─ clock.c
├─ cmsis
├─ completion.c
├─ components.c
├─ cpu.c
├─ device.c
├─ idle.c
├─ include
├─ ipc.c
├─ irq.c
├─ J-RTT
├─ Kconfig
├─ Kconfig.scheduler
├─ kservice.c
├─ libdl
├─ Makefile
├─ mem.c
├─ memheap.c
├─ mempool.c
├─ object.c
├─ openocd.c
├─ perf.c
├─ pipe.c
├─ README.md
├─ README_zh.md
├─ ringbuffer.c
├─ scheduler.c
├─ signal.c
├─ slab.c
├─ thread.c
├─ timer.c
├─ waitqueue.c
├─ workqueue.c
└─ wrapper
```

## 6 驱动层

驱动层代码在`ekernel/drivers`文件夹，包含了 `hal` 层侧重于对 SOC 寄存器等的操作，`drv` 则对 `hal` 层的函数进行格式封装以便调用。`osal` 则是驱动需要调用到操作系统相关的函数时，为增强代码可移植性进行的一层封装层，`test` 包含了针对各个驱动模块编写的测试代码。

```
drivers
├── drv
├── hal
├── include
├── Makefile
├── osal
└── test
```

## 7 文件系统

文件系统相关代码在ekernel/filesystem文件夹，依据设备注册到文件系统识别的过程，包含了设备管理器、分区管理器、文件系统管理等组件；

```
filesystem
├─ dbg
├─ devman
├─ doc
├─ fs
├─ fsys_libs.c
├─ include
├─ Kconfig
├─ Makefile
├─ part
└─ sys_fsys.c
```

## 8 subsys 子系统

subsys 子系统相关代码在ekernel/subsys文件夹，将各第三方组件的扩展容纳进内核系统。

```
subsys
├─ adbd
├─ archival
├─ avframework
├─ aw
├─ benchmark
├─ bluetooth
├─ config
├─ cplusplus
├─ debug
├─ env
├─ finsh_cli
├─ Kconfig
├─ lib
├─ libc
├─ Makefile
├─ net
├─ pthreads
├─ samples
├─ standby
└─ thirdparty
```

## 9 legacy 系统封装

ekernel/legacy 文件夹主要存放系统函数接口，将系统内部的函数进行一层封装，提供给到系统以外的驱动模块或应用等模块调用；

```
legacy
├─ az100
├─ bmp
├─ charset
├─ clock
├─ dma
├─ dramc
├─ encrypt
├─ gpio
├─ hid
├─ input
├─ interrupt
├─ Kconfig
├─ Makefile
├─ mod_api
├─ mod_install
├─ modstub
├─ powerman
├─ resman
├─ script_config
├─ service
├─ sios
└─ sys_time
```

## 著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明

、 全志科技、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。