文档编号：AN2016

上海东软载波微电子有限公司

微型操作系统架构及 API

基于 Cortex-M0

修订历史

版本	修订日期	修改概要
V1.0	2018-11-14	初版

地　址：中国上海市龙漕路 299 号天华信息科技园 2A 楼 5 层
邮　编：200235
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com/

目 录

前言 **5**
第 1 章 概述 **6**
1.1 主要功能 6
1.2 软件框架 6
1.3 文件结构 7
第 2 章 任务管理模块 **8**
2.1 功能概述 8
2.2 函数接口 8
2.2.1 创建 task 8
2.2.2 暂停 task 8
2.2.3 恢复 task 8
2.2.4 删除 task 9
第 3 章 内存管理模块 **10**
3.1 功能概述 10
3.2 函数接口 10
3.2.1 申请内存 10
3.2.2 释放内存 10
第 4 章 定时器管理模块 **11**
4.1 功能概述 11
4.2 函数接口 11
4.2.1 创建 timer 11
4.2.2 修改 timer 11
4.2.3 删除 timer 12
4.2.4 启动 timer 12
4.2.5 停止 timer 12
4.2.6 获取 ke 的 tick 12
第 5 章 消息队列管理模块 **13**
5.1 功能概述 13
5.2 函数接口 13
5.2.1 创建 queue 13
5.2.2 删除 queue 13
5.2.3 向 queue 添加消息 13
5.2.4 从 queue 取出消息 14
5.2.5 获取指定 queue 内消息的个数 14
5.2.6 初始化普通消息队列(msgq) 14
5.2.7 向普通消息队列(msgq)添加消息 14
5.2.8 从普通消息队列(msgq)取出消息 15
第 6 章 信号量管理模块 **16**
6.1 功能概述 16
6.2 函数接口 16
6.2.1 创建 sem 16
6.2.2 删除 sem 16

6.2.3 发送信号量 16
6.2.4 等待信号量 17
第 7 章 命令行模块 18
7.1 功能概述 18
7.2 函数接口 18
7.2.1 添加命令函数 18
7.2.2 修改密码函数 18
7.2.3 打印函数 18
第 8 章 RB-Tree 模块 19
8.1 功能概述 19
8.2 函数接口 19
8.2.1 初始化 rbtree 19
8.2.2 插入节点 19
8.2.3 查找节点 19
8.2.4 获取节点个数 20
第 9 章 Kernel 接口 21
9.1 功能概述 21
9.2 函数接口 21
9.2.1 初始化 kernel 21
9.2.2 启动 kernel 21
9.2.3 显示 kernel 信息 21
9.2.4 IDLE 任务处理函数 21
9.2.5 10ms 中断服务函数 22

前言

本文档主要介绍基于 Cortex-M0 的微型操作系统架构及其对外接口。Micro-kernel 是一款非抢占微内核，为应用层提供多任务环境。

第1章 概述

本篇文档，主要对微型操作系统框架进行简单介绍，并告知以后的使用者如何使用这套框架的接口。

1.1 主要功能

该操作系统主要功能有：

任务管理

内存管理

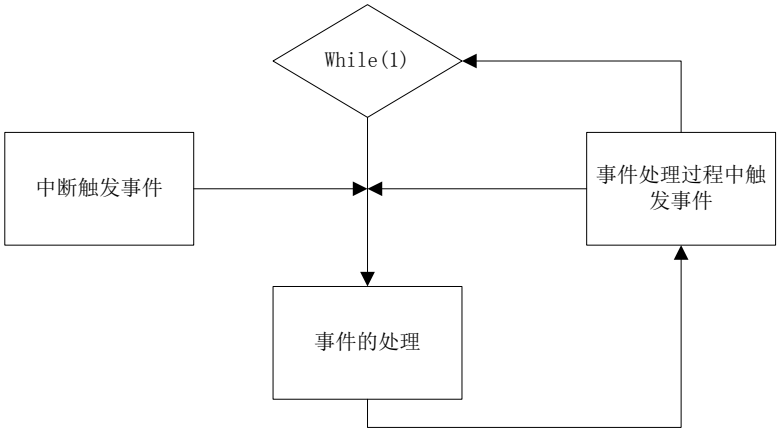
定时器管理

消息队列

信号量管理

命令行管理

1.2 软件框架



1.1 内核框架

主程序是一个 while(1)无限循环，循环体中不停的检查任务标志是否被置起，如果置起，执行任务的处理函数，否则进行下一任务标志检查。其中任务标志的置起有两个途径，一个是中断，另外一个是任务处理函数。

如果在该任务被处理之前，多次置起标志位，系统只认为是一次置起，任务的处理过程并不会被打断（中断之后，仍然会回到中断前的任务处理），因此，任务的处理过程不能持续太长时间，如果一个任务并非时间紧急、而处理过程太长，那么建议将该任务优先级调低、整个处理过程分多次执行。比如，任务执行一部分之后，保存执行结果，再将本标志位置起，结束执行。（系统下次会选择优先级高的任务处理）。

定时器也是一个特定的事件，其与任务也有关系，实际上是定时触发指定任务。

内存管理，是将系统中未使用的内存作为内存池，内存的申请与释放即是在内存池中进行操作。

消息队列是指缓存消息的队列。使用方式为在任务执行函数中读取消息队列内的消息，在中断处理程序或其他任务处理函数中将消息放入消息队列。

命令行模块，提供命令行支持，方便 Debug 问题，加快开发进度。

红黑树模块，提供红黑树的使用接口

1. 3 文件结构

ke.c 操作系统中比较基础的函数，入口函数等；
ke.h 对外接口均在此文件内；
ke_que.c 队列管理系列函数
ke_task.c 任务管理系列函数；
ke_mem.c 内存管理系列函数；
ke_sem.c 信号量系列管理函数；
ke_shell.c 命令行管理函数；
ke_tmr.c 软件定时器系列管理函数。
rbtree.[ch] 红黑树接口

第2章 任务管理模块

2. 1 功能概述

最大支持 32 个 task；
支持优先级功能，任务调度时高优先级 task 优先执行；
同一优先级只能创建一个 task，所以 task 的优先级即为 task 的 id；
系统不会强制终止运行中的 task，需要 task 的处理函数自觉适时退出；

2. 2 函数接口

2. 2. 1 创建task

类型	描述
函数原型	int task_create(task_cbk_t func, void *arg, int prio);
功能描述	创建一个 task
输入参数	func:task 的处理函数(入口函数)
	arg: 处理函数的参数
	prio:要创建 task 的优先级
返回值	0 - task 创建成功
	-1 - task 创建失败
使用示例	task_create (func0, NULL, 2);

表 2-1 创建 task 函数

2. 2. 2 暂停task

类型	描述
函数原型	int task_suspend(int prio);
功能描述	暂停 task 运行
输入参数	prio:要暂停 task 的优先级
返回值	0 - task 暂停成功
	-1 - task 暂停失败
使用示例	task_ suspend (2);

表 2-2 暂停 task 函数

2. 2. 3 恢复task

类型	描述
函数原型	int task_resume(int prio);
功能描述	恢复 task 运行
输入参数	prio:要恢复 task 的优先级
返回值	0 - task 恢复成功
	-1 - task 恢复失败
使用示例	task_ resume(2);

表 2-3 恢复 task 函数

2. 2. 4 删除task

类型	描述
函数原型	int task_delete(int prio);
功能描述	删除 task
输入参数	prio:要删除 task 的优先级
返回值	0 - task 删除成功
	-1 - task 删除失败
使用示例	task_delete(2);

表 2-4 删除 task 函数

第3章 内存管理模块

3.1 功能概述

使用系统中未使用的内存作为内存池；
申请到的内存应及时释放；
用户可以动态查看当前内存池可用内存(需 shell 支持)。

3.2 函数接口

3.2.1 申请内存

类型	描述
函数原型	void *ke_malloc(uint32_t size);
功能描述	申请一段连续的内存
输入参数	待申请的内存长度(字节个数)
返回值	成功：申请到的内存首指针
	失败：NULL
使用示例	uint8_t *p = (uint8_t *)ke_malloc(32);

表 3-1 申请内存函数

3.2.2 释放内存

类型	描述
函数原型	void ke_free(void *p);
功能描述	释放内存
输入参数	待释放内存的首指针
返回值	无
使用示例	ke_free(p);

表 3-2 释放内存函数

第4章 定时器管理模块

4.1 功能概述

定时器定时精度为 10ms；
支持单次定时器和周期性定时器；
定时器的回调函数在主线程中执行，不占用中断资源。

4.2 函数接口

4.2.1 创建timer

类型	描述
函数原型	ke_tmr_t *tmr_create(uint32_t expire, uint32_t period, timer_cbk_t func, void *arg);
功能描述	创建一个定时器
输入参数	expire: 超时时间
	period: 周期性定时器周期值，0 表示非周期性定时器
	func: 定时器超时回调函数
	arg: 回调函数参数
返回值	成功：申请到的 ke_tmr_t 结构体指针
	失败：NULL
使用示例	ke_tmr_t *tmr = tmr_create(32, 12, func, NULL);

表 4-1 创建 timer 函数

4.2.2 修改timer

类型	描述
函数原型	int tmr_modify(ke_tmr_t *tmr, uint32_t expire, uint32_t period, tmr_cbk_t func, void *arg);
功能描述	修改一个已存在的空闲定时器，运行态定时器不能修改。
输入参数	tmr: 指向定时器的指针
	expire: 超时时间
	period: 周期性定时器周期值，0 表示非周期性定时器
	func: 定时器超时回调函数
	arg: 回调函数参数
返回值	成功：返回 0
	失败：返回-1
使用示例	ke_tmr_t *tmr = tmr_create(32, 12, func, NULL);

表 4-2 修改 timer 函数

4. 2. 3 删除timer

类型	描述
函数原型	void tmr_delete(ke_tmr_t *tmr);
功能描述	删除一个指定的定时器，运行态定时器也可以删除。
输入参数	tmr: 定时器结构体指针
返回值	无
使用示例	tmr_delete(tmr);

表 4-3 删除 timer 函数

4. 2. 4 启动timer

类型	描述
函数原型	void tmr_start(ke_tmr_t *tmr);
功能描述	启动定时器
输入参数	tmr: 定时器结构体指针
返回值	无
使用示例	tmr_ start(tmr);

表 4-4 启动 timer 函数

4. 2. 5 停止timer

类型	描述
函数原型	void tmr_stop(ke_tmr_t *tmr);
功能描述	停止定时器
输入参数	tmr: 定时器结构体指针
返回值	无
使用示例	tmr_stop(tmr);

表 4-5 停止 timer 函数

4. 2. 6 获取ke的tick

类型	描述
函数原型	uint32_t get_kernel_tick(void);
功能描述	获取 kernel 的 tick，每个 tick 为 10ms
输入参数	无
返回值	当前 tick 数
使用示例	uint32_t tick = get_kernel_tick();

表 4-6 获取 kernel 当前 tick 函数

第5章 消息队列管理模块

5.1 功能概述

队列即为先进先出向数据结构；

为 task 和 task 之间传递消息提供通道；为中断处理程序向 task 传递消息提供通道；

典型应用为：在 task 处理函数中从消息队列中读取消息，其他 task 处理函数或者中断处理函数向消息队列添加消息。

提供 2 种类型的消息队列：

1. 关联目标 task，该目标 task 一直从队列中取消息，若取不到则将目标 task 自动挂起(suspend)，当中断服务程序或其他 task 向队列中添加消息时，则自动将该目标 task 唤醒(resume)。该类型消息队列的 API 如 5.2.1--5.2.5 所示；
2. 普通消息队列，不关联目标 task。该类型消息队列的 API 如 5.2.6--5.2.8 所示。

5.2 函数接口

5.2.1 创建queue

类型	描述
函数原型	ke_que_t *que_create(int prio);
功能描述	创建一个消息队列
输入参数	prio：task 的优先级，创建的 queue 属与该 task
返回值	成功：申请到的 ke_que_t 结构体指针
	失败：NULL
使用示例	ke_que_t *que = que_create(2);

表 5-1 创建 queue 函数

5.2.2 删除queue

类型	描述
函数原型	void que_delete(ke_que_t *que);
功能描述	删除一个消息队列
输入参数	que：待删除的消息队列
返回值	无
使用示例	que_delete(que);

表 5-2 删除 queue 函数

5.2.3 向queue添加消息

类型	描述
函数原型	void que_enq(ke_que_t *que, ke_msg_t *msg);
功能描述	向消息队列中添加一个消息
输入参数	que：消息队列指针
	msg：待添加的消息
返回值	无
使用示例	que_enq(que, msg);

表 5-3 向 queue 中添加消息函数

5. 2. 4 从queue取出消息

类型	描述
函数原型	ke_msg_t *que_deq(ke_que_t *que);
功能描述	从消息队列中取出一个消息
输入参数	que: 消息队列指针
返回值	当队列中有消息时，返回最先存入的消息
	当队列中没有消息时，返回 NULL
使用示例	if ((msg = que_deq(que)) == NULL) return; /* do something */ return;

表 5-4 从 queue 中取出消息函数

5. 2. 5 获取指定queue内消息的个数

类型	描述
函数原型	uint32_t que_get_number(ke_que_t *que);
功能描述	获取指定队列内消息的个数
输入参数	que: 消息队列指针
返回值	指定队列内消息的个数
使用示例	uint32_t nr = que_get_number(que);

表 5-5 获取指定队列内消息的个数

5. 2. 6 初始化普通消息队列(msgq)

类型	描述
函数原型	void msgq_init(ke_msgq_t *msgq);
功能描述	初始化普通消息队列 msgq
输入参数	msgq: 待初始化的消息队列指针
返回值	无
使用示例	msgq_init(msgq);

表 5-6 初始化普通消息队列

5. 2. 7 向普通消息队列(msgq)添加消息

类型	描述
函数原型	void msgq_enq(ke_msgq_t *msgq, ke_msg_t *msg);
功能描述	向普通消息队列添加消息
输入参数	msgq: 消息队列指针
	msg: 待添加的消息
返回值	无
使用示例	msgq_enq(que, msg);

表 5-7 向普通消息队列添加消息

5. 2. 8 从普通消息队列(msgq)取出消息

类型	描述
函数原型	ke_msg_t *msgq_deq(ke_msgq_t *msgq);
功能描述	从普通消息队列取出消息
输入参数	msgq: 消息队列指针
返回值	当队列中有消息时，返回最先存入的消息
	当队列中没有消息时，返回 NULL
使用示例	msg = msgq_deq(msgq);

表 5-8 从普通消息队列取出消息

第6章 信号量管理模块

6. 1 功能概述

为 task 和 task 之间进行事件同步提供信号机制；

为中断处理程序向 task 发送通知提供通道；

典型应用为：在 task 处理函数中等待信号，其他 task 处理函数或者中断处理函数向指定 task 发送信号。

信号量机制与消息队列机制的差别为，信号量机制不能携带消息，只能发送信号，用于事件通知。

6. 2 函数接口

6. 2. 1 创建sem

类型	描述
函数原型	ke_sem_t *sem_create(int prio);
功能描述	创建一个信号量
输入参数	prio：task 的优先级，创建的 sem 属与该 task
返回值	成功：申请到的 ke_sem_t 结构体指针
	失败：NULL
使用示例	ke_sem_t *sem = sem_create(2)

表 6-1 创建 sem 函数

6. 2. 2 删除sem

类型	描述
函数原型	void sem_delete(ke_sem_t *sem);
功能描述	删除一个信号量
输入参数	sem：待删除的信号量
返回值	无
使用示例	sem_delete(sem);

表 6-2 删除 sem 函数

6. 2. 3 发送信号量

类型	描述
函数原型	void sem_post(ke_sem_t *sem);
功能描述	向信号量所属的 task 发送信号量
输入参数	sem：信号量指针
返回值	无
使用示例	sem_post(sem);

表 6-3 发送 sem 函数

6. 2. 4 等待信号量

类型	描述
函数原型	int sem_pend(ke_sem_t *sem);
功能描述	task 等待信号量
输入参数	sem: 信号量
返回值	当有信号量时，返回 0
	当没有信号量时，返回 1
使用示例	if (sem_pend(sem)) return; /* do something */ return;

表 6-4 等待 sem 函数

第7章 命令行模块

7.1 功能概述

为 kernel 提供命令行功能，方便 DEBUG 问题，加快开发进度；

命令长度最大支持 15 个字符，可以使用大小写字符，数字，"-" 和"_"；

支持"TAB"功能键，当前匹配的不止一个命令时，将匹配的所有命令都显示出来，当匹配的命令只有一个时，自动补全命令；

支持"UP"，"DOWN"功能键，可以使用这两个键调出历史命令，最大记录 3 条命令。

支持用户登录功能，只有输入正确的密码时才能激活 shell 命令行。系统默认密码为"eastsoft"；

支持密码修改功能，客户可以设计自己专有密码；

SHELL 模块内置"exit"命令，此命令退出当前登录状态。

7.2 函数接口

7.2.1 添加命令函数

类型	描述
函数原型	int32_t shell_cmd_insert(char *cmd, void *func, uint32_t nr_arg);
功能描述	向 shell 模块添加一条命令
输入参数	cmd: 命令的名称
	func: 命令对应的回调函数
	nr_arg: 回调函数的参数个数
返回值	成功：0
	失败：-1
使用示例	shell_cmd_insert("version", cmd_version, 2);

表 7-1 添加命令函数

7.2.2 修改密码函数

类型	描述
函数原型	void shell_modify_password(char *pw);
功能描述	修改 shell 登录密码
输入参数	pw: 新密码
返回值	无
使用示例	shell_modify_password("abcdef");

表 7-2 打印函数

7.2.3 打印函数

类型	描述
函数原型	void printf_s(const char *fmt, ...);
功能描述	打印输出，与标准 printf()函数用法一致
输入参数	fmt: 可变参数，
返回值	无
使用示例	printf_s("name: %s, age: %d\n", str, age);

表 7-3 打印函数

第8章 RB-Tree模块

8. 1 功能概述

提供红黑树的插入、查找功能。

8. 2 函数接口

8. 2. 1 初始化rbtree

类型	描述
函数原型	void rb_tree_init(rb_tree_t *tree, rb_node_cmp cmp);
功能描述	初始化一个 rbtree
输入参数	tree: 待初始化的红黑树;
	cmp: 函数指针，指向元素比较函数
返回值	无
使用示例	rb_tree_init(tree, cmp);

表 8-1 初始化 rbtree 函数

8. 2. 2 插入节点

类型	描述
函数原型	void rb_tree_insert(rb_tree_t *tree, rb_node_t *node, void *key);
功能描述	向 rbtree 中插入节点
输入参数	tree: 红黑树
	node: 待插入的节点
	key: 关键字，以此在红黑树中排序
返回值	无
使用示例	rb_tree_insert(tree, node, key);

表 8-2 插入节点函数

8. 2. 3 查找节点

类型	描述
函数原型	rb_node_t *rb_tree_find(rb_tree_t *tree, void *key);
功能描述	根据关键字，在红黑树中查找指定节点
输入参数	tree: 红黑树
	key: 关键字，以此进行查找
返回值	指定的节点，若找不到，则返回 NULL
使用示例	rb_node_t *node = rb_tree_find(tree, key);

表 8-3 查找节点函数

8. 2. 4 获取节点个数

类型	描述
函数原型	uint32_t rb_tree_get_size(rb_tree_t *tree);
功能描述	获取树中节点个数
输入参数	tree: 红黑树
返回值	节点个数
使用示例	uint32_t num = rb_tree_get_size(tree);

表 8-4 获取节点个数函数

第9章 Kernel接口

9.1 功能概述

外部使用 kernel 时的接口，包括初始化 kernel、启动 kernel、获取 kernel 内信息。

9.2 函数接口

9.2.1 初始化kernel

类型	描述
函数原型	void ke_init(uint32_t ram_size, uart_handle_t *hperh);
功能描述	初始化 kernel
输入参数	ram_size: 当前芯片的内存大小;
	hperh: 指向 uart_handle_t 结构体指针，用于 shell 模块
返回值	无
使用示例	ke_init(0x8000, &h_uart);

表 9-1 初始化 kernel 函数

9.2.2 启动kernel

类型	描述
函数原型	void ke_start(void);
功能描述	启动 kernel，此后系统控制器交给 kernel
输入参数	无
返回值	无
使用示例	ke_start();

表 9-2 启动 kernel 函数

9.2.3 显示kernel信息

类型	描述
函数原型	void ke_show(void);
功能描述	显示 kernel 信息，此函数依赖 shell 模块
输入参数	无
返回值	无
使用示例	ke_show();

表 9-3 信息 kernel 信息函数

9.2.4 IDLE任务处理函数

类型	描述
函数原型	void idle_func(void *arg);
功能描述	此函数可被应用层重载，用于低功耗处理
输入参数	arg: 任务参数
返回值	无
使用示例	idle_func(void *arg); 在应用层重载此函数

表 9-4 IDLE 任务处理函数

9. 2. 5 10ms中断服务函数

类型	描述
函数原型	void ke_10ms_irq_cbk(void);
功能描述	10ms 中断服务函数，此函数可被应用层重载
输入参数	无
返回值	无
使用示例	ke_10ms_irq_cbk (); 在应用层重载此函数

表 9-5 10ms 中断服务函数