嵌入式实时操作系统

μCOS -II

一、RTOS基础

实时操作系统特点

- 任务管理
  - 多任务和基于优先级的任务调度
- 任务间同步和通信
  - 消息、事件和信号量
- 定时器管理
  - 提供确定的任务切换时间
- 中断管理
- 存储管理

何时使用实时操作系统

- 完全取决于软件项目的复杂程度
  - 并行的功能模块比较多
  - 定时处理的功能比较多
  - 程序的执行需要判断很多条件参数或资源
  - 需要规划代码执行的优先顺序
  - 要保证多个模块的执行时间

实时操作系统的选择

- 内核占用存储区尺寸
- RTOS性能：任务切换时间、调度任务数和优先级数
- 软件组件和设备驱动程序的完备程度
- 开发调试工具易用性
- 标准兼容性，是否支持POSIX标准
- RTOS发送形式，源代码或二进制代码
- 许可证发送形式，开发许可和生产许可

二、μCOS-II介绍

μCOS-II简介

- μCOS -II 是 美国 一个名为Jean Labrosse的工程师开发的实时操作系统。它以小内核、多任务、 丰富的系统服务、容易使用等特点越来越受欢迎

- μCOS -II 实时系统的商业应用非常广泛，具有非常稳定、可靠的性能，成功应用于生命科学、航天工程等重大科研项目中。由于其极小的内核，特别适用于对程序代码存储空间要求极其敏感的嵌入式系统开发

- μOS-II是一款源码公开的实时操作系统

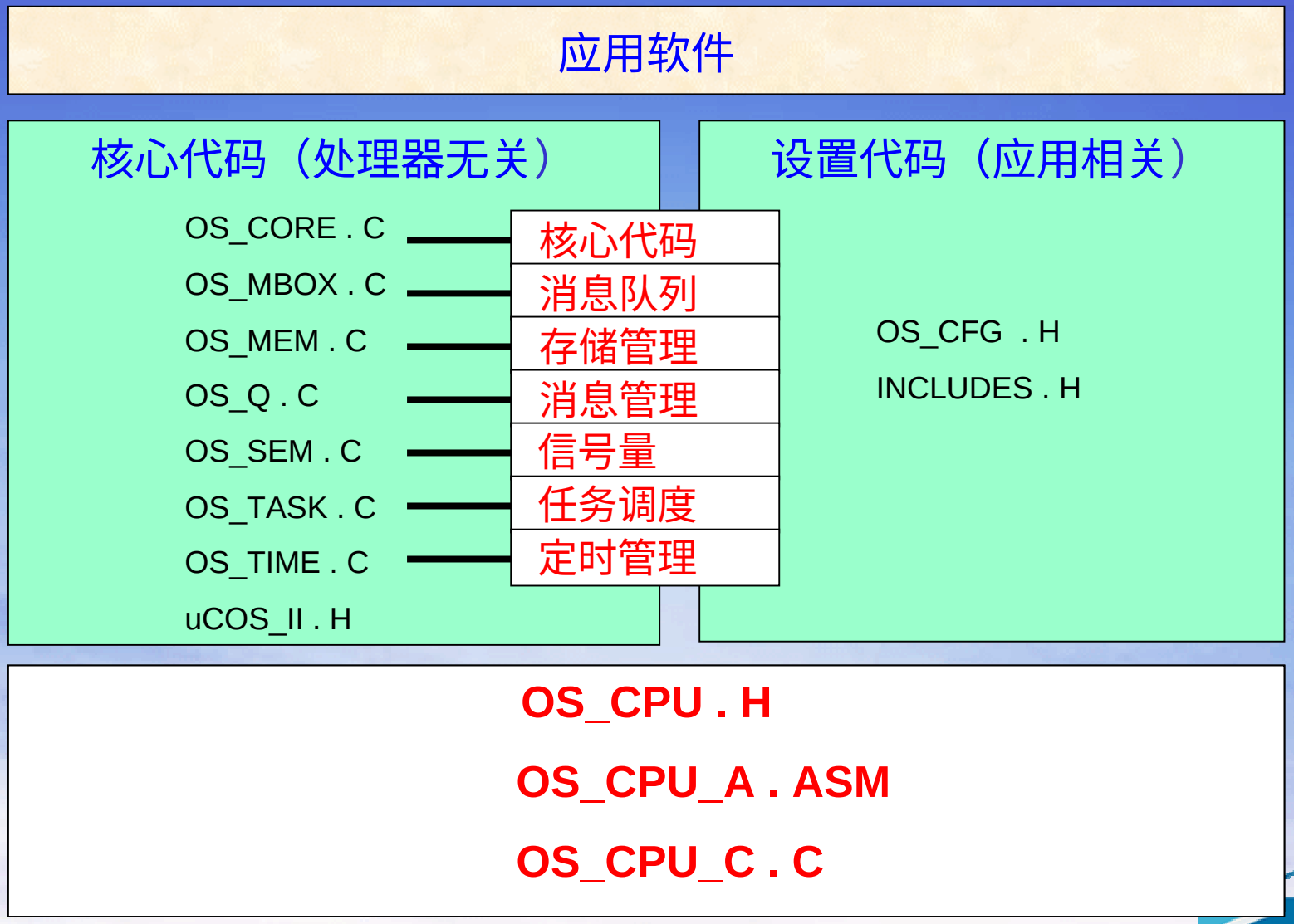
μCOS-II的特点

- 可移植性
- 可裁剪
- 占先式
- 多任务
- 中断管理
- 其它特点

- 绝大部分的μCOS-II的
- 公开源代码
- 可固化
- 可确定性
- 任务栈
- 提供很多系统服务
- 稳定性和可靠性强

三、 μCOS-II移植

μCOS-II体系


应用软件
核心代码（处理器无关）
OS_CORE . C
OS_MBOX . C
OS_MEM . C
OS_Q . C
OS_SEM . C
OS_TASK . C
OS_TIME . C
uCOS_II . H
核心代码
消息队列
存储管理
消息管理
信号量
任务调度
定时管理
设置代码（应用相关）
OS_CFG . H
INCLUDES . H
OS_CPU . H
OS_CPU_A . ASM
OS_CPU_C . C

移植的条件

1、处理器的C编译器能产生可重入代码

2、用C语言就可以打开和关闭中断

3、处理器支持中断并且能产生定时中断

4、处理器支持容纳一定量数据的硬件堆栈

5、处理器有将堆栈指针和其他CPU寄存器读出和存储到堆栈或内存中的指令

移植条件之一 ---- 可重入代码

可重入代码指的是可以被多个任务同时调用，而不会破坏数据的一段代码，或者说代码具有在执行过程中打断后再次被调用的能力。

```
int temp;
void swap (int *x,int*y)
{
        temp=*x;
        *X=*Y;
        *y=Temp;
}
```

```
void swap (int *x,int*y)
{
        int temp;
        temp=*x;
        *X=*Y;
        *y=Temp;
}
```

移植条件之二 ---- C语言开/关中断

uCOS-II在C语言代码中通过使用以下两个宏

OS_ENTER_CRITICAL ( )

OS_EXIT_CRITICAL( )

打开和关闭中断，从而保护临界代码

```
#define OS_ENTER_CRITICAL() ARMDisableInt()

ARMDisableInt:
        mrs     r12, CPSR               /* 获取模式寄存器 */
        orr     r12, r12, #I_BIT        /* 设置禁止中断位 */
        msr     CPSR_c, r12             /* 设置模式寄存器 */
        bx      lr
```

移植步骤之一

一、定义编译器相关的数据类型

二、定义允许和禁止中断

三、定义栈的增长

四、定义OS_TASK_S

```
typedef unsigned char   BOOLEAN;
typedef unsigned char   INT8U;
typedef signed   char   INT8S;
```

从低优先级任务切换到高优先级时调用

- 使用软中断直接将中断向量指向 OSCtxSw
- 直接调用 OSCtxSw

```
float           FP32;
```

```
#define OS_STK_GROWTH 0  /*从下往上*/
#define OS_STK_GROWTH 1  /*从上往下*/
```

移植步骤之二 ---- OS_CPU_A.ASM

- OSStartHighRdy( )
- OSCtxSw( )
- OSIntCtxSw()
- OSTickISR

定时中断函数;

OSTickISR( )主要负责在进入时保存处理器寄存器，完成任务的切换，退出时恢复寄存器并返回。

堆栈中恢复出来。

移植步骤之三 ---- OS_CPU_C.C

- OSTaskStkInit( )
- OSTaskCreateHook( )
- OSTaskDelHook( )
- OSTaskSwHook( )
- OSTaskStatHook( )
- OSTimeTickHook( )

任务创建时调用本函数;

OSTaskStkInit( ) 负责初始化任务的堆栈结构;

用来扩展μCOS-II功能，必须声明，但并不一定要包含任何代码。

四、μCOS-II使用

使用过程

- 分配任务栈
- 建立任务函数体
- 启动任务描述
- 系统初始化、任务启
- 编译并下载到目标板

unsigned int Stackn[STACKSIZE];
分配任务栈的主要目的是为应用程序运行时的变量、堆栈提供存放和访问空间

```
void Taskn(void *Id)
{
```

```
void TaskStart (void *Id)
{
…
OSTaskCreate(Taskn, (void *)0,
```

主要包括运行任务前的硬件初始化、操作系统的初始化、启动定时中断、启动任务等

任务间通信

μCOS-II的任务间通信可以通过邮箱和消息队列

使用邮箱：

OS_MBOX_EN

主要操作包括：

- OSMboxCreate() — 邮箱建立
- OSMboxPend() — 等待消息
- OSMboxPost() — 发送消息

使用消息队列：

OS_Q_EN

主要操作包括：

- OSQCreate() — 消息建立
- OSQPend() — 等待消息
- OSQPost() — 发送消息

任务同步

μCOS-II的任务间同步可以使用事件标志组和信号量

使用信号量：

OS_SEM_EN

主要操作包括：

OSSEMCreate() — 信号量建立

OSSEMPend() — 等待信号量

OSSEMPost() — 发送信号量

事件标志：

事件的定义

等待任务列表

同步：

等待事件的任务都放入等待任务列表；当事件发生后再从任务列表删除等待任务

RTOS调用

μCOS-II还包括时间管理、内存管理等操作功能

时钟节拍

实现延时和超时控制

时间管理包括：

- OSTimeDly() — 延时函数
- OSTimeGet() — 取系统时间
- OSTimeSet() — 修改系统时间

内存分区：

分块管理

主要操作包括：

- OSMemCreate() — 建内存分区
- OSMemGet() — 分配内存块
- OSMemPut() — 释放内存块

谢谢大家！！

http://www.embedinfo.com
Support@embedinfo.com