捷恩斯威
JEANSWAY

uC/OS-II在LM4F中的移植与应用

Let`s make your development easier!

捷恩斯威科技，最专业的TI MCU方案设计商

www.jeansway.cn

捷恩斯威
JEANSWAY

目录

捷恩斯威
JEANSWAY

uC/os-II简介

1. μC/OS-II 具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点， 最小内核可编译至 2KB 。

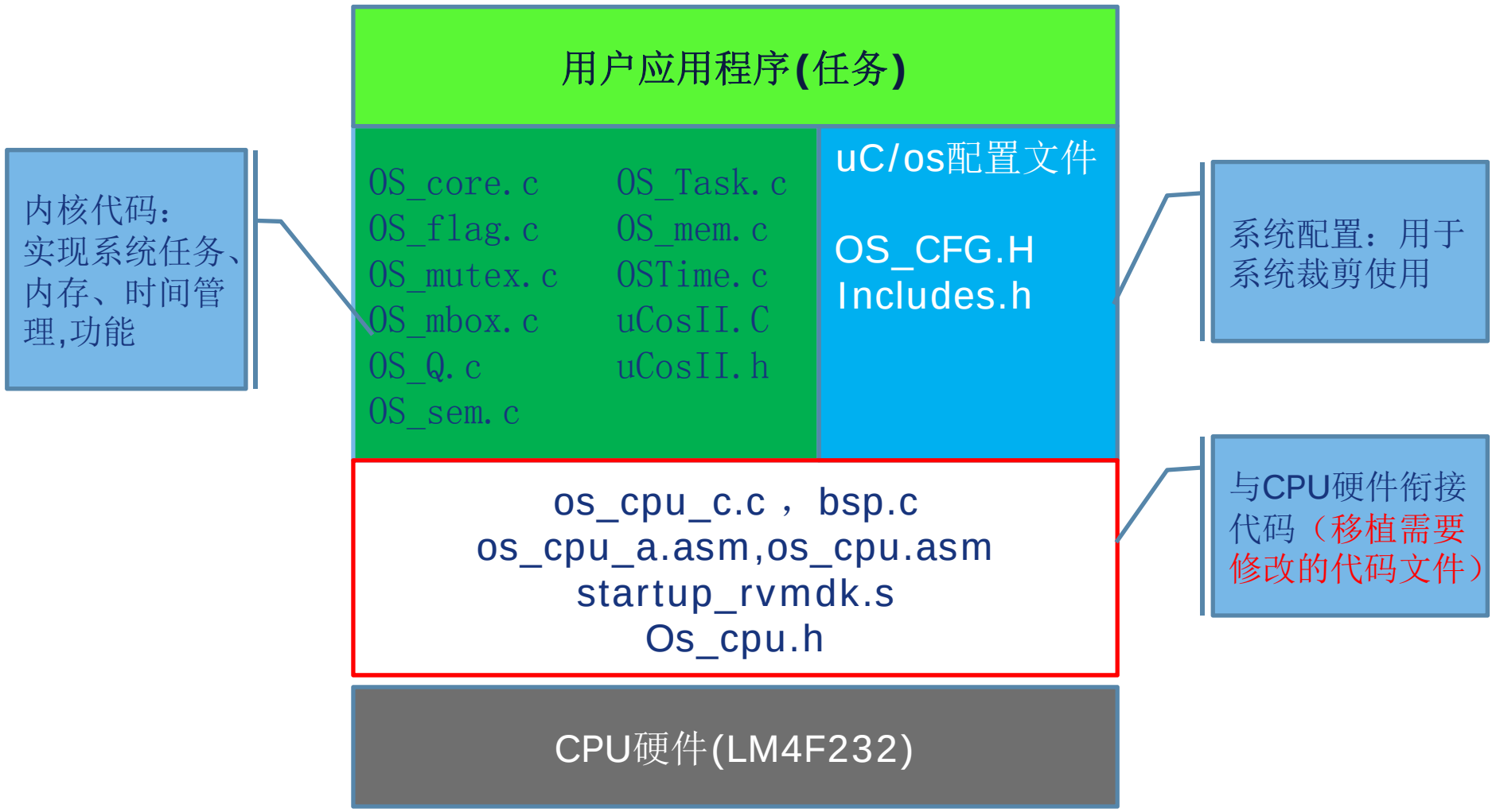
2. μC/OS-II是一种可移植的,可植入ROM的,可裁剪的,抢占式的，实时多任务操作系统内核. μC/OS-II 已经移植到了几乎所有知名的CPU上.

3. uC/OS-II只一个实时操作系统内核，它包含了任务调度，任务管理，时间管理，内存管理和任务间的通信和同步等基本功能。

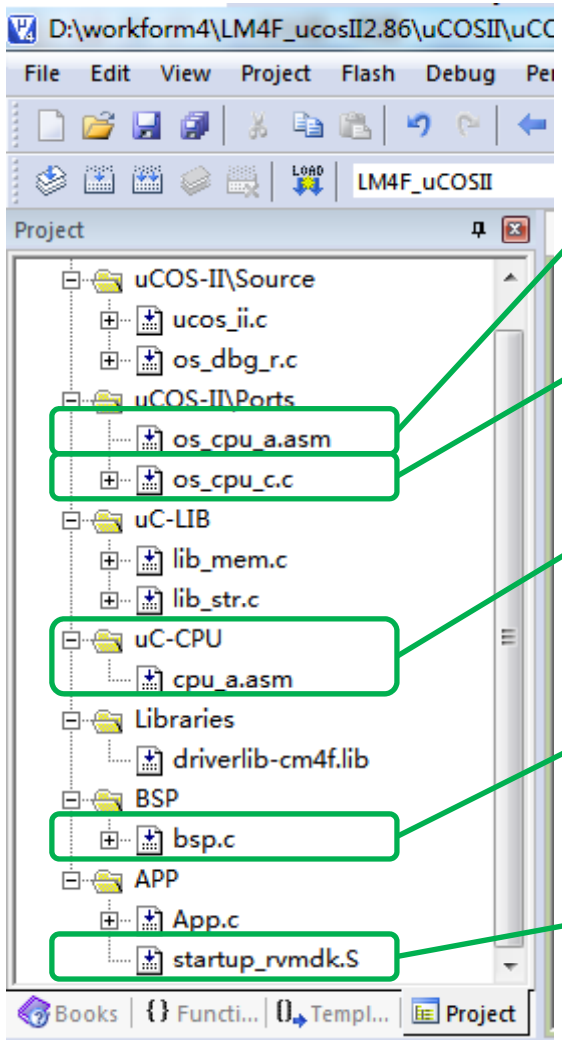
4. uC/OS-II 基于优先级调度的抢占式的实时内核，并在这个内核之上提供最基本的系统服务，如信号量，邮箱，消息队列，内存管理，中断管理等

μC/OS-II™
The Real-Time Kernel

uC/os-II体系结构


用户应用程序(任务)
OS_core.c
OS_flag.c
OS_mutex.c
OS_mbox.c
OS_Q.c
OS_sem.c
OS_Task.c
OS_mem.c
OSTime.c
uCosII.C
uCosII.h
uC/os配置文件
OS_CFG.H
Includes.h
os_cpu_c.c ， bsp.c
os_cpu_a.asm,os_cpu.asm
startup_rvmdk.s
Os_cpu.h
CPU硬件(LM4F232)
内核代码:
实现系统任务、内存、时间管理,功能
系统配置：用于系统裁剪使用
与CPU硬件衔接代码（移植需要修改的代码文件）

uC/OS-II移植到M4上需要修改的几个文件



OSStartHighRdy()
OSCtxSw()
OSIntCtxSw ()

OSTaskStkInit()

CPU_IntDis()
CPU_IntEn()

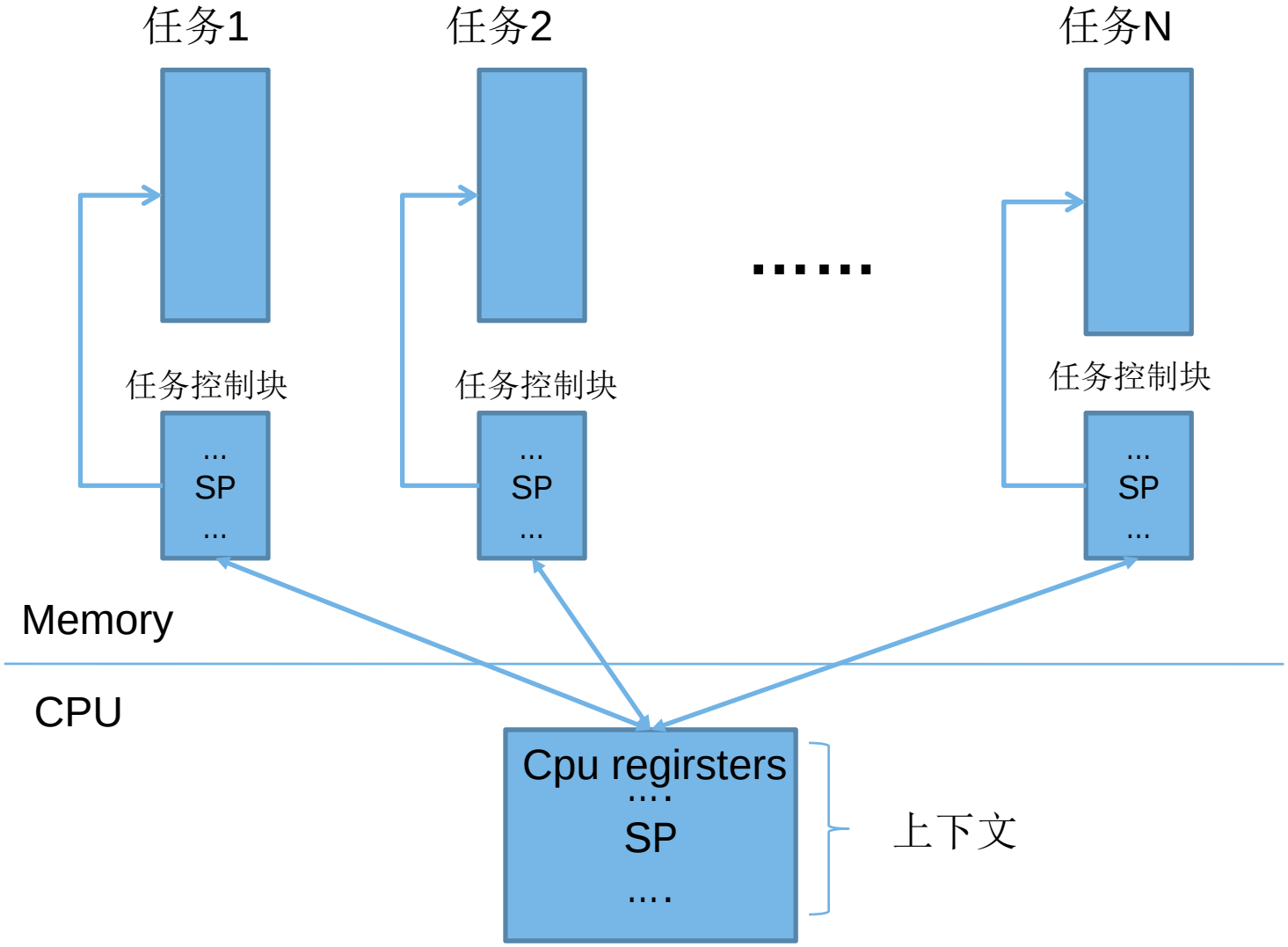
Tmr_TickInit()
Tmr_TickISR_Handler()

M4启动文件，设置系统运行堆栈，注册中断

OS_CPU.h：

数据类型和变量
临界区管理方法
堆栈方向定义

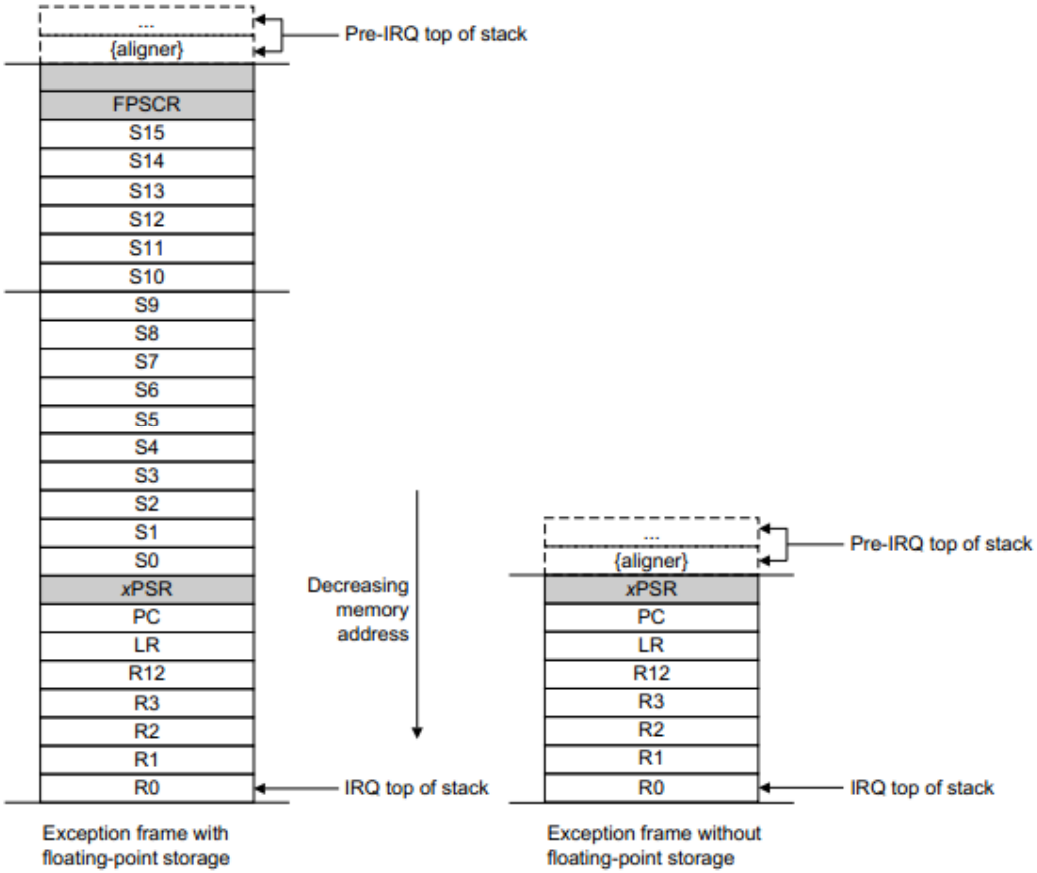
uCos的任务调度(1/2)



uCos的任务调度(2/2)

LM4F的堆栈压栈示意图

Figure 2-7. Exception Stack Frame



移植代码步骤

Step1: 移植OSStartHighRdy()

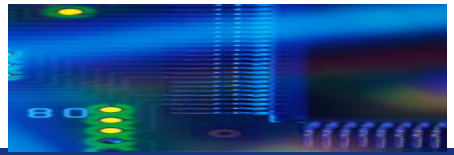
Step2: 移植OSCtxSW()和OSIntCtxSW()

Step3: 移植OSTaskStkInit()函数

Step4: 移植系统时钟节拍

Step5: 移植os_cpu.h中内容

Step6: 移植CPU_IntDis和CPU_IntEn



捷恩斯威 JEANSWAY

Step1：移植OSStartHighRdy()

该函数由OSStart()函数调用,在uCOS启动后，调度最高优先级任务运行，该函数在系统中仅调用一次

```
OSStartHighRdy
      LDR     R4, =NVIC_SYSPRI2          ; 设置PendSV中断优先级
      LDR     R5, =NVIC_PENDSV_PRI
      STR     R5, [R4]

      MOV     R4, #0                     ;  初始化SP为0
      MSR     PSP, R4

      LDR     R4, =OSRunning             ;   置位OSRunnig标志，启动任务调度
      MOV     R5, #1
      STRB    R5, [R4]

      LDR     R4, =NVIC_INT_CTRL         ;  触发PENDSV中断
     LDR     R5, =NVIC_PENDSVSET
      STR     R5, [R4]
      CPSIE   I                          ;  使能处理器中断
OSStartHang
      B       OSStartHang
```

Step2: 移植OSCtxSW()和OSIntCtxSW()

任务切换函数

```
OSCtxSw

    LDR     R0, =NVIC_INT_CTRL          ; 触发PENDSV中断，引起任务切换
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

```
OSIntCtxSw

    LDR     R0, =NVIC_INT_CTRL          ; 触发PENDSV中断，引起任务切换
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

Step2：移植OSCtxSW()和OSIntCtxSW()

PendSV()中断代码(1/2)

```
OSPendSV

    CPSID   I
    MRS     R0, PSP
    CBZ     R0, OSPendSV_nosave          ; 第一次启动任务OSStartHighRdy跳过

    SUB     R0, R0, #0x20                ; 保存R4-R11
    STM     R0, {R4-R11}

    LDR     R1, =OSTCBCur                ; 保存当前用户堆栈
    LDR     R1, [R1]
    STR     R0, [R1]                     ;
```

移植Step2: OSCtxSW()和OSIntCtxSW()

PendSV()中断代码(2/2)

```
OSPendSV_nosave
    PUSH    {R14}                       ; 保存LR(程序返回值)
    LDR     R0, =OSTaskSwHook
    BLX     R0
    POP     {R14}

    LDR     R0, =OSPrioCur
    LDR     R1, =OSPrioHighRdy
    LDRB    R2, [R1]
    STRB    R2, [R0]
    LDR     R0, =OSTCBCur
    LDR     R1, =OSTCBHighRdy
    LDR     R2, [R1]
    STR     R2, [R0]

    LDR     R0, [R2]                    ; 获取新任务堆栈
    LDM     R0, {R4-R11}                ; 从新任务堆栈中恢复R4-R11
    ADDS    R0, R0, #0x20               ;
    MSR     PSP, R0                     ; 载入新任务堆栈到SP
    ORR     LR, LR, #0x04               ;
    CPSIE   I
    BX      LR
```

Step3：移植OSTaskStkInit()函数

```
OS_STK *OSTaskStkInit (void (*task)(void *p_arg), void *p_arg, OS_STK *ptos, INT16U opt)
{
    OS_STK *stk;
    (void)opt;
    stk      = ptos;
    *(stk)    = (INT32U)0x01000000L;          /* xPSR      */
    *(--stk)  = (INT32U)task;                 /* Entry Point*/
    *(--stk)  = (INT32U)0xFFFFFFFEL;          /* R14 (LR)  */
    *(--stk)  = (INT32U)0x12121212L;          /* R12       */
    *(--stk)  = (INT32U)0x03030303L;          /* R3        */
    *(--stk)  = (INT32U)0x02020202L;          /* R2        */
    *(--stk)  = (INT32U)0x01010101L;          /* R1        */
    *(--stk)  = (INT32U)p_arg;                /* R0        */
    *(--stk)  = (INT32U)0x11111111L;          /* R11       */
       ... ...
    *(--stk)  = (INT32U)0x04040404L;          /* R4        */
    return (stk);
}
```

Step4: 移植系统时钟节拍 (bsp.c)

系统时钟初始化(10ms)

```
void  Tmr_TickInit (void)
{
    ROM_SysTickPeriodSet((INT32U)(ROM_SysCtlClockGet()/OS_TICKS_PER_SEC) -1 );
    ROM_SysTickEnable();
    ROM_SysTickIntEnable();
}
```

```
OS_CFG.H
#define OS_TICKS_PER_SEC    100
```

系统时钟中断函数(在startup_rvmdk.s中注册)

```
void  Tmr_TickISR_Handler (void)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif

    OS_ENTER_CRITICAL();    /* Tell uC/OS-II that we are starting an ISR*/
    OSIntNesting++;
    OS_EXIT_CRITICAL();
    OSTimeTick();           /* Call uC/OS-II's OSTimeTick()             */
    OSIntExit();            /* Tell uC/OS-II that we are leaving the ISR */
}
```

Step5: 移植os_cpu.h中内容

1. 定义uCOS要用到的基本数据类型和变量

2. 定义软件触发PendSV产生任务切换

```
#define   OS_TASK_SW()  OSCtxSw
```

```
OSCtxSw
    LDR     R0, =NVIC_INT_CTRL
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

3. 定义堆栈的生长方向

```
#define  OS_STK_GROWTH  1 //1 从高到低方向生长，0从低到高方向生长
```

4. 定义临界点处理方法

临界点处理使用方法3,即进入临界区时保存中断使能状态,禁止中断，退出时加以恢复

```
#define  OS_CRITICAL_METHOD   3u
#define  OS_ENTER_CRITICAL()  {cpu_sr = OS_CPU_SR_Save();}
#define  OS_EXIT_CRITICAL()   {OS_CPU_SR_Restore(cpu_sr);}
```

```
OS_CPU_SR_Save
    MRS     R0, PRIMASK
    CPSID   I
    BX      LR
```

```
OS_CPU_SR_Restore
    MSR     PRIMASK, R0
    BX      LR
```

Step6: 移植CPU_IntDis和CPU_IntEn

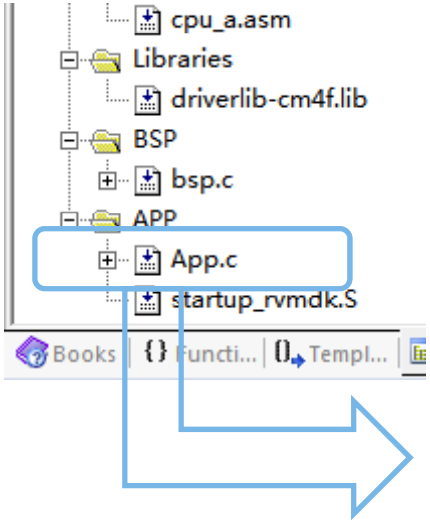
```
CPU_IntDis
        CPSID   I
        BX      LR
```

禁止中断

```
CPU_IntEn
        CPSIE   I
        BX      LR
```

使能中断

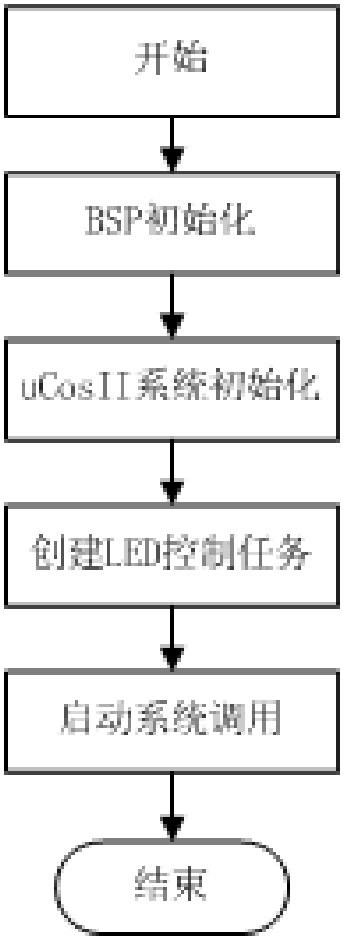
Step7: 在uCosII中创建一个任务



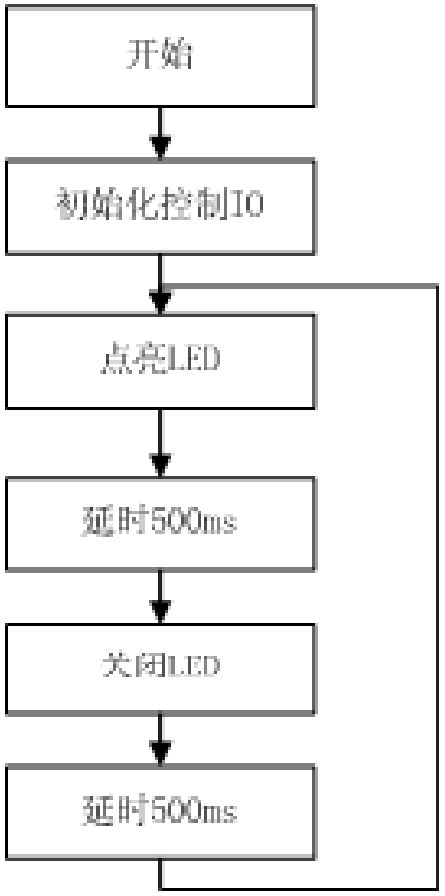

```
int main (void)
{
    BSP_PreInit ();

    BSP_IntDisAll();

    OSInit();                                              /* 初始化ucosii系统*/

    OSTaskCreateExt((void (*)(void *)) App_TaskStart,      /* 任务入口函数名 */
            (void       *) 0,                              /* 任务入口函数参数 */
            (OS_STK *)&App_TaskStartStk[APP_CFG_TASK_START_STK_SIZE - 1], /* 任务栈顶*/
            (INT8U ) APP_CFG_TASK_START_PRIO,              /* 任务优先级 */
            (INT16U) APP_CFG_TASK_START_PRIO,              /* 任务编号 */
            (OS_STK *)&App_TaskStartStk[0],                /* 任务栈底*/
            (INT32U) APP_CFG_TASK_START_STK_SIZE,          /* 任务栈大小 */
            (void       *) 0,
            (INT16U )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));  /* 任务参数 */
    OSStart();
}
```

LED控制流程图

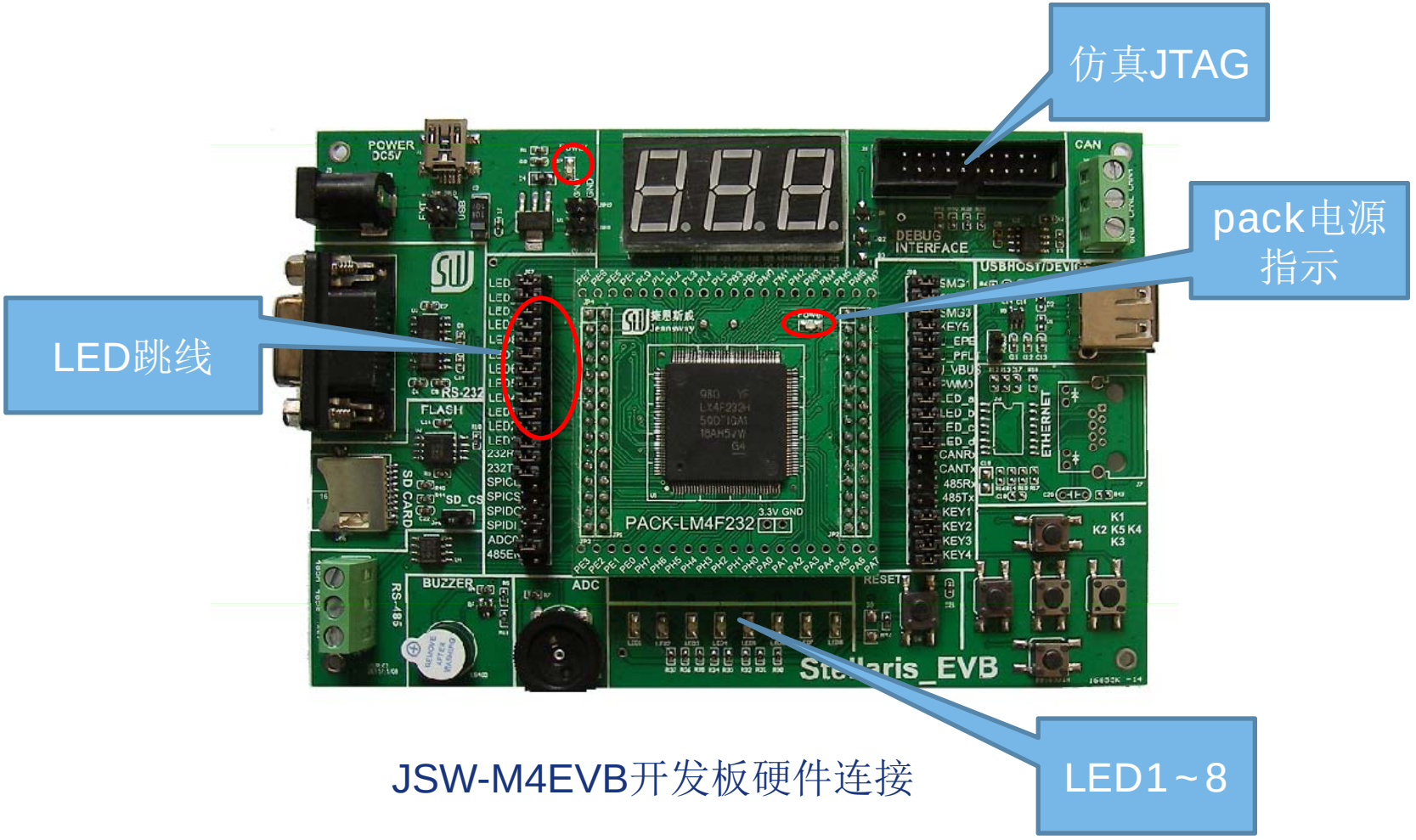
系统流程图



任务流程图

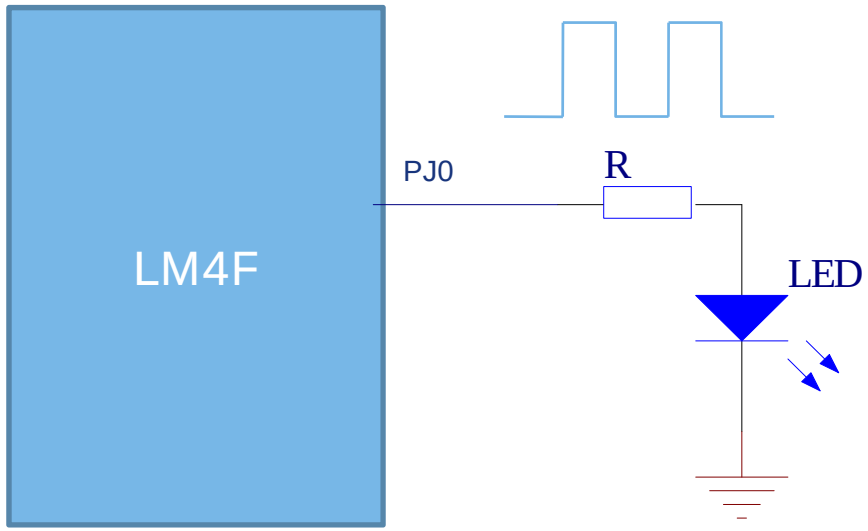


JSW-M4EVB开发板uC/os应用硬件连接



JSW-M4EVB开发板硬件连接

uC/osII LED任务输出示意图



移植好uCosII后,LED控制任务运行,控制LED闪烁

捷恩斯威
JEANSWAY

动手环节：创建一个任务实现流水灯功能

1. 在现有uCos的工程上，创建一个任务led_step_task

2. 要求实现以下功能：8个LED按顺序单个轮流点亮，不断循环。单个LED每次点亮时间在1秒之间。

1. 任务入口函数 led_step_task
2. 任务优先级 5
3. 任务堆栈 Led_StepTask_Stk
4. 任务堆栈大小 LED_STEP_TASK_STK_SIZE

完成后请马上举手示意，最快完成者，经现场确认合格后可获得奖品一个。

捷恩斯威 JEANSWAY

在线技术支持

https://www.deyisupport.com/
https://www.deyisupport.com/question_answer/f/57.aspx

捷恩斯威
JEANSWAY

Thank You !

www.jeansway.cn