uC/OS II 在LPC2200单片机移植说明

(Draft)

飞雪季节 2007-7-11

修改记录

版本号	说明	修改人	日期
01.100	初始记录	周志刚	2006-09-11
01.101	修改IRQ汇编处理，参考《ADS_DeveloperGuide_D.pdf》		2007-7-09

目录

uC/OS II 在LPC2200单片机移植说明

1 摘要

本文详细介绍uC/OS在菲利普ARM LPC2200平台移植过程，主要目的是为了经验积累以及后续软件设计参考。

以下内容主要参考周立功等编著《ARM微控制器基础与实战》，适当修改。

移植代码在这里仅做验证功能，还有优化空间，后续在嵌入式平台代码会有详细描述。

2 uC/OS 需要移植选项说明

2.1 OS_CPU_C.C

2.1.1 OSTaskStkInit()

初始化任务堆栈，主要设置堆栈内容，保存一些寄存器，任务返回地址。

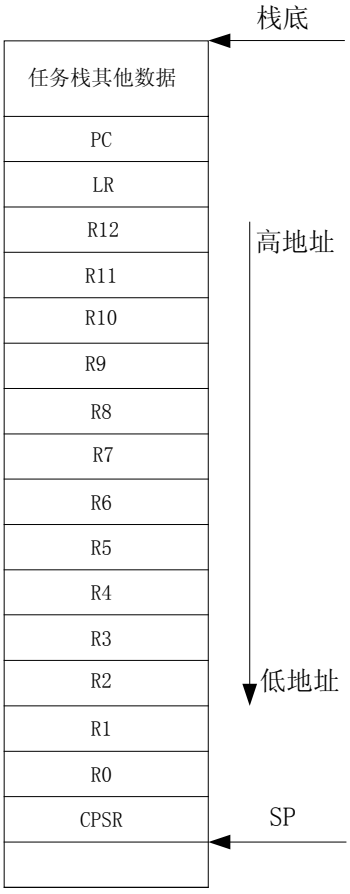


图1 ARM 堆栈结构

```
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *pdata,
                                    OS_STK *ptos, INT16U opt)
{
    OS_STK *stk;

    opt    = opt;    /* 'opt'  没有使用。作用是避免编译器警告    */
    stk    = ptos;  /* 获取堆栈指针                                  */

    /* 建立任务环境，ADS1.2使用满递减堆栈       */
    *stk = (OS_STK) task;                  /*  pc  */
    *--stk = (OS_STK) task;                /*  lr  */
    *--stk = 0;                           /*  r12  */
    *--stk = 0;                           /*  r11  */
    *--stk = 0;                           /*  r10  */
    *--stk = 0;                           /*  r9   */
    *--stk = 0;                           /*  r8   */
    *--stk = 0;                           /*  r7   */
    *--stk = 0;                           /*  r6   */
    *--stk = 0;                           /*  r5   */
    *--stk = 0;                           /*  r4   */
    *--stk = 0;                           /*  r3   */
    *--stk = 0;                           /*  r2   */
    *--stk = 0;                           /*  r1   */
    *--stk = (unsigned int) pdata;       /*  r0,第一个参数使用R0传递   */
    *--stk = (USER_USING_MODE|0x00); /*  spsr，允许 IRQ，FIQ 中断   */
    return (stk);
}
```

2.1.2 OSTaskCreateHook()

空函数

2.1.3 OSTaskDelHook()

空函数

2.1.4 OSTaskSwHook()

空函数

2.1.5 OSTaskStartHook()

空函数

2.1.6 OSTimeTickHook()

可以用来设置处理任务进程定时器链表

2.2 OS_CPU.H

主要是一些数据类型定义

于处理器相关的代码：

2.2.1 关中断

```
int32 os_enter_flag = 0;
```

该变量用于判断当前是否已经屏蔽IRQ中断，防止退出临界代码时候，打开不该打开的中断，后面的代码处理可能有些罗唆，先跑起来再说，后面我再优化。

```
void OS_ENTER_CRITICAL(void)
{
   if(_OS_ENTER_CRITICAL() > 0)
   {
      os_enter_flag = 1;
   }
   else
   {
      os_enter_flag = 0;
   }
}

_OS_ENTER_CRITICAL
             STMFD   SP!, {R1, LR}

             MRS     R0, CPSR
             LDR     R1, =0x80
             AND     R0,R1,R1
             CMP   R1, #0
             BNE  _OS_ENTER_RETURN

             ORR     R0, R0, #(NoInt)
             MSR     CPSR_c, R0
```

```
_OS_ENTER_RETURN

                MOV    R0, R1
                LDMFD   SP!, {R1, PC }
```

2.2.2 开中断

```
void OS_EXIT_CRITICAL(void)
{
  if(0 == os_enter_flag)
  {
    _OS_EXIT_CRITICAL() ;
  }
}
_OS_EXIT_CRITICAL
           STMFD   SP!, {R0, LR}
              MRS     R0, CPSR
              BIC     R0, R0, #(NoInt)
              MSR     CPSR_c, R0
              LDMFD   SP!, {R0, PC }
```

2.2.3 定义堆栈增长方向

1＝向下 ，0 ＝ 向上*

```
#define OS_STK_GROWTH        1
```

2.3 OS_CPU_A.ASM

2.3.1 OSStartHighRdy()

运行优先级高的任务，原UC/OS II说明如下：

```
void OSStartHighRdy()
{
   调用用户可定义的 OSTaskSwHook();
   获取任务的堆栈指针用户恢复:
     堆栈指针  ＝ OSTCBHighRdy->OSTCBStkPtr;
     OSRunnig = TRUE;
     从新任务的堆栈中恢复所有的处理器寄存器;
```

```
    从中断指令返回;
}
此入口通过SWI软中断进入
;软件中断处理参考2.3.2小节

__OSStartHighRdy
;进入系统模式，IRQ中断禁止
    MSR     CPSR_c, #(NoInt | SYS32Mode)
;告诉uC/OS-II自身已经运行
    LDR     R4, =OSRunning
    MOV     R5, #1
    STRB    R5, [R4]
 ;调用钩子函数
    BL      OSTaskSwHook
; 因为OSTCBHighRdy是个指针，所以这个只是找到存放最高优先级PCB数据结构的首地址
;也就是存放SP地址的:     OSTCBHighRdy->OSTCBStkPtr的地址
    LDR     R6, =OSTCBHighRdy
    LDR     R6, [R6]
    B       OSIntCtxSw_1
OSIntCtxSw_1
;获取新任务堆栈指针 OSTCBHighRdy->OSTCBStkPtr，存放在R4中
    LDR     R4, [R6]
;参考上面ARM寄存器压栈图，下面SP指针指向PC的上一个地方
    ADD     SP, R4, #64       ;16寄存器CPSR, R0-R12,LR,SP
;恢复出LR值
    LDR     LR, [SP, #-8]
;进入管理模式，禁止IRQ中断
    MSR     CPSR_c, #(NoInt | SVC32Mode)
;设置堆栈指针为正常
    MOV     SP, R4
;恢复出堆栈中的CPSR
    LDMFD   SP!, {R5}
;恢复CPSR
    MSR     SPSR_cxsf, R5
;运行新任务
    LDMFD   SP!, {R0-R12, LR, PC }^
```

2.3.2 OSCtxSw()

1) 任务切换，使用软中断SWI

```
__swi(0x00) void OS_TASK_SW(void);
__swi(0x01) void _OSStartHighRdy(void);
```

2) 软中断入口汇编处理

```
此入口通过SWI软中断进入
;软件中断，软中断制动进入管理模式
SoftwareInterrupt
        LDR     SP, StackSvc            ; 重新设置堆栈指针
        STMFD   SP!, {R0-R3, R12, LR}
        MOV     R1, SP                  ; R1指向参数存储位置，因为SWI软中断允许
                                        使用堆栈传递最多4个参数。

        MRS     R3, SPSR
        TST     R3, #T_bit              ; 中断前是否是Thumb状态
        LDRNEH  R0, [LR,#-2]            ; 是：取得Thumb状态SWI号
        BICNE   R0, R0, #0xff00
        LDREQ   R0, [LR,#-4]            ; 否：取得arm状态SWI号
        BICEQ   R0, R0, #0xFF000000
                                ; r0 = SWI号，R1指向参数存储位置
        CMP     R0, #1
        ; 如果中断号为0表示进行普通任务切换
        LDRLO   PC, =_OSIntCtxSw
        ; SWI 0x01为第一次任务切换
        LDREQ   PC, =__OSStartHighRdy
        ;其他软中断号，转移到C语言函数处理
        BL      SWI_Exception
        ;恢复任务，同时恢复CPSR
        LDMFD   SP!, {R0-R3, R12, PC}^
```

3) SWI_Exception函数处理

这个函数目前不会进入。

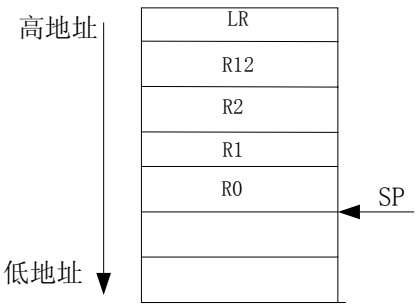
2.3.3 OSIntCtxSw()

中断退出任务切换，同时作为OSCtxSw()的后续处理，主要的功能是保存当前任务的堆栈，主要是CPSR、R0-R12，LR、PC，从UC/OS-II书本上描写来看，OSIntCtxSw()是被OSIntExit()调用，代码如下：

```
void OSIntExit (void)
{
    OS_ENTER_CRITICAL();
    if ((--OSIntNesting | OSLockNesting) == 0)
    {
/* Reschedule only if all ISRs completed & not locked */
        OSIntExitY    = OSUnMapTbl[OSRdyGrp];
        OSPrioHighRdy = (INT8U)((OSIntExitY << 3) +
                        OSUnMapTbl[OSRdyTbl[OSIntExitY]]);
        if (OSPrioHighRdy != OSPrioCur)
        {
 /* No context switch if current task is highest ready */
            OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
            OSCtxSwCtr++;
  /* Keep track of the number of context switches      */
            OSIntCtxSw();
   /* Perform interrupt level context switch           */
        }
    }
    OS_EXIT_CRITICAL();
}
```

这个函数的主要目的是判断是否需要进行任务调度，因为发生了中断，可能激活了更高优先级的任务，如果是的话，为了保证实时性直接调用OSIntCtxSw()进行任务切换。

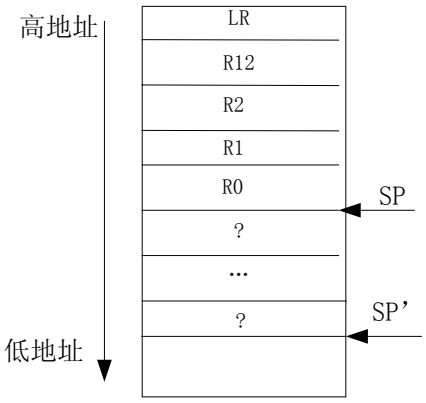
这里带来一个问题，因为进入函数OSIntExit()有寄存器压栈，但你不知道确切如栈的寄存器个数，需要对编译器了解的很清楚，下图是进入OSIntExit()前IRQ模式下堆栈结构（LR上部的内容没有给出）。



IRQ中断模式，进入OSIntCtxSw()前堆栈结构

图2 进入 OSIntExit()前堆栈结构

下图为进入OSIntExit()后堆栈结构



IRQ中断模式，进入OSIntCtxSw()后堆栈结构

图3 进入 OSIntExit()后堆栈结构

目前堆栈指针的地址为SP'，这个时候如果立即进行任务切换，我们必须知道上次堆栈SP的位置，由于OSIntExit()用C语言编写，我们无法知道编译器的编译结果，那么也就无从调整SP'到SP的位置，任务切换的时候就不能正确恢复USER状态下的用户堆栈。

为了简单处理，我们可以直接这样定义：

#define OSIntCtxSw() 或者直接 RETURN，实际代码我给注释掉了。

把这个函数定义为空，具体调度与否让后面的汇编去处理，这样编译器就会自动的计算出栈位置，恢复出SP来，因为目前是禁止IRQ中断的，所以不会被IRQ中断给打断处理。下面给出汇编的中断恢复任务处理。

```
_OSIntCtxSw
                                        ;下面为保存任务环境
        LDR     R2, [SP, #20]             ;获取PC
        LDR     R12, [SP, #16]            ;获取R12
        MRS     R0, CPSR
;切换到SYS模式，因为该模式同USER模式共用所有寄存器。
        MSR     CPSR_c, #(NoInt | SYS32Mode)
        MOV     R1, LR                     ;LR是否正确，后面需要验证
        STMFD   SP!, {R1-R2}              ;保存LR,PC
        STMFD   SP!, {R4-R12}             ;保存R4-R12
;恢复CPSR，切换到IRQ模式
        MSR     CPSR_c, R0
        LDMFD   SP!, {R4-R7}            ;获取R0-R3
        ADD     SP, SP, #8              ;出栈R12,PC，还原IRQ模式下SP地址
  ;切换到SYS模式，因为该模式同USER模式共用所有寄存器
        MSR     CPSR_c, #(NoInt | SYS32Mode)
```

```
        STMFD   SP!, {R4-R7}                    ;保存R0-R3
        ;这里不用OsEnterSum
        ; LDR     R1, =OsEnterSum                ;获取OsEnterSum
        ;LDR     R2, [R1]
        ;STMFD   SP!, {R2, R3}                  ;保存CPSR,OsEnterSum
         STMFD   SP!,{R3} ;保存CPSR
                                                ;保存当前任务堆栈指针到当前任务的TCB
        LDR     R1, =OSTCBCur
        LDR     R1, [R1]
        STR     SP, [R1]

        BL      OSTaskSwHook                    ;调用钩子函数
                                                ;OSPrioCur <= OSPrioHighRdy
        LDR     R4, =OSPrioCur
        LDR     R5, =OSPrioHighRdy
        LDRB    R6, [R5]
        STRB    R6, [R4]
                                                ;OSTCBCur <= OSTCBHighRdy
        LDR     R6, =OSTCBHighRdy
        LDR     R6, [R6]
        LDR     R4, =OSTCBCur
        STR     R6, [R4]
OSIntCtxSw_1
;获取新任务堆栈指针 OSTCBHighRdy->OSTCBStkPtr，存放在R4中
        LDR     R4, [R6]
;参考上面ARM寄存器压栈图，下面SP指针指向PC的上一个地方
        ADD     SP, R4, #68       ;17寄存器CPSR,OsEnterSum,R0-R12,LR,SP
;恢复出LR值
        LDR     LR, [SP, #-8]
;进入管理模式，禁止IRQ中断
        MSR     CPSR_c, #(NoInt | SVC32Mode)
;设置堆栈指针为正常
        MOV     SP, R4
;恢复出堆栈中的CPSR,OsEnterSum
        LDMFD   SP!, {R4, R5}
;恢复新任务的OsEnterSum
        LDR     R3, =OsEnterSum
        STR     R4, [R3]
;恢复CPSR
        MSR     SPSR_cxsf, R5
;运行新任务
            LDMFD   SP!, {R0-R12, LR, PC }^
```

2.3.4 OSTickISR()

时钟处理，完全可以写成C函数，参考函数 VICInit()。

2.4 向量中断注册函数的编写

1) ISR宏定义

```
  MACRO
$IRQ_Label HANDLER $IRQ_Exception_Function

       EXPORT  $IRQ_Label                  ; ?????
       IMPORT  $IRQ_Exception_Function       ; ???????

$IRQ_Label
       SUB     LR, LR, #4                  ; 计算返回地址
       STMFD   SP!, {R0-R3, R12, LR}         ; 保存任务环境
       MRS     R3, SPSR                    ; 保存状态

       STMFD   SP!, {R3}


       LDR     R2,  =OSIntNesting           ; OSIntNesting++
       LDRB    R1, [R2]
       ADD     R1, R1, #1
       STRB    R1, [R2]

       MSR  CPSR_c, #(0x1f);process it asn arm UserGuid
       STMFD SP!, {R0-R3, LR};process it asn arm UserGuid

       BL      $IRQ_Exception_Function       ; 调用c语言的中断处理程序

       LDMFD SP!, {R0-R3, LR};process it asn arm UserGuid

       MSR  CPSR_c, #(0x92);process it asn arm UserGuid


       BL      OSIntExit


       MSR     CPSR_c, #(NoInt | IRQ32Mode)    ; 切换回irq模式
       LDR     R0, =OSTCBHighRdy
       LDR     R0, [R0]
```

```
    LDR     R1, =OSTCBCur
    LDR     R1, [R1]
    CMP     R0, R1

    LDMFD   SP!, {R3}

    MSR     SPSR_cxsf, R3
    LDMEQFD SP!, {R0-R3, R12, PC}^          ; 不进行任务切换
    LDR     PC, =OSIntCtxSw                 ; 进行任务切换
MEND
```

2) C语言编写

注册中断服务程序：

3 附录

3.1 参考资料