

## 晶心科技 技术文章

### $\mu$ C/OS-II 在 AndesCore™ N1033A-S 上的移植

發表人：周杰，应用工程师，晶心宏科技（杭州）有限公司

Tel : 0571 - 85269013 - 307

Mobile: 13706511046

[zhoujie@andestech.com.cn](mailto:zhoujie@andestech.com.cn)

.....晶...心...科...技...新...闻...联...络...人.....

市场部 Janine 徐家玲经理

电话: 886-3-6668300 ext. 614

手机: 886-932-315015

E-mail: [Janine@andestech.com](mailto:Janine@andestech.com)

Web: [www.andestech.com](http://www.andestech.com)

# μC/OS-II 在 AndesCore™ N1033A-S 上的移植

周杰, 应用工程师, 晶心宏科技(杭州)有限公司

μC/OS-II 是一种代码公开、可裁剪的嵌入式实时多任务操作系统。该内核通过实现抢占式任务调度算法和多任务间通信等功能, 使之具有执行效率高、实时性能优良等特点。另外, 其占用空间非常小(最小可裁剪至 2KB)并且具有高度可移植性, 因此被广泛的应用于微处理器和微控制器上。

晶心科技 (Andes)作为亚洲首家原创性 32 位微处理器 IP 与系统芯片平台设计公司, 推出的 AndesCore™ N10 系列产品 N1033A-S, 搭配应用广泛的嵌入式实时操作系统 μC/OS-II 以及相关的软硬件开发资源, 有效的帮助客户降低现有成本、提升系统效能、减少系统功耗, 并缩短产品开发上市时程。本文将介绍如何将 μC/OS-II 移植到 AndesCore™ N1033A-S 处理器上。

## 1. 开发环境及处理器介绍

### 1.1 软/硬件开发环境

本移植过程使用的软件环境是 AndeSight™ v1.4 集成开发套件, 它是晶心科技最新推出的针对各种 AndesCore™ 的软件集成开发环境, 包括编译器、调试器、分析器以及强大的 ESL 工具。硬件平台采用晶心科技的 FPGA 评估板 ADP-XC5, 该评估板采用 AndesCore™ N1033A-S 作为处理器内核, 并具有丰富的片上资源。

### AndesCore™ N1033A-S 介绍

AndesCore™ N10 系列产品 N1033A-S 是一款哈佛结构的 32 位 RISC 处理器内核, 具有 5 级流水线(pipeline)及动态分支预测(Dynamic branch prediction)架构。N1033A-S 新加入了最新 AndeStar™ V2 指令集, 把 CPU 效能推至 1.66DMIPS/Mhz 之上。同时还实现完整的 Audio 指令集, 达到完全整合 CPU 与 DSP 功能的目标。N1033A-S 还支持向量中断模式以及 2D 直接内存访问(DMA)功能, 更为实时信号处理添增效能。

技术文章发表, 请尽速发布

2011年8月2日

## 2. $\mu$ C/OS-II 在 N1033A-S 上的可移植性分析

$\mu$ C/OS-II 具有高度可移植性, 目前已经移植到近 40 多种处理器体系上, 涵盖从 8 位到 64 位的各种 CPU (包括 DSP)。

$\mu$ C/OS-II 的正常运行需要处理器平台满足以下要求: 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)处理器支持能够容纳一定量数据的硬件堆栈; 5)处理器有将堆栈指针和其它 CPU 寄存器读出和存储到堆栈或内存中的指令。

AndesCore™ N1033A-S 内部提供了 32 个通用寄存器, 其中 R31 被用来做专门的堆栈指针。32 根地址线最多可访问 4GB 存储单元, 因此只要系统 RAM 空间允许, 堆栈空间理论不会产生限制。N1033A-S 处理器提供的 AndeStar™ V2 指令集包含了丰富且十分高效的对堆栈进行操作的指令。例如指令 SMW (store multiple word) 可实现仅使用一条指令将多个寄存器的值存储到堆栈中并同时更新堆栈指针位置, 而且还能很好的处理地址非对齐字的存取。N1033A-S 支持中断并能产生定时器中断, 处理器中的 PSW (Processor Status Word) 寄存器中包含一个全局中断禁止位 GIE, 控制它便可实现打开和关闭中断。此外, AndeSight™ 集成开发环境中内置的编译器可以产生可重入代码, 并且支持内联汇编, C 环境中可以任意进行开关中断的操作。综上所述,  $\mu$ C/OS-II 完全可以移植到 N1033A-S 上运行。

## 3. 移植步骤

为了方便移植, 大部分的  $\mu$ C/OS-II 代码是用 C 语言写的, 用户只需要用 C 语言和汇编语言写一些与处理器相关的代码就可以实现移植。这部分工作的内容包括: 一个完成基本设置的头文件 os\_cpu.h、一个与处理器相关的汇编文件 os\_cpu\_a.S 和一个与操作系统相关的 C 代码文件 os\_cpu\_c.c。

### 3.1 在 os\_cpu.h 中完成基本的配置和定义

#### 3.1.1. 定义与处理器相关的数据类型

为保证可移植性,  $\mu$ C/OS-II 没有直接使用 C 语言中的 short、int 和 long 等数据类型的定义, 因为不同的处理器有不同的字长。对于 N1033A-S 这样的 32

技术文章发表, 请尽速发布

2011年8月2日

位处理器, 其数据类型定义实现如下:

```
typedef unsigned char    BOOLEAN;
typedef unsigned char    INT8U;    /* Unsigned 8 bit quantity */
typedef signed   char    INT8S;    /* Signed   8 bit quantity */
typedef unsigned short   INT16U;   /* Unsigned 16 bit quantity */
typedef signed   short   INT16S;   /* Signed   16 bit quantity */
typedef unsigned int     INT32U;   /* Unsigned 32 bit quantity */
typedef signed   int     INT32S;   /* Signed   32 bit quantity */
typedef float           FP32;      /* Single precision floating point */
typedef double          FP64;      /* Double precision floating point */
typedef unsigned long    OS_STK;   /* Each stack entry is 32-bit wide */
typedef unsigned long    OS_CPU_SR; /* Define size of CPU status */
                                /* register (PSR = 32 bits) */
```

### 3.1.2. 定义中断禁止/允许宏

做为实时内核,  $\mu\text{C}/\text{OS-II}$  需要先禁止中断再访问代码临界区, 并且在访问完毕后重新允许中断。 $\mu\text{C}/\text{OS-II}$  定义了两个宏来禁止和允许中断:

OS\_ENTER\_CRITICAL()和 OS\_EXIT\_CRITICAL()。在 N1033A-S 处理器上的实现代码如下

```
#define OS_ENTER_CRITICAL() do{ GIE_SAVE( &cpu_sr);} while(0)
#define OS_EXIT_CRITICAL() do{ GIE_RESTORE(cpu_sr);} while(0)
```

GIE\_SAVE 和 GIE\_RESTORE 的实现如下:

```
static inline void GIE_SAVE(unsigned long *var)
{ *var = GET_PSW();    // save PSW register
  __asm__ volatile (" setgie.d \n\t"    //disable interrupt
                    " isb \n\t" ); //to guarantee setgie.d have done
                                //before fetching the following instruction
}
static inline void GIE_RESTORE(unsigned long var)
{ if (var & PSW_mskGIE)
  __asm__ volatile ( "setgie.e" ); // restore PSW.GIE
}
```

中断禁止时间是判断系统实时性的重要指标之一。中断禁止时间能否达到最短, 不仅与操作系统的设计有关, 还依赖于处理器结构和编译器产生的代码质量。

技术文章发表, 请尽速发布

2011年8月2日

从上面的实现代码看到, 由于 Andes 处理器提供了 setgie.d 和 setgie.e 两条直接控制中断的开关的指令, 整个禁止/允许中断的过程经过编译器产生的机器码只有 3/2 条, 最大限度地减小了中断禁止时间。

### 3.1.3. 定义栈增长方向

$\mu$ C/OS-II 使用结构常量 OS\_STK\_GROWTH 来指定堆栈的增长方式, 设置为 0 表示堆栈从下往上增长, 设置为 1 表示从上往下增长。这里我们定义成后者, 即堆栈的增长方向是从内存高地址向低地址方向递减并且堆栈指针总是指向栈顶数据:

```
#define OS_STK_GROWTH 1 /* Stack grows from HIGH to LOW memory */
```

### 3.1.4. 定义 OS\_TASK\_SW()宏

OS\_TASK\_SW()是一个宏, 它在  $\mu$ C/OS-II 从低优先级任务切换到最高优先级任务时被调用的。任务切换只是简单的将处理器寄存器保存到将被挂起的任务的堆栈中, 并且将更高优先级的任务从堆栈中恢复出来。可采用两种方式定义这个宏, 使用软中断将中断向量指向 OSCtxSW()函数; 或者直接调用 OSCtxSW()函数, 这里我们采用后者(OSCtxSW()函数的实现将在后面介绍):

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

## 3.2 处理器相关部分汇编实现

$\mu$ C/OS-II 的移植需要用户编写三个最基本的汇编语言函数: OSStartHighRdy(), OSCtxSw(), OSIntCtxSw()。它们会共用一些代码, 为了方便阅读将它们写在同一个汇编文件 os\_cpu\_a.S 中。

### 3.2.1 OSStartHighRdy(): 运行优先级最高的就绪任务。

OSStartHighRdy()函数是在 OSStart()多任务启动之后, 负责从最高优先级任务的 TCB 控制块中获得该任务的堆栈指针 SP, 并通过 SP 恢复 CPU 现场以启动最高优先级的任务执行。另外 OSStartHighRdy()还必须在最高优先级任务恢复之前和调用 OSTaskSwHook()之后设置 OSRunning 为 TRUE。其实现代码如下:

技术文章发表, 请尽速发布

2011年8月2日

```
OSStartHighRdy:
  #if OS_TASK_SW_HOOK_EN
    CallFn  OSTaskSwHook // Call OSTaskSwHook ()
  #endif

  li  $a0, #1          ! set OSRunning = TRUE
  la  $a1, OSRunning
  sbi $a0, [$a1]
  ! Switch to highest priority task
  la  $a0, OSTCBHighRdy ! Get highest priority task TCB->stack pointer
  lwi $a1, [$a0]
  lwi $sp, [$a1]      ! Switch to the new stack
  IntlSwitch #1      ! Switch to interruption level 1
  CtxRestore          ! Restore to original task's context
  Iret
```

### 3.2.2 OSCtxSw()和 OSIntCtxSw()

OSCtxSw()是任务优先级切换函数，它的作用是先当前任务的 CPU 现场保存到该任务的堆栈中，然后获得最高优先级任务的堆栈指针，并从该堆栈中恢复此任务的 CPU 现场，使之继续执行，该函数就完成了一次任务切换。

OSIntCtxSw()是中断级的任务切换函数。由于中断可能会使更高优先级的任务进入就绪态，因此为了让更高优先级的任务能立即运行，在中断服务子程序最后会调用 OSIntCtxSw()做任务切换。这样做能够尽快的让高优先级的任务得到相应的处理，保证系统的实时性能。

OSCtxSw()和 OSIntCtxSw()都是用于任务切换的函数，其区别在于，在 OSIntCtxSw()中无需再保存处理器寄存器，因为在 OSIntCtxSw()之前已发生中断，所以可以保证所有的处理器寄存器都被正确地保存到了被中断的任务的堆栈之中。OSCtxSw()和 OSIntCtxSw()实现代码如下：

技术文章发表, 请尽速发布

2011年8月2日

```

OSCtxSw:
    mfsr $a0, $PSW
    mtsr $a0, $IPSW      ! PSW->IPWS
    mtsr $lp, $IPC       ! $lp -> IPC
    CtxSave               ! Save current task's context
    la  $a0, OSTCBCur    ! OSTCBCur->OSTCBStkPtr = $SP
    lwi $a1, [$a0]
    swi $sp, [$a1]

OSIntCtxSw:
#ifdef OS_TASK_SW_HOOK_EN
    CallFn  OSTaskSwHook
#endif
    la  $a0, OSPrioHighRdy ! OSPrioCur = OSPrioHighRdy
    lbi $a1, [$a0]
    la  $a0, OSPrioCur
    sbi $a1, [$a0]
    la  $a0, OSTCBHighRdy ! OSTCBCur = OSTCBHighRdy
    lwi $a1, [$a0]
    la  $a0, OSTCBCur
    swi $a1, [$a0]
    lwi $sp, [$a1]      ! $SP = OSTCBHighRdy->OSTCBStkPtr
    IntlSwitch #1      ! Switch to interruption level 1
    CtxRestore         ! Restore to original task's context
    Iret

```

N1033A-S 处理器定义了四级 ( 0-3 ) 中断，在各级中断的转换时需要保存当前中断层级的寄存器。调用 OSCtxSw() 时，中断将由 0 级 ( 即没有中断 ) 转到 1 级，所以需要将第 0 级的寄存器 PSW 和 PC 保存到第 1 级的寄存器 IPSW 和 IPC 中。CtxSave 和 CtxRestore 两个宏用来保存和恢复任务上下文。需要保存或恢复的寄存器包括 32 个通用寄存器 ( R0-R31 ) 的值、程序计数器(PC) 的值以及处理器状态字寄存器(PSW) 的值。宏 IntlSwitch n 通过修改 PSW.INIT 的值来切换中断层级。CtxSave 和 IntlSwitch 的汇编实现如下 ( 由于 CtxRestore 与 CtxSave 过程类似，这里不做赘述 ):

技术文章发表, 请尽速发布

2011年8月2日

```

! Switch to interruption level \lv
    .macro IntlSwitch lv
        mfsr $a0, $PSW
        li  $a1, #~PSW_mskINTL
        and $a0, $a0, $a1                ! mask PSW.INIL
        ori $a0, $a0, #(\lv << PSW_offINTL) ! Set PSW.INIL
        mtsr $a0, $PSW
        isb
    .endm

! Save current task's context
    .macro CtxSave
        pushm $r0, $r30                ! save r0-r30

        mfusr  $a5, $d0.hi  ! get d0.hi
        mfusr  $a4, $d0.lo  ! get d0.lo
        mfusr  $a3, $d1.hi  ! get d1.hi
        mfusr  $a2, $d1.lo  ! get d1.lo
        mfsr $a1, $IPC      ! get return address
        mfsr $a0, $IPSW     ! get IPSW
        pushm  $a0, $a5

        move $a0, $sp
        andi $a1, $sp, #7
        bnez $a1, 1f
        push $a0                ! adjust sp to 8byte align
1:
        push $a0                ! save sp(r31)
    .endm

```

### 3.3 移植 C 语言编写的几个与操作系统相关的函数

μC/OS-II 有六个与 CPU 相关的函数：OSTaskStkInit()、OSTaskCreateHook()、OSTaskDelHook()、OSTaskSwHook()、OSTaskStatHook()、OSTimeTickHook()，它们被定义在 ucos\_ii.h 中。其中唯一必须移植的函数是任务堆栈初始化函数 OSTaskStkInit()，其它五个函数必须

技术文章发表, 请尽速发布

2011年8月2日

得声明但没必要包含代码。因此这里我们只介绍 OSTaskStkInit(), 其代码的实现如下:

```

OS_STK *OSTaskStkInit(void (*task)(void *pd), void *p_arg, OS_STK *ptos,
INT16U opt)
{
    int i;
    OS_STK *stk = ptos;          /* load stack pointer */
    OS_STK *old_stk;            /* unaligned stack pointer */

    *--stk = (INT32U) 0x01010101UL * 30;    /* R30 */
    *--stk = &_SDA_BASE_;    /* For relax support, set R29($gp) to
_SDA_BASE_ */
    for (i = 28; i >= 1; i--)    /* R 28 ~ R1 */
        *--stk = (INT32U) 0x01010101UL * i;
    *--stk = (OS_STK)p_arg;    /* R0 : argument */

    for (i = 35; i >= 32; i--)    /* d0.hi, d0.lo, d1.hi, d1.lo */
        *--stk = (INT32U) 0x01010101UL * i;
    *--stk = (OS_STK)task;    /* IPC */
    *--stk = (OS_STK) (GET_PSW() | (1UL << PSW_offGIE));

    old_stk = stk;
    *--stk = (OS_STK) ((INT32U)old_stk);    /* Save curent sp */
    if ((INT32U)stk & 0x7)
        *--stk = (OS_STK) ((INT32U)old_stk);    /* adjust unligned sp (R31)
*/

    return stk;
}

```

OSTaskStkInit()在任务创建时被调用,负责初始化任务的堆栈结构并返回新堆栈的指针,使得堆栈看起来就像刚发生过中断并将所有的寄存器保存到堆栈中的情形一样。除了要保存任务的地址、变量的指针以及处理器状态字的值外, Andes N1033A-S 处理器还要求用户保存所有 32 个通用寄存器 (R0-R31)、四个用户寄存器(d0.hi, d0.lo, d1.hi, d1.lo)。还有一点需要注意,在 N1033A-S 处理器中,堆栈指针的地址必须满足 8Byte 对齐,程序最后一段逻辑即将堆栈指针调整到正确的位置,这一点在编写其他代码例如在宏 CtxSave 中同样需要注意。

技术文章发表, 请尽速发布

2011年8月2日

## 4. 结语

基于 AndesStar™ 架构的优势, 可以很容易的实现  $\mu\text{C}/\text{OS-II}$  在 N1033A-S 处理器上的移植。不仅  $\mu\text{C}/\text{OS-II}$ , 其它嵌入式操作系统也可以很方便地移植到 AndesCore™ 相应的处理器上, 例如 Nuclues、FreeRTOS 以及 Contiki。

晶心科技利用 AndesCore™ N1033A-S 高效能的 Audio ISA 和 FPGA 开发平台弹性的设计架构, 基于各种 RTOS, 为客户提供了丰富的软件资源 ( 中间件、优化的函数库、应用实例等 ) 以及完整的多媒体语音解决方案, 从而帮助客户更快地在 Andes 平台上进行产品开发。

( 如您有技术方面的疑问, 欢迎来信至 [zhoujie@andestech.com.cn](mailto:zhoujie@andestech.com.cn) 与我们进行讨论。)

技术文章发表, 请尽速发布

2011年8月2日

.....关...于...晶...心.....



为适应嵌入式系统应用的快速成长, 2005年晶心科技创立于新竹科学园区, 致力于开发基于 32 位处理器的系统芯片设计平台 (**Processor-based SoC Platforms**)。晶心科技拥有来自联发科、智原科技及其他投资人的充裕资金, 以及来自大学研究的互动, 已成功建立起亚太唯一的具备软硬件协同设计能力和系统集成能力且专注于支持系统芯片开发的公司。

随着电子产业产品的日趋多功能化, 更多的厂商开始要求处理器和设计平台具备更好的集成性、可扩展性、设计弹性以及高效率、低成本与低功耗。这样的复杂度已经超越了传统供应厂商所能提供的解决方案。晶心科技考虑了未来电子系统层面 (ESL) 更加广泛的设计要求, 以**创新的弹性可配置平台的 (Configurable Platforms)**, **搭配独特的软硬件 IP**, 来满足未来客户对产品高品质以及快速上市的要求。过去几年, 晶心仍然处于开拓自主研发 CPU 平台架构的市场阶段, 目前的 CPU 平台包括主流、低、中、高階等级的 **N8、N9、N10 和 N12 系列**都已成熟到位, 能满足客户的各种应用需求。

关于晶心科技提供的 32 位 CPU IP 以及 ESL 系统开发工具, 欢迎访问晶心科技网站: [www.andestech.com](http://www.andestech.com)

.....新...闻...联...络...人.....

市场部 Janine 徐家玲经理

电话: 886-3-6668300 ext. 614

手机: 886-932-315015

E-mail: [Janine@andestech.com](mailto:Janine@andestech.com)

Web: [www.andestech.com](http://www.andestech.com)