eCos 操作系统针对并行 S698P-SOC 的启动初始化过程

许学艺[1]，龚永红[1]，梁宝玉[1]，富宏亚[2]，颜军[1]

（1. 欧比特（珠海）软件工程有限公司，广东 珠海 519080；2. 哈尔滨工业大学机电工程学院）

摘要：本文介绍了实时操作系统 eCos 关于对称多处理器的支持，着重讨论了 eCos 在 S698P-SOC 上的启动初始化过程。

关键词：S698P-SOC；eCos；SMP；SPARC；

The Boot Procedure Of eCos On The S698P-SOC

Xu Xueyi[1], Gong Yonghong[1], Liang Baoyu[1], Fu Hongya[2], Yan Jun[1]

(1.Orbita Software Engineering Inc. Guangdong, Zhuhai, 519080, 2. School of Mechatronics Engineering, HARBIN Institute of Technology)

Abstract: The article introduces the real-time operating system eCos how to support the SMP, and mainly describes the boot procedure of the eCos on the S698P-SOC.

Keyword: S698P-SOC, eCos, SMP, SPARC

1 引言

SMP 的全称是"对称多处理"（Symmetrical Multi-Processing）技术，是指在一个计算机上汇集了一组处理器(多 CPU),各处理器之间共享同一个操作系统、内存子系统、总线结构和 I/O 系统等。在这种架构中，一台计算机不再由单个处理器组成，而同时由多个处理器运行操作系统的单一复本，并共享内存和一台计算机的其他资源。系统将任务队列对称地分布于多个处理器之上，从而极大地提高了整个系统的数据处理能力。所有的处理器无主从之分，都可以平等地访问内存、I/O 和外部中断。在对称多处理系统中，系统资源被系统中所有处理器共享，工作负载能够均匀地分配到所有可用处理器之上。并且因为结构共享存储器、统一地址空间，系统编程比较容易。

S698P-SOC 处理器是欧比特（珠海）软件工程有限公司的 S698 系列的其中一款处理器，是一款高性能的、SPARC V8 架构的、32-bit RISC 嵌入式 SMP 微处理器。图 1 是 S698P 处理器的内部结构图。

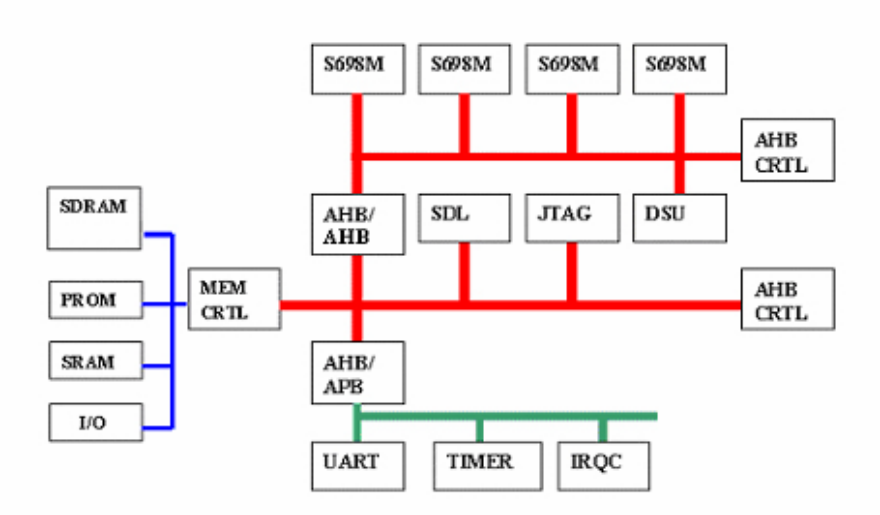


图 1　S698P-SOC 处理器内核结构

S698P-SOC 各处理器之间的通信是通过多核中断控制器(MP IRQCTRL)的中断来实现的，其结构如图 2 所示：

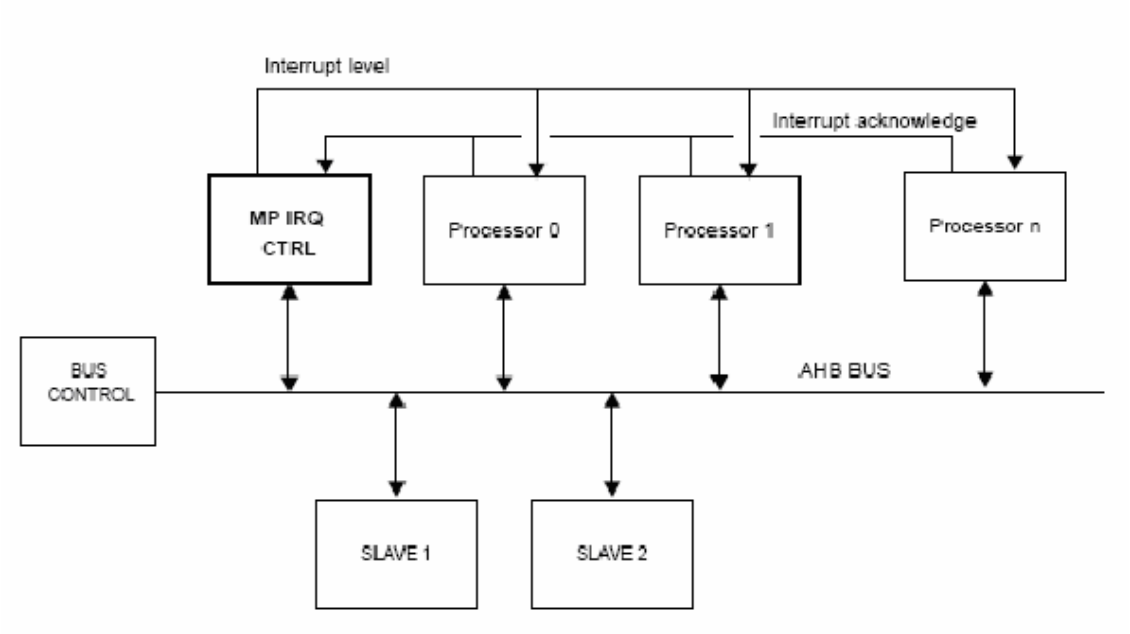


图2　S698P-SOC内部通信结构

S698P-SOC中的每个处理器都可以通过多核中断控制器向其它处理器发中断请求，每个处理器都可以响应其它处理器的中断请求，并且在多核中断控制器其中一个称为多处理器状态寄存(Multi-processor status register)的后16位(STATUS[15:0]),写入1，其相应的处理器启动。

31 — 28	16 15 — 0
NCPU	“000…0”	STATUS[15:0]

eCos(Embedded Configurable Operating System，嵌入式可配置操作系统)是一种针对 16 位、32 位和 64 位处理器的可移植的开源嵌入式实时操作系统。eCos 可在选定的处理器架构和平台上提供 SMP 支持。

eCos 在它的 SMP 支持中加入了一些目标硬件限制，包括以下内容：

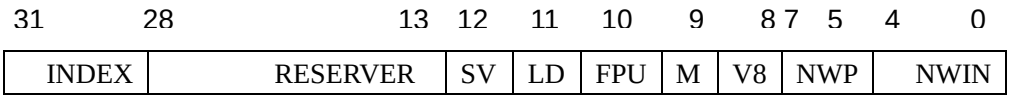
- 最大支持处理器个数为 8，典型是 2 或 4；

- 硬件必须提供测试并设置或者比较并交换指令同步机制。eCos 内核使用这些硬件指令实现螺旋锁；
- 不使用 cache，或所有的处理器共享系统 cache，或硬件维护 cache 的一致；
- 所有的处理器共享的内存对所有的处理器的地址是一致的；
- 每个处理器都可以访问任何外设；
- 中断控制器必须将中断信号传送给指定的处理器；
- 允许一个处理器上的事件导致另一个处理器的重新调度（需要允许系统中一个处理器中断另一个处理器的机制）；
- 在某个处理器上运行的软件必须能识别其所运行的处理器；

S698P-SOC 满足 eCos 对 SMP 的硬件限制条件：

- S698P-SOC 目前支持 4 核；
- S698P-SOC 内有 swapa 指令，可以提供测试并设置同步机制，利用该条指令在 eCos 实现 HAL_TAS_SET 和 HAL_TAS_CLEAR 两个宏定义，eCos 在 HAL 层以这两个宏来实现螺旋锁；
- 在 SMP 系统中，各 CPU 通过 Cache 访问内存数据时，要求系统必须经常保持内存中的数据与 Cache 中的数据一致，若 Cache 的内容更新了，内存中的内容也应该相应更新，否则就会影响系统数据的一致性。S698P-SOC 用总线侦听技术(snooping)维护了 4 个处理器的 data cache 的一致性；
- S698P-SOC 所以的处理器共享系统 sram 和 sdram，其编址是一致的，使编程容易；
- S698P-SOC 中所有外设的寄存器，所有处理器都能访问到；所有外设的中断，都能引起所有处理器的中断处理，并且可以通过设置各处理器的中断屏蔽寄存器指明哪个外设的中断由哪个处理器处理；
- S698P-SOC 各个处理器之间通过多核中断控制器(MP IRQCTRL)的中断来通信，允许将中断传递到系统中特定的处理器，该处理器执行中断处理函数，根据消息的内容执行优先级抢占重新调度或者时间片轮转调度；
- S698P-SOC 各个处理器均有一个称为%asr17 的寄存器，其 31-28 位(下图 INDEX 部分)，指明当运运行的是哪个处理器,并由此实现 CYG_KERNEL_CPU_THIS()宏定义，该宏定义得到当前是在哪个处理器上运行，程序可以根据这个宏定义作相应的处理。

%asr17 寄存器如下所示:



SMP 系统的启动顺序是不同的。其中定有一个处理器，称为主处理器（在 S698P-SOC 中，编号为 0 的是主处理器），处理启动初始化顺序；而其它处理器，称为附属处理器，或者被 HAL 置于待机状态，或者被 HAL 置于空闲循环。当应用程序调用 cyg_sheduler_start 函数,附属处理器启动运行。因此，S698P-SOC 处理器上的启动初始化过程必然会根据%ars17 寄存器分主从处理器分别初。

本文将具体介绍 eCos 在 S698P-SOC 处理器上的启动初始化过程，使开发人员对 eCos 在 S698P-SOC 多核处理器运行有一个全局的了解。

2 启动初始化过程

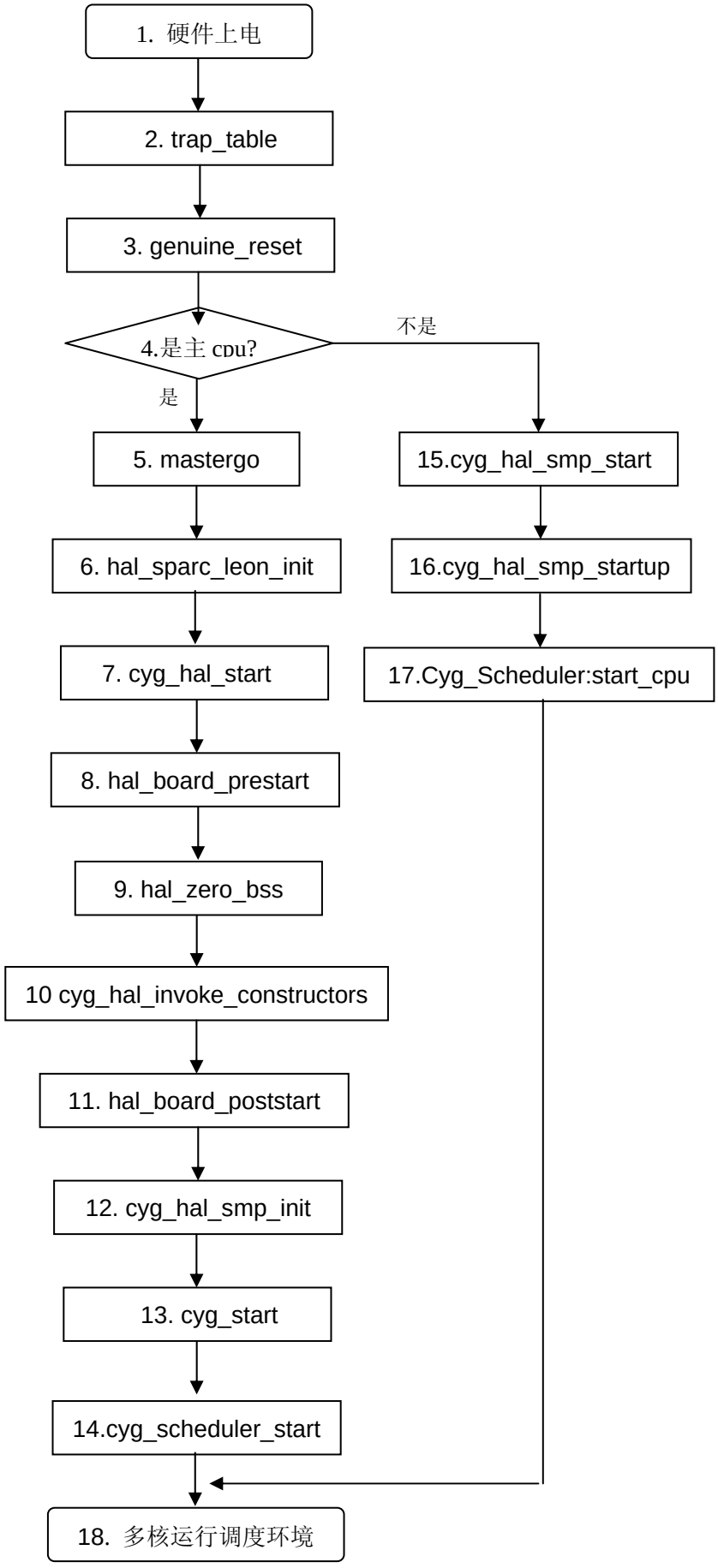

1. 硬件上电
2. trap_table
3. genuine_reset
4.是主 cpu?
不是
是
5. mastergo
15.cyg_hal_smp_start
6. hal_sparc_leon_init
16.cyg_hal_smp_startup
7. cyg_hal_start
17.Cyg_Scheduler:start_cpu
8. hal_board_prestart
9. hal_zero_bss
10 cyg_hal_invoke_constructors
11. hal_board_poststart
12. cyg_hal_smp_init
13. cyg_start
14.cyg_scheduler_start
18. 多核运行调度环境

图 3　启动初始化过程

下面详细说明图 3 所示的 HAL 启动过程中的每个操作：

操作 1：eCos 上电或者软复位后，S698P-SOC 只启动 cpu0,其它 CPU 处于 power down 状态。系统的程序指针会自动指向地址 0。后程序搬移到 0x40000000 地址开始的内存区。运行的第一个程序是位于体系结构抽象层 arch 子目录中的 vector.S 文件。

操作 2：vector.S 执行代码一开始是定义了 4K 字节的中 trap_table，该表包括 256 组 trap。其中比较重要的有 0 号 trap：硬件复位中断向量；5 号和 6 号 trap；寄存器窗口上溢和下溢处理中断向量；17－31 号 trap：外设中断 1-15;128 号：操作系统软件软中断处理向量。S698P-SOC 所有的处理器的中断向量表是一样的。只是能够通过设置中断相关寄存器，指定哪个处理器响应哪个外设中断，而其它处理器不响应。

操作 3：S698P-SOC 的处理器 0(其它处理器也一样)根据 0 号 trap,跳转到 genuine_reset:设置%tbr 寄存器指定中断向量表的位置为 0x40000000 开始的 4K BYTE；设置%wim 指明处理器的寄存器窗口数为 8；清指令 cache 和数据 cache；设置总线侦听使能（snoop enable），使各个处理器的数据 cache 能一致。

操作 4：S698P-SOC 的所有处理器根据%asr17(处理器配置寄存器)的 31-28 位，得到处理器的索引号，指明当前是运行在哪个处理器。执行代码如下：

```
rd        %asr17,%g2
srl       %g2,28,%g2
```

%g2 为当前运行处理器的索引号，主处理器为 0，其它处理器为大于 0 的数字。

操作 5：如果是处理器 0,跳转到 mastergo；

操作 15：如果是 S698P-SOC 中的其它处理器（其它的处理器是由 cpu0 在 mpstatus（多处理器状态寄存器寄存器）写入相应的位启动）,程序执行 cyg_hal_smp_start,设置%tbr，设置中断堆栈；设置%psr；

操作 16：S698P-SOC 中的其它处理器调用 cyg_hal_smp_startup 函数，设置该处理器已运行标志（在变量 cyg_hal_smp_cpu_running 相应位置位）；检查是否内核设置了总线侦听使用，否则打印告警信息；系统运行处理器数量加 1，指明多启动了一个核，以后系统调度可以使用该核；

操作 17：S698P-SOC 中的其它处理器调用 cyg_kernel_smp_startup 函数。在该函数内先用 Cyg_Scheduler::lock()锁调度器，再调用 Cyg_Scheduler::start_cpu()，该函数如下：

```
void Cyg_Scheduler::start_cpu()
{

  #ifdef CYGPKG_KERNEL_SMP_SUPPORT
    Cyg_Interrupt * intr = new( (void *)&cyg_sched_cpu_interrupt[HAL_SMP_CPU_THIS()] )
    Cyg_Interrupt( CYGNUM_HAL_SMP_CPU_INTERRUPT_VECTOR( HAL_SMP_CPU_THIS() ),
                   0,
```

```
                    0,
                    cyg_hal_cpu_message_isr,
                    cyg_hal_cpu_message_dsr
                    );

    intr->set_cpu( intr->get_vector(), HAL_SMP_CPU_THIS() );
    intr->attach();
    intr->unmask_interrupt( intr->get_vector() );
#endif
  register Cyg_Thread *next = scheduler.schedule();
  clear_need_reschedule();                  // finished rescheduling
  set_current_thread(next);                 // restore current thread pointer
  #ifdef CYGVAR_KERNEL_COUNTERS_CLOCK
    CYG_REFERENCE_OBJECT( Cyg_Clock::real_time_clock );
  #endif
    HAL_THREAD_LOAD_CONTEXT( &next->stack_ptr );
}
```

该函数先生成 Cyg_Interrupt 中断处理类的一个实例，设置该中断在指定的处理器处理，再用 attach()函数在指定的处理器通信中断（目前为 14 号中断）挂中断处理函数 isr 和 dsr；再开放指定的中断。最后从调度器得到应该调度的一个任务，用 HAL_THREAD_LOAD_CONTEXT 载入该任务的上下文环境。从此，该处理器进入了多核运行调度环境；

操作 6：S698P-SOC 中的处理器 0 调用 hal_sparc_leon_init 函数，在该函数调用 ahbslv_scan 函数扫描 IP 核在 AHB 总线上的 slave 设备。调用 apbslv_scan 函数扫描 IP 核在 APB 总线上的 slave 设备。初始化中断控制器、定时器、串口。设置处理器 0 栈地址、%wim、%psr。如果不使用 DSU，则设置内存配制寄存器 1、内存配制寄存器 2、sdram 控制寄存器；

操作 7：S698P-SOC 中的处理器 0 调用 cyg_hal_start 函数，该函数作 S698P-SOC 系统的硬件初始化操作。

操作 8：S698P-SOC 中的处理器 0 在 cyg_hal_start 函数中调用 hal_board_prestart 函数,执行板子开始运行前的处理动作；

操作 9：S698P-SOC 中的处理器 0 在 cyg_hal_start 函数调用 hal_zero_bss 函数，清 bbs 段；

操作 10：接下来在 cyg_hal_start 函数调用 cyg_hal_invoke_constructors 函数，执行各个内嵌的构造函数；

操作 11：cyg_hal_start 函数调用 hal_board_poststart 函数，执行板子开始运行后的处理动作，在应该函数内调用 amba_init 函数和开放系统中断。其中，在 amba_init 函数中执行的 amba 总线的扫描，找出中断控制寄存器和定时器的起始地址；操作系统以后可以根据中断控制寄存器和定时器的起始地址对两者的寄存器进行配置。目前操作系统所需要的系统实时时钟由第一个定时器 Timer1，中断编号为 8。定时器的初始化函数如下：

```
void hal_sparc_leon3_clock_init(cyg_uint32 period) {
   cyg_hal_sparc_clock_period = (period);

   if (LEON3_GpTimer_Regs) {
      //设置定时器周期长度
      LEON3_BYPASS_STORE_PA(&LEON3_GpTimer_Regs ->e[0].val,(period));
      LEON3_BYPASS_STORE_PA(&LEON3_GpTimer_Regs ->e[0].rld,(period));
      LEON3_BYPASS_STORE_PA(&LEON3_GpTimer_Regs ->e[0].ctrl,0);
      //设置定时器使能并自动重启动
      LEON3_BYPASS_STORE_PA(&LEON3_GpTimer_Regs ->e[0].ctrl,
                                          LEON3_GPTIMER_EN |
                                          LEON3_GPTIMER_RL |
                                          LEON3_GPTIMER_IRQEN |
                                           LEON3_GPTIMER_LD);
   } else {
      diag_printf("Clock init failed");
   }
}
```

操作 12：S698P-SOC 中的处理器 0 调用 cyg_hal_smp_init 函数，检查处理器 0 是否打开了总线侦听机制，计算系统所有处理器的中断堆栈地址，设置变量 cyg_hal_smp_cpu_running 第 0 位为 1，表明处理器 0 已经启动，变量 cyg_hal_smp_cpu_running_count 加 1，表示系统中启动的处理器加了一个。
检查打开总线侦听的代码如下：

```
      if (cfg & ASI_LEON3_SYSCTRL_CFG_SNOOPING) {
         sparc_leon3_enable_snooping();
      }
      else {
         leon3smp_diag_printf("Caches disabled due to lack snooping\n");
         sparc_leon3_disable_cache();
       }
```

代码首先检查是否 IP 核配置了 snooping，有则打开。没有则告警并关闭 data cache；

操作 13：S698P-SOC 中的处理器 0 调用 cyg_start 函数，该函数是 ecos 的用户程序入口函数。用户程序至少要在这个函数内生成一个任务，并且这个任务处于可运行状态；

操作 14：用户程序在 cyg_start 内调用 cyg_scheduler_start 函数,该函数调用调度器类的 start() 函数。在 start 函数内，处理器 0 调用 cyg_hal_cpu_reset 函数启动其它从处理器,并且调用 Cyg_Scheduler 调度器类的 start_cpu 函数，如上述 17 所示，处理器 0 也加载了处理器之间通信的中断处理函数，并执行第一个任务；

总之，主处理器执行 1->2->3->4->5->6->7->8->9->10->11->12->13->14 后，再启动其它三个从处理器；从处理器启动后，执行 1->2->3->4->15->16->17->18。两者都完成后，系统进入了多核运行调度环境，eCos 操作系统开始根据用户程序生成的任务以及之间的相互关系进行调度。至此，eCos 操作系统针对 S698P-SOC 的启动初始化过程结束。

3 结束语

本文主要研究了 eCos 操作系统在 S698P-SOC 的启动初始化过程，并移植代码已在处理器上经过实际测试，系统稳定可靠，可运行多任务应用程序。本文所具体讨论的启动过程不仅有助于开发人员了解 eCos 操作系统的运行，而且有助于了解 S698P-SOC 多核处理器的工作原理，方便开发人员使用 S698P-SOC 多核处理器进行军用民用领域项目开发。

参考文献：

[1] S698P-SOC eCos Reference Manual. 欧比特（珠海）软件工程有限公司，2006

[2] eCos 使用说明. 欧比特（珠海）软件工程有限公司，2006

[3] 嵌入式操作系统 eCos 在 S698 系列处理器上的移植. 欧比特(珠海)软件工程有限公司，2006

[4] Anthony J.Massa. 嵌入式可配置实时操作系统 eCos 软件开发(Embedded Software Development with eCos). 北京航空航天大学出版社，2006

[5] John D, Carpinelli. 计算机系统组成与体系结构 (Computer Systems Organization & Architecture). 人民邮电出版社，2003