VxWorks操作系统在 OMAP平台上的移植

李 永 孙士明 王爱国

(中国石油大学 (华东) 计算机与通信工程学院 东营 257061)

摘要: VxWorks操作系统是美国风河 (WindRiver) 公司的一款嵌入式实时操作系统 (RTOS), 被广泛应用于通信、军事、航空航天等实时性要求极高的领域中; 而 OMAP处理器则兼容 DSP内核和 ARM处理器, 具有广泛的应用领域。本文基于 OMAP5910处理器架构和 VxWorks操作系统的体系结构, 将 VxWorks移植到 OMAP平台上。即根据 OMAP5910的存储器地址空间映射关系和硬件资源配置, 修改 VxWorks操作系统的 BSP相关文件, 包括系统异常处理、CPU初始化、串口驱动和网络驱动的修改并加载等, 并编译成内核镜像, 将其移植到 OMAP5910的 ARM端。最后通过相应的测试点检测移植结果。

关键词: 嵌入式系统 VxWorks OMAP BSP 移植

Transplant of VxWorks Operating System to the OMAP Platform

LI Yong, SUN Shiming, WANG Aiguo

(College of Computer and Communication Engineering, China University of Petroleum (Ease China), Dongying, 257061, China)

Abstract: VxWorks operating system is an embedded real-time operating system (RTOS) and sold by Wind River Systems of Alameda, California, USA. It is applied in the fields which ask for high real-time features like signal communication, military, aviation, space flight and so on; On the other side, OMAP processor contains DSP kernel and ARM processor, and it has widespread application fields to use. This article transplants VxWorks to OMAP platform according to the structures of OMAP5910 processor and VxWorks operating system. First according to the memory address space mapping relations and hardware resource configure of OMAP5910, BSP of VxWorks is modified, including system exceptional operations, CPU initialization, serial driver, network driver and so on. Then the kernel image is compiled, and transplanted to the ARM side of OMAP5910. At last the system is verified correct by some corresponding test points.

Keywords: Embedded System, VxWorks, OMAP, BSP, transplant

VxWorks操作系统是美国风河 (WindRiver) 公司的一款嵌入式实时操作系统 (RTOS)。VxWorks因其高性能内核、友好的开发环境、卓越的可靠性和实时性被广泛应用于通信、航空航天等实时性要求极高的领域中。基于 VxWorks风河公司又推出一套实时操作系统开发环境, 即 Tornado。首先 Tornado保证即使目标机不同, 开发者也能使用一致的图形用户接口; 其次 Tornado所有的开发工具都驻留在其平台上, 这样对嵌入式系统开发非常方便; 最后所有开发工具都通过一个中央服务器 (Target Server) 处理与目标机的通讯。因此无论连接方式是网络、串口、ICE仿真器、ROM仿真器或客户设计的调试通道, 所有工具均可使用。

OMAP5910处理器, 是由 TI应用最为广泛的 TMS320C55XDSP内核与低功耗增强型 ARM925微处理器组成的双核应用处理器。55X系列 DSP可提供对低功耗应用的实时多媒体处理的支持, 而 ARM925MPU可满足控制和接口方面的处理需要。基于双核结构, OMAP5910具有低功耗和强运算能力双重优点。一方

本文于 2010-05-04收到。

面产品低功耗且高性能，另一方面其 ARM端采用开放式的软件设施，支持广泛的操作系统。

基于 OMAP5910处理器架构和 VxWorks操作系统的体系结构，提出了 VxWorks操作系统在 OMAP5910平台上的移植方案。通过修改 VxWorks操作系统的 BSP相关文件，将其移植到 OMAP5910的 ARM端，并通过相应的测试检测是否成功。

1　VxWorks体系结构

VxWorks提供了一种高级体系结构[1]，即其应用程序代码独立于硬件。基于这种思想，VxWorks具有很强的可移植性。其原因是它采用了模块层次化设计，将所有硬件的功能函数分别放到一系列库中提供调用。这些库被称为板级支持包 BSP (Board Support Package)[2]。BSP包含了一系列程序段，它们为 VxWorks操作系统提供了与硬件环境之间的主要接口，图 1包括了 VxWorks操作系统的各种组件，指明了依赖于硬件和独立于硬件的各个模块。

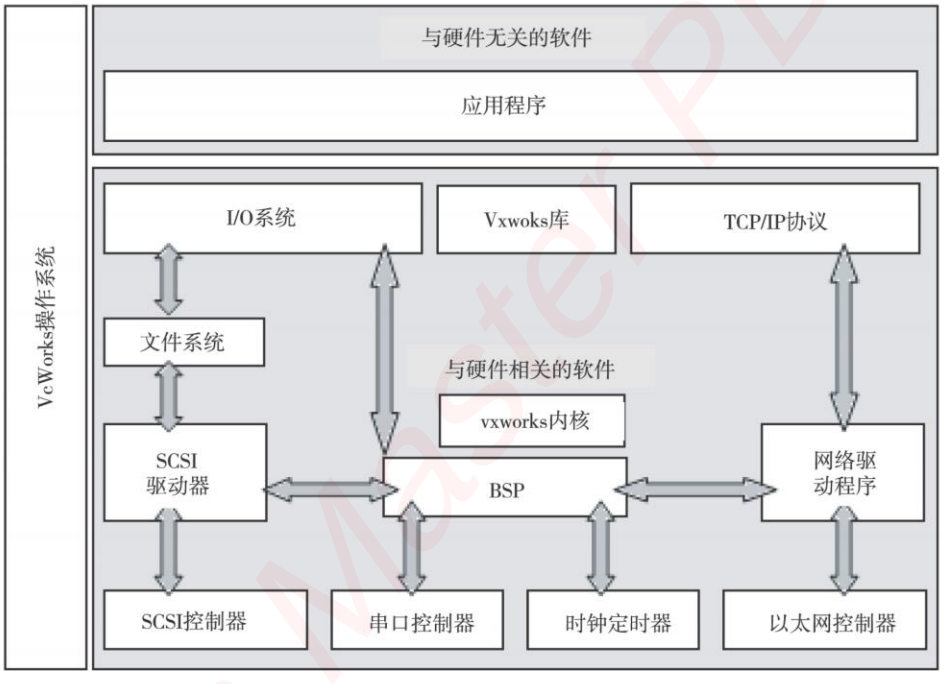

图 1　VxWorks体系结构

虽然风河公司如今已提供支持多种硬件的 BSP，但是随着多种新款 CPU架构的出现，还是无法满足对适用于各种目标板 BSP的需求，其中就包括 OMAP5910的 BSP。所以要将 VxWorks移植到 OMAP5910平台，需要修改 BSP中相应的文件，使之适用于 ARM925的硬件结构。

2　移植过程

Tornado集成了 VxWorks BSP开发工具包，该工具包可以帮助开发者创建、注释以及测试新创建的 BSP、组件和项目工程。首先利用 Tornado创建一个类似 BSP模板[3]；之后利用 Beyond Compare将模板中的文件与其它已经成功移植的 BSP文件进行比较，例如和基于 S3C2410架构的 BSP文件进行比较[4]；然后根据 OMAP5910的存储器地址空间映射关系以及 OMAP5910开发板上的硬件资源配置[5]，修改 BSP中的相应的文件；最后再修改设备驱动并进行烧写和测试。

2.1　异常处理

OMAP5910的 ARM925T核共支持 7种异常，每种异常都有固定的优先级和相应的处理器特权模式。

在处理 OMAP5910异常向量时，在 Flash存储器的起始地址硬编码异常入口，并在 RAM的起始地址仿 VxWorks建立异常向量表。当异常发生时，经 Flash存储器起始地址的异常入口跳转到 RAM中异常向量表入口，再调用 VxWorks提供的异常处理函数。异常向量表如表 1所示。

表 1 OMAP5910异常向量表

Address	Exception	Mode in Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

按照该异常处理流程，首先要在 Flash存储器的起始处添加硬件中断入口，即在 romInit.s文件起始处添加如下代码：

```
_romInit:
B       cold    /*上电复位后执行的第一条指令，也可看作是复位向量*/
B       _romUndef
B       _romSwi
B       _romPrefetch
B       _romDataAbort
B       cold    /*_romReserved*/
B       _romIRQ
B       cold    /*_romFIQ*/
```

然后定义相应的异常处理函数，下面以 IRQ中断为例说明中断处理函数的作用，在 romInit.s中添加如下代码：

```
_ARM_FUNCTION_(romIRQ)
_romIRQ:
sub sp, sp, #4              /*减少 sp，保存跳转地址*/
stmfd sp!, {r0}             /*将工作寄存器压入堆栈*/
ldr r0, LMYM_promIRQ        /*将存放中断程序入口地址的内存地址放入 r0*/
ldr r0, [r0]                /*将中断程序入口地址放入 r0*/
str r0, [sp, #4]            /*将中断程序入口地址压入堆栈*/
ldmfd sp!, {r0, pc}         /*将工作寄存器和中断程序入口地址弹出到 r0和 PC*/
```

其中 L$_promIRQ被定义为

```
L$_promIRQ:
.long OMAP_EXC_BASE + 20
#define OMAP_EXC_BASE 0x10000100    /*SDRAM的起始地址*/
```

最后在初始化硬件即进入 sysHwInit()函数之前，先在 RAM空间建立异常向量表。该向量表和 VxWorks建立的向量表只是位置不同。具体代码如下：

```
void OMAPExcVecSet (void)
{ int i;
  i = (int) &excEnterUndef;
  *((volatile int*)(OMAP_EXC_BASE + 0x0)) = i;
  i = (int) &excEnterSwi;
```

```
    *  ( (volatile int*) (OMAP_ EXC_ BASE + 0x4)) = i;
    i = (int) &excEnterPrefetchAbort;
    *  ( (volatile int*) (OMAP_ EXC_ BASE + 0x8)) = i;
    i = (int) &excEnterDataAbort;
    *  ( (volatile int*) (OMAP_ EXC_ BASE + 0xc)) = i;
    i = (int) &intEnt;
    *  ( (volatile int*) (OMAP_ EXC_ BASE + 0x14)) = i;
return; }
```

2.2　CPU初始化

由于OMAP5910的寄存器与S3C2410的差别很大[6]，因此在omap5910.h文件中根据OMAP5910的datasheet文档更新CPU寄存器并替换S3C2410.h文件[7]。在系统上电时，首先要做的就是初始化CPU并配置相关寄存器。代码如下：

```
/*选择 ARM925T模式 */
mov r1, #0x81                  /* 设置 ARM925T配置参数 */
mcr p15, 0, r1, c15, c1, 0   /* 写入 ARM925T配置寄存器 */
/*关闭看门狗定时器 */
ldr     r1, =OMAP_ WDT_ TIMER_ MODE            /* 0xfffec808 */
mov     r2, #OMAP_ WDT_ TIMER_ MODE_ INIT1     /* 0x00F5 */
str   r2, [r1]
ldr     r1, =OMAP_ WDT_ TIMER_ MODE            /* 0xfffec808 */
mov     r2, #OMAP_ WDT_ TIMER_ MODE_ INIT2     /* 0x00A0 */
str     r2, [r1]
/* flush v4 I/D caches */
mov   r0, #0
mcr   p15, 0, r0, c7, c7, 0   /* 清空高速缓存和写缓存 */
mcr   p15, 0, r0, c8, c7, 0   /* 清空 TLB */
/* disable MMU stuff and caches */
mrc   p15, 0, r0, c1, c0, 0
bic   r0, r0, #0x00002300   /* clear bits 13, 9: 8 (--V- --RS) */
bic   r0, r0, #0x00000087   /* clear bits 7, 2: 0 (B--- -CAM) */
orr   r0, r0, #0x00000002   /* set bit 2 (A) Align */
orr   r0, r0, #0x00001000   /* set bit 12 (I) I-Cache */
mcr   p15, 0, r0, c1, c0, 0
```

以上代码是通过清零c1寄存器的bit[13]，使系统选择异常中断向量表的位置为0x00000000-0x0000001c（OMAP5910支持高端异常中断向量表）；清零bit[9:8]，使在MMU启用时用作系统保护；清零bit[7]，使系统选择little-endian内存模式；清零bit[2:0]，从而禁止MMU、地址对齐检查功能以及数据cache；置位bit[2]，从而使能地址对齐检查功能；置位bit[12]，从而使能指令cache。

```
mov   r1, #0                          /*屏蔽进程标识符寄存器 c13 */
mcr   p15, 0, r1, c13, c0, 0
/*设置 CPU为 SVC32模式并禁止 IRQ和 FIQ中断 */
mrs   r1, cpsr                        /*读取 cpsr寄存器的值到 r1 */
bic   r1, r1, #0x0000001f
orr   r1, r1, #(0x000000d3 | IRQ_ DISABLE_ Bit | FIQ_ DISABLE_ Bit)
msr   cpsr, r1
/*屏蔽所有的中断源 */
```

```
mov   r1, #0xffffffff
ldr   r2, =OMAP_ IH1_ MIR
str   r1, [r2]                          /*屏蔽所有一级中断源*/
ldr   r2, =OMAP_ IH2_ MIR
str   r1, [r2]                          /*屏蔽所有二级中断源*/
/*设置系统时钟*/
ldr   r2, =DPLL_ CTL                    /*DPLL控制寄存器地址*/
mov   r1, #OMAP_ DPLL_ CTL_ INIT        /*0x0010, 设置DPLL为lock模式*/
str   r1, [r2]
/*设置MPU - Clock - Control Register*/
ldr   r2, =ARM_ IDLECT2
ldr   r1, =ARM_ CKCTL_ INIT             /*0x03DF, 使能所有的时钟*/
str   r1, [r2]
ldr   r2, =ARM_ RSTCT1
ldr   r1, =ARM_ RSTCT1_ INIT            /*0x0002, 使能DSP*/
str   r1, [r2]
ldr   r2, =ARM_ RSTCT2
ldr   r1, =ARM_ RSTCT2_ INIT            /*0x0001, PER reset*/
str   r1, [r2]
/*设置EMIFS配置寄存器*/
ldr   r2, =EMIFS_ CONFIG
mov   r1, #EMIFS_ CONFIG_ INIT/*0x00000010, 使Flash.RDY信号为高电平*/
str   r1, [r2]
/*设置EMIFF SDRAM配置寄存器*/
ldr   r2, =EMIFF_ MRS
ldr   r1, =EMIFF_ MRS_ INIT             /*0x00000037*/
str   r1, [r2]
ldr   r2, =EMIFF_ SDRAM_ CONFIG
ldr   r1, =EMIFF_ SDRAM_ CONFIG_ INIT0x000100F4*/
str   r1, [r2]
```

至此，CPU的初始化工作已完成。以上代码全部在 romInit.s文件中，因此上电后首先执行这部分代码。

2.3 串口驱动

2.3.1 串口设备层次

VxWorks的串行设备驱动不同于一般设备驱动。一般的设备都是在系统启动时调用 xxDrv()来安装驱动程序表。然后在应用层使用设备时直接通过 fd ->设备列表 ->驱动程序表的顺序调用相应的驱动函数即可。而 VxWorks串行设备的层次关系则不同，它采用了3层抽象的软件结构：标准 I/O库（ioLib）->tty库（ttyDrv/tyLib）->底层 SCC驱动（xxDrv），如图2所示。

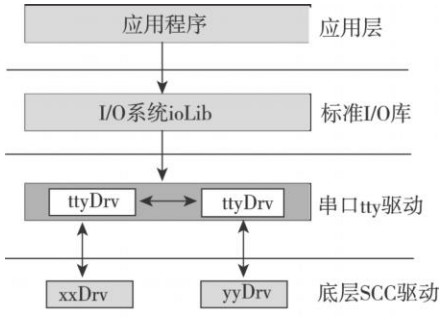

图2 VxWorks串口结构

从图中可看出串行设备驱动 xxDrv并不是直接和 I/O系统交互，实际上内核启动时在驱动程序表中安装的不是 xxDrv函数，而是 ttyDrv/tyLib提供的函数。串口 tty驱动（ttyDrv/tyLib）使 I/O系统独立于具体 SCC驱动，保证代码可复用和统一界面，而 SCC驱动（xxDrv/yyDrv）处理和底层硬件有关的部分。

2.3.2　tty驱动

创建 tty设备包括驱动程序初始化和创建设备，这两步都在 usrRoot（）中完成。创建完 tty设备后，用户可以调用 ioLib中的 write（）、read（）函数对串口进行读写操作。如图 3所示。

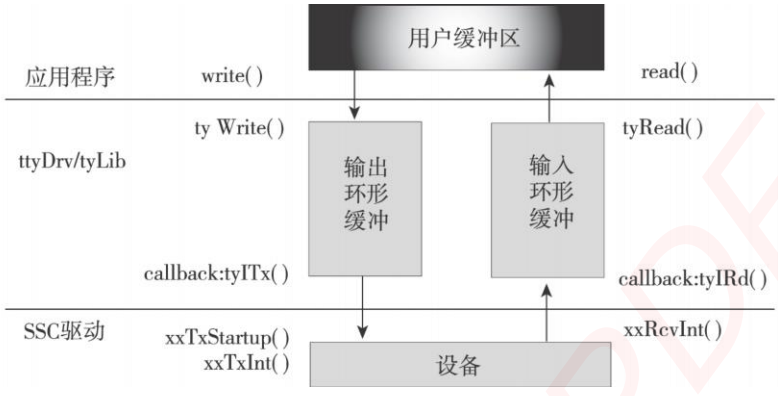

图 3　tty输入输出

2.3.3　修改底层 SCC驱动

串口 tty驱动属于系统提供的一般化代码，几乎不需要修改。但底层硬件相关的部分是由 SCC驱动处理，因此必须根据自己的硬件环境修改 SCC驱动程序。首先分析 OMAP5910的串口电路原理图可知，其开发板上的两个串行接口分别与 OMAP5910芯片的 UART1和 UART3接口连接。为了修改以及测试方便，在 BSP中只对 UART1的相关寄存器进行初始化配置。这里用到 VxWorks中用于初始化串口的数据结构 xx_CHAN，结构如下：

```
typedef struct
{    SIO_CHAN        sio;                /*标准 SIO_CHAN 结构 */
   STATUS        (*getTxChar)();
   STATUS        (*putRcvChar)();
   void *        getTxArg;
   void *        putRcvArg;
   UINT32 *          regs;               /* UART口寄存器 */
   UINT8         intLevelRx;             /*接收设备中断的优先级 */
   UINT8         intLevelTx;             /*发送设备中断的优先级 */
   UINT32        regDelta;               /*寄存器地址空间 */
   uint_t        options;                /*硬件选项 */
   int           mode;                   /*当前模式 (中断或轮询) */
   int           baudRate;               /*输入波特率 */
} OMAP_CHAN;
```

SCC驱动初始化分两步。第一步在内核启动之前和 usrInit（）初始化后，串口 SCC被复位、禁止中断、串口能通过查询方式访问、实现系统级调试。通过 sysSerialHwInit（）初始化设备描述符，串口结构 OMAP_CHAN赋值并连接具体串口参数到 VxWorks的 I/O系统。然后再调用 OMAPDevInit（）初始化串口功能指针 pChan->sio.pDrvFuncs=&OMAPSioDrvFuncs，并调用 OMAPIoctl（）对串口芯片的参数进行设置，包括 8位数据位、无奇偶校验、1个停止位、选择串口通信模式为轮询方式、初始化波特率等；第二步是在根任务 usrRoot（）中进行的第二次初始化，使串口可以以中断方式实现对 tty库的底层支持。在 sysHwInit2（）中调用 intLibInit初始化 intVecTable。OMAPDevInit（）初始化中断驱动，调用 intConnect（）将 sysClkInt（）和 sysAuxClkInt（）的入口地址写入 intVecTable，sysSerialHwInit2（）函数连接串口中断。sysSerialHwInit2（）通过 intConnect（）把串口的中断处理程序 OMAPIntTx（）和 OMAPIntRcv（）连接接到相应的中断向量上，并由 intEnable（）开启两个中断。调用 OMAPDevInit2（）完成对串口的最终配置由轮询模式转换为中断模式。

2.3.4 中断服务程序

串口中断处理是当有数据发送接收时硬件产生中断，然后执行相应的中断服务程序进行处理。这样CPU不必像轮询方式那样花费大量的时间查询外部设备的工作状态，有效提高CPU的使用效率并使系统具有较高的实时性能。

2.4 网络驱动

由于OMAP5910处理器内部不带网络控制器，因此需要外接带有MAC/PHY层的网卡接口芯片。VxWorks整个网络接口呈层次结构，如图4所示。用户网络应用程序通过socket接口调用TCP/IP协议层系列软件，网卡驱动程序则为协议软件提供对网卡的访问。VxWorks也为网卡驱动程序进行了分层，其中老式的BSD4.3驱动程序功能，现在可以由协议层驱动、MUX层和END驱动实现。

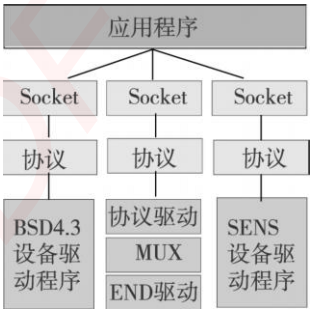

图4 网络接口

由于使用的网卡芯片兼容NE2000，因此采用NE2000END驱动。这里需要修改寄存器地址并初始化参数等。首先修改ne2000End.c，修改头文件包含目录。sysIntConnect修改为intConnect，sysLanIntEnable改为intEnable，并修改返回类型void为STATUS；其次修改config.h文件，在其中添加网络驱动宏定义：#define INCLUDE_NE2000_END；再次修改configNet.h文件。向endDevTbl[]表中添加新驱动条目，填写初始化串。注意，初始化串中不能使用宏定义，数值都采用16进制表示。如下：

```
END_TBL_ENTRY endDevTbl[] =
{#ifdef INCLUDE_NE2000_END
   { 0, ne2000EndLoad, " 0x04000000:0x0e:0x0e:1:0:2", 1, NULL, FALSE},
#endif}
```

最后修改sysLib.c文件。在sysLib.c中完成CPU端口初始化，添加ne2000EnetAddr定义，并实现sysInByte、sysOutByte、sysInWordString和sysOutWordString函数。注意，sysInWordString和sysOutWordString的端口地址不用累加，并需按8位操作。端口初始化代码如下：

```
static void portInit (void)
{  GPIO_REG_BIT_SET (GPIO_DIRECTION_CONTROL, 1<<9);
   GPIO_REG_BIT_SET (GPIO_INTERRUPT_CONTROL, 1<<9);
   GPIO_REG_BIT_CLR (GPIO_INTERRUPT_MASK, 1<<9);  }
```

这段代码设置GPIO9端口为input端口；当电平由低变为高时产生中断信号；使能GPIO9中断。

2.5 下载调试

完成了上述工作后，修改BSP的工作已基本完成。但是在下载调试之前还要做最后一步修改。修改的主要文件是config.h和Makefile。

2.5.1 修改config.h

该文件使得用户可以根据需要配置bootrom程序中一些与开发板硬件资源联系非常密切的参数，从而使bootrom正常地在开发板上运行。config.h文件中的配置参数是在configAll.h文件内容的基础上根据开发板的硬件资源设置的，其中包括系统的启动方式、各种类型存储空间的范围[8]、硬件接口的使用情况等。主要修改内容如下：

（1）定义引导参数，采用网络下载VxWorks映像

```
#define DEFAULT_BOOT_LINE
" ene (0,0) wloveg:D:Tornado2.2\target\proj\Projec \t0\default\vxWorks" \
" h=192.168.0.43 e=192.168.0.31:ffffff00 u=wag pw= wag tn=UP-OMAP5910"
```

（2）根据OMAP5910的地址映射配置目标板的Flash和SDRAM：

```
#define LOCAL_MEM_LOCAL_ADRS 0x10000000  /*板载SDRAM的起始地址*/
```

```
#define LOCAL_MEM_BUS_ADRS  LOCAL_MEM_LOCAL_ADRS  /*总线地址*/
#define LOCAL_MEM_SIZE  0x02000000  /*32MB*/
#define USER_RESERVED_MEM  0x0  /*see sysMemTop()*/
#define ROM_BASE_ADRS  0x00000000  /*ROM的基址*/
#define ROM_TEXT_ADRS  0x00000000  /*ROM中代码的起始地址*/
#define ROM_SIZE  0x01000000
#define ROM_COPY_SIZE  ROM_SIZE
#define ROM_SIZE_TOTAL  0x01000000
#define RAM_LOW_ADRS  0x10001000  /*VxWorks映像入口点*/
#define RAM_HIGH_ADRS  0x10100000  /*bootrom入口点*/
```

(3) 定义VxWorks映像加载方式，选择通过网络加载主机VxWorks内核：

```
#define INCLUDE_END
#define INCLUDE_NE2000_END  /*使用兼容NE2000驱动*/
#undef WDB_COMM_TYPE
#define WDB_COMM_TYPE WDB_COMM_END  /*定义VxWorks加载方式为网络加载，END驱动*/
```

MMU与Cache相关参数配置：

```
#define USER_I_CACHE_MODE  CACHE_WRITETHROUGH
#define USER_D_CACHE_MODE  CACHE_COPYBACK
#define INCLUDE_MMU_BASIC  /*不带VxVMI选项的基本MMU支持*/
#define INCLUDE_CACHE_SUPPORT
```

2.5.2 修改Makefile

在Makefile中主要修改的是存储器部分，修改后的存储器参数必须与config.h中的一致。具体修改如下：

```
ROM_TEXT_ADRS = 00000000 #ROM中代码的起始地址
ROM_WARM_ADRS = 00000004 #ROM热启动的入口地址
ROM_SIZE = 01000000 #ROM地址空间大小
RAM_LOW_ADRS = 10001000 #VxWorks映像入口点
RAM_HIGH_ADRS = 10100000 #bootrom入口点
MACH_EXTRA = ne2000End.o 28f128.o #链接外部驱动
```

2.5.3 修改sysLib.c

由于在config.h文件中支持基本MMU，为了完善BSP的虚拟内存配置，还要对sysLib.c中的sysPhysMemDesc进行配置。sysPhysMemDesc用于初始化MMU的TLB表，它是以PHYS_MEM_DESC为元素的常量数组。PHYS_MEM_DESC在vmLib.h中定义，用于部分内存的虚拟地址到物理地址的映射：

```
typedef struct phys_mem_desc
{   void *virtualAddr;  /*虚拟地址*/
   void *physicalAddr;  /*物理地址*/
   UINT len;  /*这段地址空间的大小*/
   UINT initialStateMask;  /*可以初始化的地址状态*/
   UINT initialState;  /*实际设置这部分内存的初始状态*/
}   PHYS_MEM_DESC;
```

sysPhysMemDesc中的值需要根据系统的实际配置进行修改。在本BSP中，采用平坦式地址映射的方法将OMAP5910的部分物理地址映射到MMU的虚拟内存地址。例如Flash地址的映射：

```
PHYS_MEM_DESC sysPhysMemDesc[] =
{   (void *) (ROM_BASE_ADRS),  /*Flash在系统中的虚拟地址*/
   (void *) (ROM_BASE_ADRS),  /*Flash在系统中的物理地址*/
```

```
ROUND_UP (ROM_SIZE_TOTAL, PAGE_SIZE), /*Flash的大小，必须是页面大
小的整数倍*/
VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE |
VM_STATE_MASK_CACHEABLE,
VM_STATE_VALID | VM_STATE_WRITABLE |
VM_STATE_CACHEABLE_NOT/*Flash状态为：有效、可写、禁止 Cache*/}
```

2.5.4 点灯调试

在串口没有启动之前，只能通过点灯的方式来了解代码的运行情况。控制 LED1和 LED2亮暗的是 GPIO3/LED1和 MPUIO4/LED2引脚，因此在设计点灯调试程序时只需要配置相关寄存器，让 GPIO3/MPUIO4输出高电平，LED1/LED2就会被点亮。具体配置如下：

```
ldr r1, =COMP_MODE_CTRL_0 /*使用引脚复用寄存器之前必须先向
COMP_MODE_CTRL_0写入 0x0000eaef*/
ldr r2, =0x0000eaef
str r2, [r1]
ldr r1, =FUNC_MUX_CTRL_7 /*系统默认使用 GPIO3/MPUIO4引脚*/
ldr r2, =0x00000000
str r2, [r1]
点亮 LED1:
ldr r1, =GPIO_DIRECTION_CONTROL
ldr r2, =0x0000 /*设置 GPIO3为 output*/
str r2, [r1]
ldr r1, =GPIO_DATA_OUTPUT
ldr r2, =0x0008 /*向 GPIO3端口输出高电平*/
str r2, [r1]
点亮 LED2:
ldr r1, =MPUIO_IO_CNTL
ldr r2, =0x0000 /*设置 MPUIO4为 output*/
str r2, [r1]
ldr r1, =MPUIO_OUTPUT
ldr r2, =0x0010 /*向 MPUIO4端口输出高电平*/
str r2, [r1]
```

2.5.5 编译烧写 bootrom

完成上述所有工作后，就可以进行 BSP调试了。利用 CCS和仿真器先将 u-boot.out加载到目标板内存，在主机上建立 tftp服务器并将 bootrom.bin文件放到服务器的根目录下，然后利用 u-boot的 tftpboot命令将 bootrom烧写到目标板的 Flash中；配置 Tornado的 FTP服务器，创建新用户和密码，该用户名和密码必须与 config.h中所定义的一致；重启目标板，然后测试。

测试的几个关键点包括：系统上电后能否执行 romInit.s中的第一条指令；能否跳到第一个 C函数 romStart；能否完成将引导代码从 Flash拷贝到内存中执行；能否完成系统第一次硬件初始化；能否进入第一个内核根任务 usrRoot；能否完成系统第二次硬件初始化；能否在超级终端输出系统启动信息；能否正常加载并运行 VxWorks内核。本课题通过验证，说明 VxWorks成功移植到 OMAP5910平台上。

3 结束语

本课题完成了基于 OMAP5910平台的 VxWorks的移植。包括 VxWorks BSP的修改工作，编译生成了内核引导映像 bootrom，并将其烧写 UP-OMAP5910开发板的 Flash中，最后设计点灯调试程序验证了硬

件初始化工作的正确性。

参 考 文 献

[1] Wind River Systems Corporation. Vxworks Programmer’s Guide [OL]. 2002. http://www.bluetop.com.cn/bluetv/aspnet_client/1/guide/

[2] Wind River Systems Corporation. VxWorks BSP开发人员指南 [M]. 王金刚等译. 北京：清华大学出版社，2003.

[3] 蒋巧，潘孟春. 基于 ARM体系的嵌入式系统 BSP的程序设计 [J]. 电子技术应用，2004 (9)：4-6.

[4] 康静，郑建勇，袁涛，曾伟. VxWorks在 AT91RM9200上的 BSP设计 [J]. 单片机与嵌入式系统应用，2006 (10)：78-81

[5] TEXAS INSTRUMENTS. OMAP5910 Dual-Core Processor Data Manual [K]. TEXAS INSTRUMENTS TECHNOLOGY. 2002.

[6] Justin Helmig. Developing Core Software Technologies for TI’s OMAP? Platform白皮书 [K]. TEXAS INSTRUMENTS TECHNOLOGY. 2002.

[7] 杜春雷. ARM体系结构与编程 [M]. 北京：清华大学出版社，2003.

[8] 马昕，张士洋. 基于 Vxworks的大容量信息存储实现方法研究 [J]. 微计算机应用，2009，28 (10)：94-99

作者简介

李永，男，(1981-)，中国石油大学 (华东) 计算机与通信工程学院实验师，主要研究方向为嵌入式系统和操作系统。

孙士明，男，(1970-)，中国石油大学 (华东) 计算机与通信工程学院讲师，主要研究方向为嵌入式系统和人脸识别。

王爱国，男，(1986-)，中国石油大学 (华东) 计算机与通信工程学院硕士研究生，主要研究方向为嵌入式系统和人脸识别。