基于 VxBus 的设备驱动开发

赵永钢，韩国义

（哈尔滨威克科技股份有限公司，黑龙江 哈尔滨 150090）

摘　要：介绍了在 VxWorks 下，基于 VxBus 的设备驱动程序的开发。结合 PCI2040，讲述了 VxBus 原理、设备驱动开发步骤及具体实现过程。

关键词：VxWorks；VxBus；设备驱动；BSP；PCI2040

中图分类号：TP31　　**文献标识码**：A　　**文章编号**：1674-7720(2010)18-0005-03

Device driver development on VxBus

ZHAO Yong Gang，HAN Guo Yi

(Harbin Veic Technology Co.，Ltd，Harbin 150090，China)

Abstrac：This article describes the VxBus-based device driver development in VxWorks，discusses VxBus bus principle，device driver development process，specifically the implementation process with PCI2040.

Key words：VxWorks；VxBus；device driver；BSP；PCI2040

VxBus 是风河公司新的设备驱动程序架构，是 VxWorks 新增的特性，它是在 VxWorks6.2 及以后版本被增加到 VxWorks 中的。在以前的版本中，驱动程序并没有和工程配置集成到一起，如果要配置设备驱动，就要通过修改 BSP 目录下的 config.h 和 syslib.c 文件来完成。而基于 VxBus 架构模型的好处是允许驱动的集成和配置在 Workbench 工程中完成。这就意味着在 Workbench 环境下，每个驱动程序都能通过可视化环境进行配置，都能够按要求添加或删除设备。本文结合基于 PCI2040 数据采集卡驱动的开发过程[1]，分析了 VxBus 架构下驱动的设计实现。

1 VxBus 简介

VxBus 是指在 VxWorks 中用于支持设备驱动的特有的架构，这种架构包含对 minimal BSP 的支持。它包括以下功能：①允许设备驱动匹配对应设备；②提供驱动程序访问硬件的机制；③软件其他部分访问设备功能；④在 VxWorks 系统中，实现设备驱动的模块化。VxBus 在总线控制器驱动程序服务的支持下，能在总线上发现设备，并执行一些初始化工作，使驱动与硬件设备之间正常的通讯。

图 1 是 VxBus 在整个系统中的位置示意图。从图 1 中可以看到，VxBus 起到了辅助总线的作用，提供了对总线控制驱动的支持。

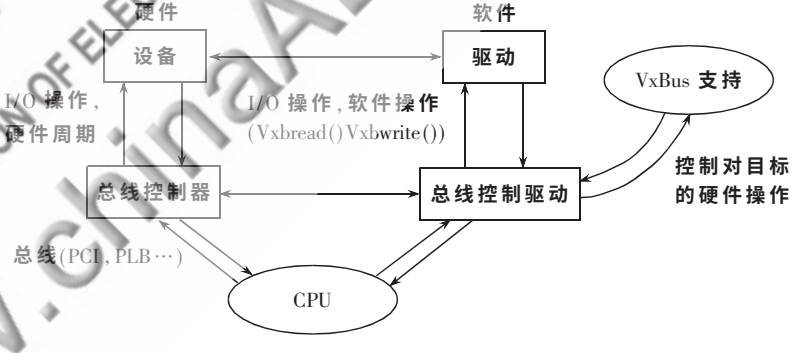

图 1　VxBus 在整个系统中的位置示意图

在 VxWorks6.2 版本发布前，设备驱动并不能被集成到 VxWorks 工程配置当中，为了添加或移出设备驱动，需要有丰富的 BSP 和驱动开发相关的知识[2]。并且在驱动被添加或移出时要去做一些管理 VxWorks 工程的额外的工作。作为 VxWorks 系统组件的一部分，VxBus 消除了上面遇到的一些难题，各种驱动和支持组件的添加与删除完全可以在 Workbench 工程中进行，而不需要 BSP 和驱动相关的知识，也不会在添加、删除驱动时增加管理 VxWorks 工程的额外工作。因此大大方便了 BSP 的开发。

2 硬件介绍

TI 公司推出的 PCI2040 是一款用于实现 PCI 局部总

线与 DSP 之间无缝链接的专用芯片。在 VxWorks 实时操作系统环境下实现主机与 DSP 的通讯，系统利用 PCI2040 实现 TMS320VC5410 与主机的通讯。由于 PCI2040 是 TI 的配套专用芯片，硬件级的连接比较简单，将对应的引脚连接即可。需要注意的是，未用的输入信号线需要通过上拉电阻上拉至有效逻辑电平。TMS320VC5410 的 MCBSP0 与 TLC2548 连接，实现 8 路 12 位 A/D 数据的采集。TMS320VC5410 将采集到的数据通过 PCI2040 传输到主机上，数据在主机上得到进一步的处理。硬件连接框图如图 2 所示。

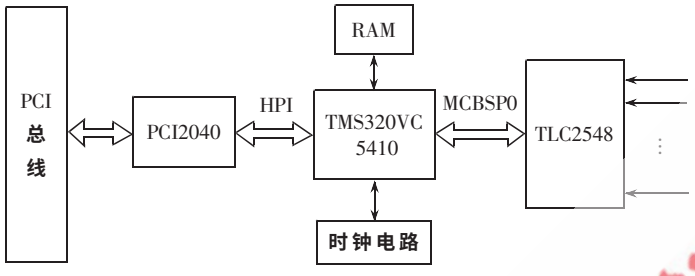

图 2 硬件连接框图

3 驱动开发

基于 VxBus 架构下 PCI2040 设备驱动的开发主要包括设备的初始化、设备控制以及设备驱动如何以组件形式添加到 Workbench 配置界面中。下面分步介绍它的实现。

3.1 设备驱动初始化

设备的初始化，包含在 BSP 的初始化过程中[3]，主要分三个阶段，如图 3 所示。

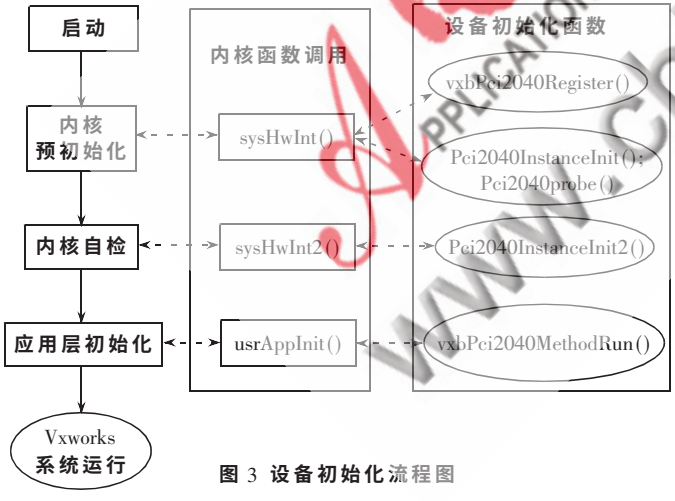

图 3 设备初始化流程图

3.1.1 内核预初始化阶段

系统上电启动，CPU 在上电时跳转到一个指定的地址，开始执行指令，初始化内存和 CPU，然后是 VxWorks 的初始化处理。

在 VxWorks 内核预初始化早期，BSP 的 sysHwInit ( ) 函数被执行[4]，在这个函数中，设备驱动初始化工作第一步被执行。sysHwInit( )函数执行一些早期的初始化，调用 hardWareInterFaceInit( )函数，执行初始化硬件内存分配机制，这步允许在系统内存池初始化之前，限制为设备驱动分配内存，这个函数接着调用 hardWareInterFaceBusInit( )，在 hardWareInterFaceBusInit( )函数中完成所有设备驱动和模块的注册工作。PCI2040 的注册函数是 vxbPci2040Register()。vxbPci2040Register()通过数据结构，向系统注册一些设备初始化函数。其中涉及到三个数据结构：

```
LOCAL struct drvBusFuncs PciFuncs =
    {
    Pci2040InstInit,      /* devInstanceInit */
    Pci2040InstInit2,     /* devInstanceInit2 */
            Pci204InstConnect   /* devConnect */
    }
```

在这个结构中，包含了初始化阶段要调用的函数。下面的初始化过程会用到这些函数。

```
LOCAL struct vxbDeviceMethod Pci2040Methods[] =
{
DEVMETHOD(ReadHPID,     PCI2040ReadHPID),
DEVMETHOD(WriteHPID,    PCI2040WriteHPID),
DEVMETHOD(ReadHPIA,     PCI2040ReadHPIA),
DEVMETHOD(WriteHPIA,    PCI2040WriteHPIA),
DEVMETHOD(ReadHPIC,     PCI2040ReadHPIC),
DEVMETHOD(WriteHPIC,    PCI2040WriteHPIC),
DEVMETHOD(ReadCSR,PCI2040ReadCSR),
DEVMETHOD(WriteCSR,     PCI2040WriteCSR),
{ 0, 0 }
}
```

这个结构提供了应用软件操作硬件的一些函数及方法。

```
LOCAL struct vxbPciRegister Pci2040DevPciRegistration =
{
    {
    NULL,                   /* pNext */
    VXB_DEVID_DEVICE,       /* devID */
    VXB_BUSID_PCI,          /* busID = PCI */
    VXB_VER_4_0_0,          /* vxbVersion */
    LNPCI_NAME,             /* drvName */
    &Pci2040Funcs,          /* 总线驱动函数 */
    Pci2040Methods,         /* 设备方法结构 */
    Pci2040Probe,           /* 设备探测函数 */
    Pci2040ParamDefaults    /* 参数 */
    },
NELEMENTS(PciPci204DevIDList),
PciPci204DevIDList          /* 设备资源列表 */
};
```

最后这个结构在 vxbPci2040Registe()中被使用。这个结构包括几个驱动的初始化入口，其中 Pci2040Probe()是 PCI2040 采集卡的硬件探测函数，该函数在 VxBus 初始

化过程中检测采集卡的数量，当检测到采集卡时，将采集卡与驱动结合，形成设备的一个实例，以便应用程序使用。Pci204InstanceInit( )函数在 VxBus 初始化的第一阶段被调用，Pci204InstanceInit( )函数只是简单地确保设备的中断被禁止。

当所有驱动在 VxWorks 注册之后，hardWareInterFaceBusInit( )和 hardWareInterFaceInit( ) 函数返回，sysHwInit( ) 完成非 VxBus 驱动的初始化并返回。sysHwInit( ) 函数返回后，VxWorks 内核被初始化。

3.1.2 内核自检

在这个阶段，内核在 sysHwInit2( )中执行，BSP 调用 Pci2040InstanceInit2( )函数[5]。在这个函数中，建立系统内存到设备空间的映射。关键部分代码如下：

```
LOCAL void Pci204InstInit2(VXB_DEVICE_ID pDev)
    {……
        for (i = 0; i < VXB_MAXBARS; i++)
            {
            if (pDev->regBaseFlags[i] == VXB_REG_IO)
                break;
            }
           if (i == VXB_MAXBARS)
           return;
    pDrvCtrl->Pci2040Bar = pDev->pRegBase[i];
    vxbRegMap (pDev, i, &pDrvCtrl->Pci2040Handle);
                                    //设备 I/O 映射到系统内存
    ……
    }
```

此时，完成内核服务初始化，并可以被驱动访问。但是，中间层的服务仍然无效。

3.1.3 应用程序初始化驱动部分

在 devInstanceInit2( )函数最后，创建用户的运行任务，并完成设备驱动的初始化。在这个阶段，Pci2040InstanceConnect( )函数被调用，完成最后的初始化工作，在这个函数中，主要是建立中断与中断服务程序的连接。

至此，设备驱动的初始化完成。

3.2 驱动程序的配置

采用 VxBus 驱动的一个主要优点是：设备的驱动程序可以被看成 VxWorks 系统的一个组件，通过集成的 Workbench 开发环境来配置设备驱动。为了实现这一功能，开发的驱动需要增加一些额外的扩展文件。标准 VxWorks 设备驱动有一个最小的文件集，对于大多数 VxWorks 设备驱动，最小的设备驱动集要求有 6 个单独的文件[6]。PCI2040 数据采集卡需要有以下文件：

· 一个驱动源文件 PCI2040.c，执行驱动运行逻辑，包括 PCI2040 驱动的实现代码。

· 一个组件描述文件 PCI2040.cdf，允许集成驱动到 VxWorks 开发工具 Workbench 当中。

· 一个 PCI2040.dc 文件，提供驱动注册函数原型。

· 一个 PCI2040.dr 文件，提供一个调用注册函数的 C 语言代码段。

· 一个 readme 文件，提供版本信息。

· 一个 makefile 文件，提供建立驱动的编译规则。

当上述文件在 workbench 环境下进行相应的配置后，PCI2040 的设备驱动就会以组件的形式出现在开发工程的 Kernel Configuration 选项中，可以方便地进行 PCI2040 驱动配置。

4 应用程序与驱动的通信

为了使设备和驱动能够在 VxWorks 系统中使用，让应用程序、中间件、VxWorks 内核模块访问设备，执行一些操作，最基本的方法是在 VxWorks 中采用 VxBus 方法来实现硬件设备的访问。VxBus 方法是在驱动中公开一个入口，使 VxBus 中 API 函数可以调用这些入口函数。在 PCI2040 初始化阶段，Pci2040Methods 结构中注册的函数就是在驱动中公开的函数，用于对 PCI2040 的操作。

例如，通过 PCI2040 完成对 DSP 数据寄存器的访问

```
struct vxbDriverControl ctrl;
vxbDevMethodRun(DEVMETHOD_CALL(ReadHPID),&ctrl);
```

vxbDevMethodRun( )函数够被用于调用一个指定的驱动方法，这个函数反复查找所有的实例，并检查每一个，看是否有指定公开申明的方法，如果实例有指定的方法，vxbDevMethodRun( )调用方法函数。

为了避免重复遍历在系统上的所有实例，可以用 vxbDevMethodGet ( ) 函数找出驱动函数相对应的驱动方法，然后通过下面代码完成函数调用。

```
STATUS  (*methodFunc)(VXB_DEVICE_ID devID, void *pArg);
methodFunc = bDevMethodGet (devID,DEVMETHOD_CALL(ReadHPID));
if(methodFunc ! = NULL )
(*methodFunc)(devID, pArg);
```

在 PCI2040 的数据采集卡中，通常是 DSP 在采集完数据后，通过中断通知主机，去读取数据。下面是中断服务相关代码。

```
void PCI2040Isr()
{ ……
    temp=*(PCI2040. instID.pRegister+0x4); //读中断寄存器
    if((tempr&0x1)! =0)              //检查是否是该实例中断
    { *(PCI2040. instID.pRegister +0x4)=0x1;
    temp= *(PCI2040. instIDpDspHpicRegister);
    *(PCI2040. instIDpDspHpicRegister)=temp|0x0808;
                                    //通知 DSP,清除 HINT 中断
    semGive(semForPci2040Int);
  }
}
```

采用基于 VxBus 架构来开发 PCI2040 数据采集卡的驱动，通过扩展文件实现驱动的配置。与简单的非VxBus

驱动相比，显然增加了工作量，然而对于基于多个 BSP 设备的复杂的驱动，VxBus 驱动是优于非 VxBus 驱动的。通过实际运用证明，所开发采集卡的驱动能够稳定运行，并且能很方便地将该驱动移植到其他的系统。

参考文献

[1] 高超，郝燕玲，吴润. VxWorks 下网卡驱动程序的开发[J]. 微计算机信息. 2004(9):18-20.

[2] 周启平，张杨. VxWorks 下设备驱动程序及 BSP 开发指南[M]. 北京.中国电力出版社，2004.

[3] VxWorks Device Driver Developer's Guide Wind River Systems, Inc.2007.1,6.6

[4] VxWorks Device Driver Developer's Guide Volume 2, 6.6. Wind River Systems, Inc.2007.2,6.6

[5] VxWorks Device Driver Developer's Guide Volume 3,6.6. Wind River Systems, Inc.2007.3,6.6

[6] BSP Developer's Guide，6.6 Wind River Systems,Inc.2007.

（收稿日期：2010-04-23）

作者简介：

赵永钢，男，1975 年生，学士学位，工程师，主要研究方向：嵌入式设备的开发应用、数据处理与模式识别等。