基于 VxBus 的驱动程序架构分析

付月生，王　丽

（中船重工第 722 研究所 基础平台专业部，湖北 武汉　430079）

摘　要：VxBus 设备驱动架构是 VxWorks 操作系统中引入的最新概念，风河公司在 VxWorks 6.2 中加入这个架构以来，不断完善，到目前的 VxWorks 6.9，设备驱动程序的开发基本上都采用 VxBus 架构。VxBus 驱动程序架构是 VxWorks 操作系统未来驱动程序发展的方向，也是设计 VxWorks 操作系统中设备驱动的必经之路。本文分析 VxBus 设备驱动架构的优点，VxBus 组成结构以及基于 VxBus 架构的设备驱动开发流程。

关键词：VxBus；设备驱动程序；VxWorks

中图分类号：TP316.2　　**文献标识码**：A

The Driver Structure Analysis Based on VxBus

FU Yue-sheng, WANG li

(The 722th Institute of China Shipbuilding Industry Corporation, Wuhan　430079, China)

Abstract: The VxBus device driver structure is fairly new to VxWorks, having been added in VxWorks release 6.2 by WindRiver. Now in the VxWorks release 6.9, Basically all the development of device driver use the VxBus structure. VxBus structure is the direction of the device driver development in VxWorks, also, it's the only way to design driver in VxWorks. This paper analyze the advantage of VxBus structure, the component of VxBus and the device driver development process based on VxBus structure.

Key words: VxBus; device driver; VxWorks

1　引　言

VxBus 是风河公司 VxWorks 操作系统的最新设备驱动程序架构，在 VxWorks6.2 系统中首次引入。在以前的版本中，设备驱动程序作为一个单独的部分，没有被整合到 VxWorks 的工程配置中，因此设备驱动程序的开发配置依赖于修改操作系统的 BSP。随着 VxBus 驱动架构的不断成熟，大部分驱动开始采用 VxBus 架构，VxBus 架构集成了多种通用驱动开发程序，并将驱动的安装配置集成到了开发环境 WorkBench 中，每个设备驱动程序的安装、配置、删除都能通过可视化的界面操作。

2　VxBus 架构分析

VxBus 驱动架构的最主要改变和优点就是配置。多种类别的设备驱动被集成到 VxBus 中，通过 WorkBench 可以方便的对其添加、配置、删除。

2.1　VxBus 架构支持的驱动类别

对于一个设备，最基本的属性就是它实现什么功能，驱动则是对这些功能进行一系列的管理。不同的设备完成不同的任务，有一些设备是存储数据用的硬盘或磁盘，有一些设备具有打印功能或者图形显示功能，还有一些设备能够进行控制操作，等

等。不同的设备可能具备相同的功能，就像在显示屏上显示图形的图形显示控制设备，它可以是VGA显卡，也可以是目前流行的PCI－E显卡，不同的设备实现相同的功能，正是因为这些相同的功能，设备驱动根据实现任务的不同，可以分为不同的种类。

风河公司官方提供的驱动类别包括[1]：串行设备驱动；存储设备驱动；网络接口驱动；NVRAM驱动；定时器驱动；DMA控制器驱动；总线控制器驱动；USB驱动；中断控制器驱动；多功能设备驱动；控制台设备驱动；远程处理器单元驱动；以及其它类型的驱动。

风河同时提供方法，可以将用户自己开发的驱动，移植为符合VxBus架构下的驱动。

这些类别的驱动文件组织结构与VxWorks之前版本略有不同，理解VxWorks驱动文件组织结构对VxBus开发理解有重要的意义，下面是VxWorks6.8中三个设备驱动文件相对路径[1]：

installDir/vxworks－6.8/target/3rdparty，这个路径中存放的主要是第三方提供的基于VxBus架构的设备驱动程序；

installDir/vxworks－6.8/target/src/hwif，这个路径中存放的是风河官方提供的基于VxBus的设备驱动程序，上面介绍的各种类别的驱动源程序都在这个路径下面；

installDir/vxworks－6.8/target/src/drv，这个路径中存放的是风河老版本中使用的设备驱动架构，不是VxBus兼容的版本。

2.2 VxBus驱动架构的关键概念

理解VxBus设备驱动架构，首先需要理解三个概念：设备、驱动、实例。设备是指一些硬件信息；驱动是指为了让硬件设备访问操作系统所需要的可执行代码以及配置信息；每一个驱动可以关联0个或多个设备，这一对关联关系就是一个实例。这三者之间的关系示意图如图1所示。

将驱动程序和硬件设备关联起来的是一个称作"Method"的概念，每一个VxBus的驱动都有驱动方法，即driver method。当使用method的时候，会查询一个单独的实例或者所有实例，这种查询可以查询哪个实例支持请求的功能或者执行请求操作。如图2所示，是一个简单的例子，描述了这种查询操作的情况。图中网络协议栈使用vxbDevMethodGet()操作查询哪个实例支持muxDevConnect操作，遍历之后查找到了Yukon II Network支持这种操作，这个时候会进行匹配，返回Yukon II Network驱动提供的相应函数的指针。图中Auxiliary Clock也进行了同样的操作，如果没有查询到符合条件的method，这种匹配就会返回NULL指针。

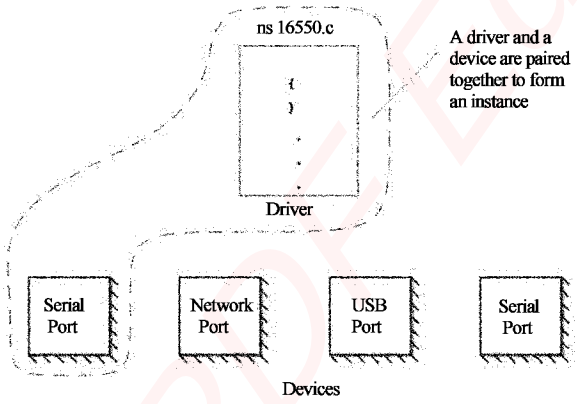

图1 VxBus驱动结构

VxBus架构将各种不同功能的设备集成到一个统一的架构中，使驱动的配置更加简单，这些变化与一个文件紧密相连，这就是"hwconf.c"。VxBus驱动很大一部分的配置都是在这个文件中完成的，这个文件列出了PLB上的所有设备，每个设备的资源信息以及参数信息。"hwconf.c"文件中有两个关键的数据结构：hcfResource和hcfDevice。这两个数据结构的原型在installDir/vxworks－6.x/target/h/hwif/vxbus/hwConf.h中定义。每一个设备都需要定义一个hcfResource类型的变量来描述它的基本信息，这些信息包括regBase，irq，intrN等等。hcfDevice是PLB上的设备列表，包含了设备名，设备号，总线类型，hcfResouce等信息。

下面是一个1655x串口的例子[2]，这些变量都在hwconf.c中定义，目的就是将一个符合1655x标准的串口驱动包含到BSP中，并可以通过PLB进行访问配置。

```
const struct hcfResource ns1655x0Resources[] = {
        { "regBase",        HCF_RES_INT,
    { (void *)UART0_BASE } },
        { "clkFreq",        HCF_RES_ADDR,
    { (void *)sysClkFreqGet } },
        { "regInterval",    HCF_RES_INT,
  { (void *)DUART_REG_ADDR_INTERVAL } }
    };
```

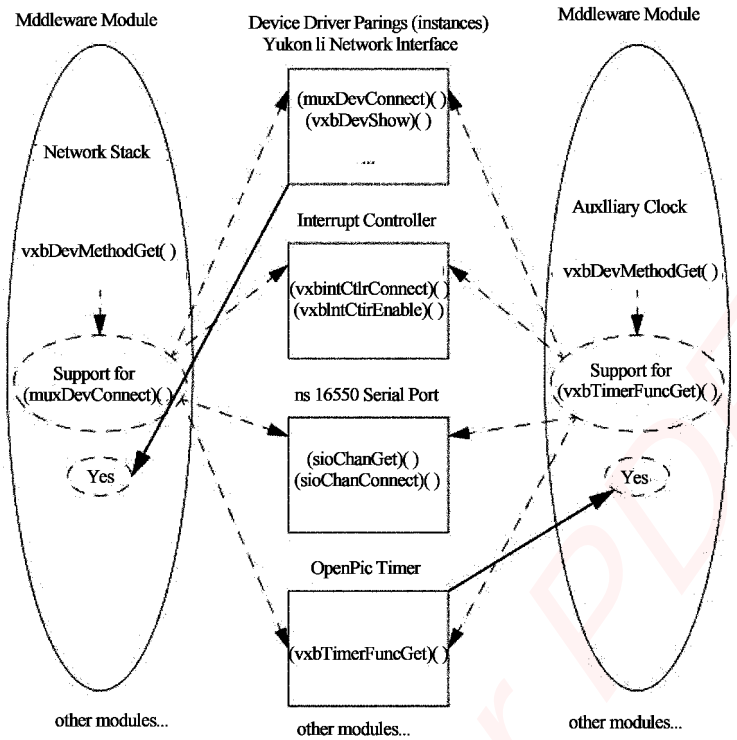

图 2　Method 简介

```
# define ns1655x0Num NELEMENTS(ns1655x0Resources)
const struct hcfDevice hcfDeviceList[] =
{
    { "ns16550","0", VXB_BUSID_PLB, 0, ns1655x0Num, ns1655x0Resources },
};
const int hcfDeviceNum = NELEMENTS(hcfDeviceList);
```

上文中提到的 PLB，是 VxBus 架构中引入的一个新的概念，PLB－processor local bus，即处理器局部总线。PLB 是 VxBus 架构中最重要的一个概念，以前版本的设备驱动都是独立存在的，VxBus 架构引入 PLB 之后，各个独立的设备驱动形成了一个链表，我们使用的大部分设备驱动都是直接或者间接的挂接在 PLB 上的。PLB 实际上是一个虚拟的 CPU 总线，下一节将介绍 VxBus 的初始化流程，同时也会对 PLB 有进一步的介绍。

2.3　VxBus 驱动架构的初始化流程

VxBus 驱动的初始化在 sysLib.c 中完成，整个初始化流程如图 3 所示。

每一个符合 VxBus 结构的驱动都包含多个初始化函数，这些函数有着统一的形式[3]，如：driverNameInstInit，driverNameInstInit2，driverNameInstConnect。在驱动初始化过程中，这几个函数起到关键性作用。VxBus 驱动初始化过程第一阶段从 syslib.c 的 sysHwInit 函数开始，sys

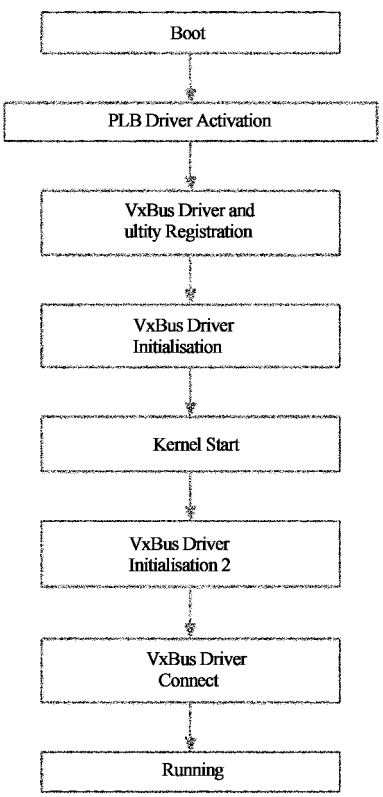

图 3　VxBus 初始化流程

HwInit 函数会调用 hardWareInterFaceInit()，这个函数是 VxBus 驱动初始化的入口函数，这个函数的第一个任务是初始化硬件内存分配机制，随后这个函数会调用 hardWareInterFaceBusInit()，这个函数将会完成 VxBus 驱动初始化的大部分工作。hardWareInterFaceBusInit 函数首先会根据 config.h 中包含的使用 VxBus 架构驱动的设备类型进行 VxBus 设备驱动的注册，完成注册后程序会调用 vxbInit()进行设备扫描并建立实例。hardWareInterFaceBusInit 函数中第一个进行的就是 plbRegister()，plbRegister 函数所进行的就是注册我们上一节所提到到的 PLB，VxBus 架构下设备驱动都是挂在 PLB 下面的，所以首先进行的就是 PLB 的注册，PLB 是一个特殊的设备，是 vxbus 子系统中最重要的设备，plbRegister 函数中实际工作是创建了几条 vxbus 子系统中重要的链表，所有的设备驱动注册都是建立在这个链表的基础上的。vxbInit 函数会读取 hwConf.c 文件中的 hcfDeviceList 这个数组，根据这个数组中的 Resources 与已经注册了的设备驱动进行匹配查找，如果找到匹配的会创建一个设备实例，vxbInit 中会建立实例链表，将所有实例统一管理，并调用对应驱动的 driverNameInstInit 函数。driverNameInstInit 是驱动程序第一次对硬件进行初始化，这个时候因为没有操作系统的支持，很多设备的初始化都比较简单。至此 VxBus 驱动初始化的第一个阶段完成，sysHwInit 会继续完成 vxWorks 内核的初始化。

VxBus 初始化的第二个阶段是在 syslib.c 中 sysHwInit2()这个函数中完成的。这个时候大部分内核服务已经初始化完成，sysHwInit2()中首先会调用 vxbDevInit()，这个函数会调用对应驱动的 driverNameInstInit2 函数。sysHwInit2 函数接着以任务的形式运行 vxbDevConnect()，这个函数会调用对应驱动的 driverNameInstConnect 函数。至此整个 VxBus 设备驱动的初始化完成。

3 使用 VxBus 架构的设备驱动

VxBus 架构使设备驱动的使用更加方便，下面以 freescale powerpc8308 为例，介绍 VxBus 驱动的使用。powerpc8308 是 freescale 推出的一款多功能处理器[5]，具备网口、串口、PCIE、USB 等多种接口，下面介绍网口、串口的 VxBus 驱动使用方法。

powerpc8308 的千兆以太网接口在 vxworks 中以 etsec 命名，串口是兼容 PC16552D 的通用串行设备。这两类设备驱动在 vxWorks 中均提供了相应的 VxBus 驱动[4]，使用起来非常方便，下面进行简要介绍：

1)通过包含相应宏定义的方式，在 config.h 文件中加入对 VXBUS 以及相应驱动程序的支持，例如：INCLUDE_VXBUS、INCLUDE_ETSEC_VXB_END、DRV_SIO_NS16550，对于网口还需要加入 INCLUDE_END、INCLUDE_MII_BUS、INCLUDE_GENERICPHY。

2)根据硬件情况，修改 hwconf.c 文件。powerpc8308 具有两路千兆以太网接口和两路串口，因此，需要在 hwconf.c 中分别定义。如下：

```
const struct hcfResource ns1655x0Resources[] = {
        { "regBase",    HCF_RES_INT,    { (void *)UART0_BASE } },
        { "clkFreq",    HCF_RES_ADDR,   { (void *)sysClkFreqGet } },
        { "regInterval", HCF_RES_INT,   { (void *)DUART_REG_ADDR_INTERVAL } }
    };
    # define ns1655x0Num                NELEMENTS(ns1655x0Resources)
    const struct hcfResource ns1655x1Resources[] = {
        { "regBase",    HCF_RES_INT,    { (void *)UART1_BASE } },
        { "clkFreq",    HCF_RES_ADDR,   { (void *)sysClkFreqGet } },
        { "regInterval", HCF_RES_INT,   { (void *)DUART_REG_ADDR_INTERVAL } }
    };
    # define ns1655x1Num                NELEMENTS(ns1655x1Resources)
    const struct hcfResource motEtsecHEnd0Resources[] =
        {
        { "regBase",    HCF_RES_INT,    { (void *)ETSEC1_BASE } },
        { "phyAddr",    HCF_RES_INT,    { (void *)1 }},
        { "miiIfName",  HCF_RES_STRING, { (void *) "motetsec" } },
```

```
        {  "miiIfUnit ","  HCF_RES_INT,
    { (void *)0 } }
        };
    # define etsecHEnd0Num        NELE-
MENTS(motEtsecHEnd0Resources)
    const struct hcfResource motEtsecH-
End1Resources[] =
        {
        { "regBase ","  HCF_RES_INT,    {
(void *)ETSEC2_BASE } },
        { "phyAddr ","  HCF_RES_INT,
{ (void *)2 }},
        { "miiIfName ","HCF_RES_STRING,
{ (void *) "motetsec  " },
        { "miiIfUnit ","HCF_RES_INT,    {
(void *)0 } }
        };
    # define etsecHEnd1Num    NELEMENTS
(motEtsecHEnd1Resources)
```

同时在 hcfDeviceList 中做出相应的定义：

```
    const struct hcfDevice hcfDeviceList[] =
    {
    ……
    { "motetsec ","0, VXB_BUSID_PLB, 0, et-
secHEnd0Num, motEtsecHEnd0Resources },
    { "motetsec ","1, VXB_BUSID_PLB, 0, et-
secHEnd1Num, motEtsecHEnd1Resources },
    { "ns16550 ","0, VXB_BUSID_PLB, 0,
ns1655x0Num, ns1655x0Resources },
    { "ns16550 ","1, VXB_BUSID_PLB, 0,
ns1655x1Num, ns1655x1Resources },
    ……
    }
```

对于 powerpc8308 的网口和串口来说，完成这两步的配置操作，在系统启动过程中就会根据相应的设置对其进行初始化，启动之后可以使用 VxBusShow 来查看相应的设备实例运行情况，如果需要修改也仅仅需要修改 hwconf.c 中的相应配置即可，这样的架构方式大大提高驱动程序开发效率和可重用性。

4 结 论

本文介绍了 vxWorks 操作系统的最新驱动架构 VxBus 的基本功能，分析了 VxBus 架构的优点以及对驱动开发带来的提高。通过对 VxBus 驱动架构初始化流程的分析，详细介绍了驱动程序在 VxBus 架构下的初始化流程，以及 VxBus 驱动架构下驱动程序开发的关键概念。最后，通过 powerpc8308 下网口、串口如何使用 VxBus 驱动架构的例子，简要介绍了 VxBus 驱动的使用方法。VxBus 驱动架构是 VxWorks 操作系统未来驱动程序的统一模型，提高了 VxWorks 操作系统驱动程序的开发效率、可重用性及易配置型。

参考文献

[1] WindRiver Systems. vxworks device driver developers guide volume 1_6.8[J]. WindRiver Systems, Inc. 2010.

[2] WindRiver Systems. vxworks device driver developers guide volume 2_6.8[J]. WindRiver Systems, Inc. 2010

[3] WindRiver Systems. cvxworks device driver developers guide volume 3_6.8[J]. WindRiver Systems, Inc. 2010

[4] WindRiver Systems. vxworks bsp developers guide 6.8[J]. WindRiver Systems, Inc. 2010

[5] Freescale. MPC8308 PowerQUICC II Pro Processor Reference Manual[J]. Freescale, 2010.4

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案

9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计

6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计