VxBus 的 A429 接口驱动①

刘　宇, 唐法荣

(中电科航空电子有限公司, 成都 611731)

摘　要: VxBus 驱动架构是在 VxWorks6.2 版本开始加入到 VxWorks 操作系统中的全新的驱动架构,VxBus 驱动架构具有模块化并且对 BSP 依赖度小的特点. 分析了 VxBus 驱动架构的工作方式, 并基于 VxBus 设计实现了一种 ARINC429 接口驱动. 在 VxWorks 操作系统中采用 VxBus 驱动架构, 不仅满足了驱动高可靠性的要求, 并且提高了驱动的可扩展性.

关键词: VxBus; VxWorks; ARINC429; 驱动

VxBus A429 Interface Driver

LIU Yu, TANG Fa-Rong

(China Electronic Technology Avionics Company Limited, Chengdu 611731, China)

Abstract: The new driver architecture VxBus was integrated in VxWorks operation system after the release of VxWorks 6.2. VxBus dirver architecture was modularized with minimal BSP support. This paper analyzes the work mode of VxBus driver architecture. We designed and implemented a type of ARINC429 interface driver based on VxBus. The implement of vxWorks driver based on VxBus has a high degree of reliability and expansibility.

Key words: VxBus; VxWorks; ARINC429; driver

VxBus 是一种基于 VxWorks 操作系统全新的驱动架构, WindRiver 公司从 VxWorks 6.2 版本开始加入 VxBus 用以替代 VxWorks 传统的驱动架构[1]. VxBus 驱动采用模块化的设计思想, 降低了驱动对 BSP 的依赖度, 并且使驱动的配置和移植更加灵活和方便.

ARINC429 总线是民用飞机上最常用的一种数据总线. 它采用差分的工作方式, 具有结构简单、可靠性高和抗干扰性强的特点. 主要于飞机航电设备数据传输和控制命令传输. 飞机通信导航系统中常采用 ARINC429 总线来传输 VHF、HF、VOR 和 DME 等通信导航设备的调谐控制命令. 本文结合通信导航系统项目中 ARINC429 接口模块的应用, 提出了一种 VxBus 驱动架构下 ARINC429 接口驱动的设计.

1　VxBus驱动模式

基于 VxBus 驱动架构设计的设备驱动需要极少的 BSP 支持. 由于模块化的设计, VxBus 驱动在 VxWorks 操作系统下作为一个组件, 根据需要可以通过 Workbench 开发环境或者 vxprj 命令行功能进行添加和删除配置.

1.1　VxBus 驱动架构分析

在 VxBus 驱动架构下, 驱动的可执行代码和配置信息与设备硬件结合起来形成一个可以被操作系统使用的实例[2], 如图 1 所示. 并且一个驱动可以和多个设备硬件相关联.

应用程序通过使用 VxBus 的驱动方法可以实现对硬件设备的操作, VxBus 驱动方法实现了应用程序和设备硬件之间的控制命令和数据等信息的传递, 对应用程序屏蔽了底层的硬件操作, 并且应用程序可以实现对一个或者多个实例的操作.

VxBus 驱动架构下实例通过声明驱动方法使其对整个系统有效. 应用程序可以通过调用 vxbDevMethodGet()

函数查找并获取单个实例的驱动方法. 也可以通过调用vxbDevMethodRun()函数直接查找包含某个驱动方法所有实例并运行.

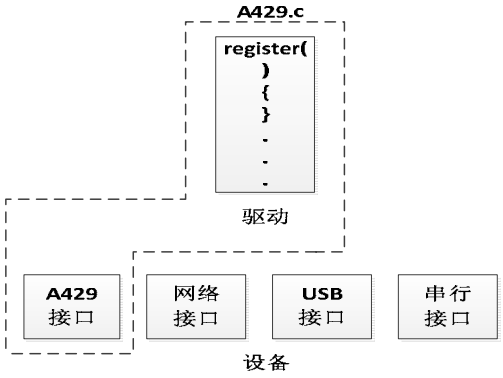

图 1　VxBus 实例示意图

1.2　VxBus 驱动初始化流程分析

VxBus 驱动架构下的驱动初始化流程依据其与VxWorks 内核的初始化流程的关联, 大体上可以分为硬件发现阶段、驱动注册阶段、驱动初始化第一阶段、驱动初始化第二阶段和驱动初始化第三阶段共五个阶段[3], 如图 2 所示.

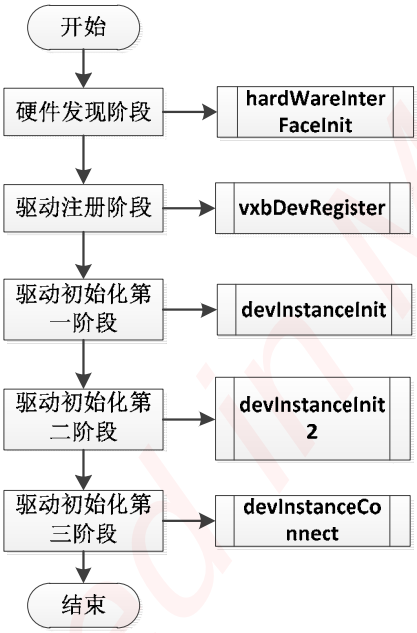

图 2　VxBus 初始化流程图

在硬件发现阶段, VxWorks 系统通过调用sysHwInit 函数来完成对硬件的前期初始化, sysHwInit 函数则通过调用 hardWareInterFaceInit 函数来初始化硬件内存分配机制. 在驱动注册阶段, hardWare-InterFaceInit 函数对 hardWareInterFaceBusInit 函数进行调用, hardWareInterFaceBusInit 函数则调用 vxbDev-Register 函数来完成 VxBus 驱动的注册和激活.

VxBus 驱动完成注册之后, 就进入驱动初始化第一阶段, 完成一些前期初始化操作, 例如驱动方法的连接. 在该阶段, VxWorks 操作系统内核并未初始化完全, 内核服务并未运行, 中断控制器驱动和串口驱动也并未就绪, 所以 devInstanceInit 函数在初始化时必须确保中断时关闭的.

驱动初始化第一阶段完成之后, VxWorks 操作系统内核才完成初始化. 进入驱动初始化第二阶段, 完成驱动资源分配等操作. 在该阶段, 系统调用 sysHwInit2 函数完成内核服务的初始化, 此时内核服务才可以使用, sysHwInit2 则调用 devInstanceInit2 函数完成驱动第二阶段的初始化操作. 驱动初始化第三阶段则最终完成驱动程序与外部设备的连接等操作, 使得应用程序可以正常使用驱动.

2　VxBus的ARINC429驱动设计

ARINC429 接口硬件采用了自研的 ARINC429 模块板卡. 该板卡具有 8 路 ARINC429 数据输入接口, 4 路 ARINC429 数据输出接口. 具体的硬件结构如图 3 所示, 主控芯片采用了可编程逻辑器件[4]XC95144XL, 接口芯片采用了 ARINC429 接口芯片 DEI1016 以及 ARINC429 驱动芯片 BD429[5]. 应用程序对数据的读取采用查询的方式, 数据的发送采用写固定地址寄存器的方式.

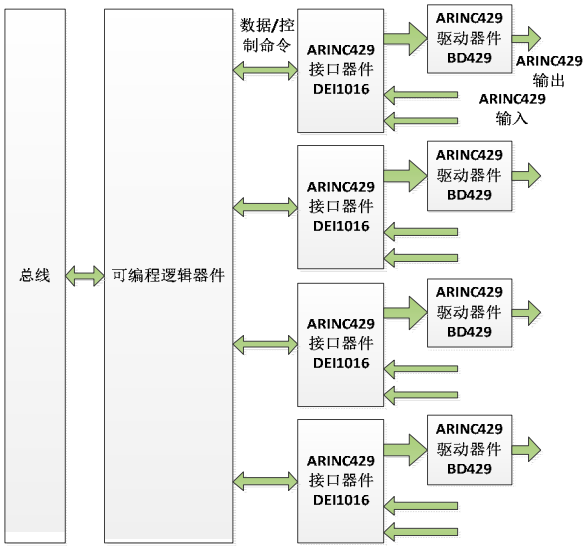

图 3　ARINC429 模块硬件结构

ARINC429 接口驱动采用 driverControl[6]通用驱动方法实现设备控制操作, driverControl 方法通过传递指针参数的方式实现驱动和应用程序之间数据和控制命令的传递和返回. 对应于 ARINC429 驱动的 driverControl 方法的函数命名为 a429DriverCtrl, a429DriverCtrl 实现了驱动的主要操作, 主要包括 ARINC429 接口的读写操作和速率配置操作. 具体工作流程如图 4 所示.

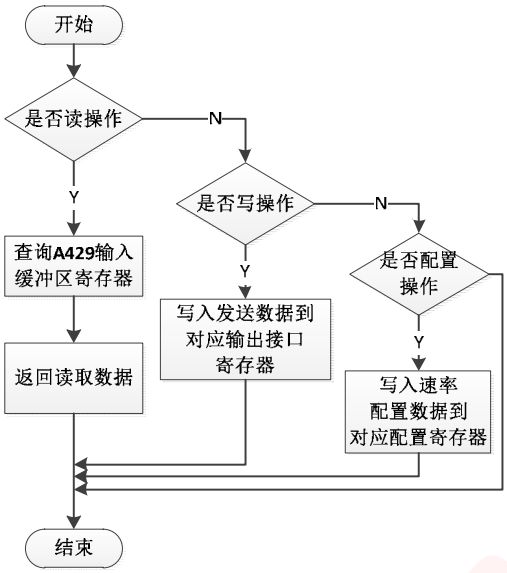

图 4　driverControl 驱动方法流程图

2.1　驱动的初始化

VxBus 驱动架构下 A429 驱动的初始化根据初始化阶段的不同, 由 a429Register、a429InstInit、a429InstInit2 和 a429InstConnect 四个函数来实现.

a429Register 函数通过调用 vxbDevRegister 函数完成 a429 驱动的注册. a429InstInit 函数完成将 a429 驱动方法接口拷贝到具体实例 a429InstInit2 函数实现 a429 驱动资源的分配, 这里定义了一个 A429_DEVICE 结构体类型用于存储 A429 接口配置信息. 具体的函数代码如下:

```
LOCAL void a429InstInit2 (VXB_DEVICE_ID pInst)
    {
        A429_DEVICE * pDrvCtrl;
        pDrvCtrl = (A429_DEVICE *)hwMemAlloc(sizeof(A429_DEVICE));
    if (pDrvCtrl == NULL) return;
    bzero ((char *)pDrvCtrl, sizeof(A429_DEVICE));
    pInst->pDrvCtrl = pDrvCtrl;
    }
```

a429InstConnect 函数主实现对 A429 模块板卡的硬件初始化配置, 包括清除接收端口中断、复位接口寄存器和接口速率配置. 具体的函数代码如下:

```
LOCAL void a429InstConnect(VXB_DEVICE_ID pInst)
    {
    uint32_t index = 0;
    uint32_t speed = 0;
        /* clear a429 recv port interrupt */
    for(index = 0; index < E_RECV_PORT_COUNT ; index++)
        {
            SET_INT_REG(index, CLR_INT_CMD);
        }
        /* reset a429 port */
        SET_SEND_CONFIG_REG(RESET_CMD);
    SET_RECV_CONFIG _REG(RESET_CMD);
        /* init a429 port speed */
    for(index = 0; index < E_RECV_PORT_COUNT; index++)
        {
            speed = (speed | (s_tRecvAddr[index].speed<<index));
        }
    for(index = 0; index < E_SEND_PORT_COUNT; index++)
        {
            speed = (speed | (s_tSendAddr[index].speed<<(index + 8)));
        }
        SET_CONFIG_REG(speed);
    }
```

2.2　驱动的配置

ARINC429 驱动的配置通过组建描述文件 40a429.cdf、a429.dc 和 a429.dr 文件来实现. 其中 a429.dc 文件提供了驱动注册函数 a429Register 的声明, a429.dr 文件提供了 a429Register 的调用 C 代码片段, 40a429.cdf 文件则提供了集成 ARINC429 驱动所需要的相关信息. 40a429.cdf 的具体内容如下:

```
Component DRV_CLASS_CETCAA429 {
        NAME            a429 VxBus driver
        SYNOPSIS        a429 3rd-party VxBus driver provided by CETCA
        _CHILDREN       FOLDER_DRIVERS
        MODULES         a429.o
        REQUIRES        INCLUDE_VXBUS \
    INCLUDE_PLB_BUS
        _INIT_ORDER hardWareInterFaceBusInit
        HDR_FILES       ../3rdparty/cetca/a429/a429.h
        INIT_RTN        a429Register();
        PROTOTYPE       void a429Register (void);
        INIT_AFTER      INCLUDE_PLB_BUS
        _CHILDREN       FOLDER_DRIVERS
    }
```

由于 ARINC429 驱动采用了 PLB 总线类型，所以需要在 hwconf.c 文件中定义 hcfResource 结构[7]，添加 ARINC429 模块板卡需要的相关资源参数．然后维护 hcfDeviceList 表，添加 ARINC429 模块板卡的名称、单元号、总线类型以及其单元号．

2.3 驱动的编译与加载

ARINC429 驱动需要在 wrenv 控制台下采用命令行的方式进行编译，主要是完成 vxbUsrCmdLine.c 和 CxrCat.txt 两个文件的生成和驱动可执行代码 a429.o 的生成．具体命令如下:

```
cd  installDir\vxworks-6.x\target\config\comps\src\hwif
make vxbUsrCmdLine.c
cd  installDir\vxworks-6.x\target\config\comps\vxWorks
del CxrCat.txt
make
cd installDir\vxworks-6.x\target\3rdparty\cetca\a429
make CPU=cpuName TOOL=tool
```

需要注意的是由于采用了 driverControl 驱动方法，由于默认系统并未编译该方法定义文件，需要在 installDir \vxworks-6.x\target\src\hwif\methods 目录下手动编辑 Makefile，在 OBJS_COMMON= \后加入 driverControl.o \并使用命令 make CPU=cpuName TOOL=tool 手动编译 driverControl 驱动方法到 VxWorks 库.

对 ARINC429 驱动以及接口板卡的实际测试采用了回环测试的方法，4 路输入通道与 4 路输出通道分别进行直接连接．使用测试程序操作输出通道以周期 200ms 输出 ARINC429 数据，并从对应的输入通道接收数据．所有通道速率均配置为 100kbit/s．实际测试中驱动程序工作稳定，未出现丢包和数据错误的情况．测试数据输出界面如图 5 所示.

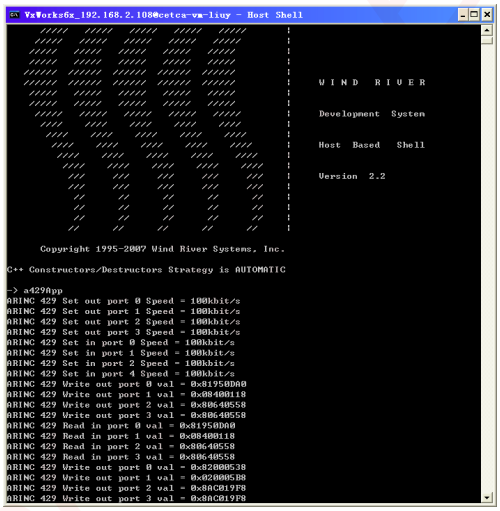

图 5　驱动测试数据输出界面

3　结语

采用 VxBus 驱动架构来实现的 ARINC429 接口驱动，通过实际项目测试验证，完全可以满足航空等领域对 ARINC429 接口数据收发的需求．后续还可根据需要，在最小的改动之下加入中断机制以及增加 ARINC429 接口板卡数量．虽然与传统的驱动架构开发工作相比增加了驱动配置工作量，但驱动的扩展性能和可移植性能具有了较大的提高.

参考文献

1 赵永钢,韩国义.基于 VxBus 的设备驱动开发.微型机与应用,2010,29(18):5-7.

2 WindRivcr Systems. VxWorks Device Driver Developer's Guide (Volume 1-6.8). WindRiver Systems, Inc. 2010.

3 WindRiver systems. VxWorks Bsp Developer's Guide 6.8. WindRiver Systems, Inc. 2010

4 曲建清,陈欣,吕迅竑.基于单片机和 CPLD 的 ARINC429 接口设计.计算机测量与控制,2009,17(3):558-560.

5 雷海鹏,刘久.基于 PCI 的 ARINC429 总线适配器设计与驱动开发.测控技术,2005,24(1):43-46.

6 WindRivcr Systems. VxWorks Device Driver Developer's Guide (Volume 2-6.8). WindRiver Systems, Inc. 2010.

7 WindRivcr Systems. VxWorks Device Driver Developer's Guide (Volume 3-6.8). WindRiver Systems, Inc. 2010.

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

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

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）

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

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