

文章编号:1001-9081(2005)02-0273-03

嵌入式 CAN-Ethernet 网关的设计与实现

杨波^{1,2}, 徐成¹

(1. 湖南省嵌入式计算及系统重点实验室, 湖南长沙 410082; 2. 湖南大学软件学院, 湖南长沙 410082)
(yangbo9981@163.com)

摘要:介绍了以太网与 CAN 现场总线网间嵌入式网关的软硬件结构设计, 描述了 uClinux 上 CAN 设备驱动程序的处理流程及其设计方法和技巧。针对 CAN 协议的特点, 为不同实时等级的信息报文设立四个分组, 设计了多帧数据发送报文结构, 为设备驱动程序的收发缓冲区设计合理的数据结构和管理方法, 提高了 CAN 通讯效率。

关键词:嵌入式系统; uClinux; CAN 现场总线; 设备驱动程序

中图分类号: TP393.02 **文献标识码:** A

Design and implementation of embedded CAN-Ethernet gateway

YANG Bo^{1,2}, XU Cheng¹

(1. Hunan Provincial Key Laboratory of Embedded Computing and System, Changsha Hunan 410082, China;
2. College of Software, Hunan University, Changsha Hunan 410082, China)

Abstract: The architecture of software and hardware of the embedded CAN-Ethernet gateway were introduced, and the principle, the designing methods and technoloques of the CAN Device Driver in uClinux were described. According to the features of the CAN protocol, data package was classified into four groups with different real-time request; the structure of multi-frame was proposed to satisfy the request of sending mass data; the data structure and the method of management for the buffer of the CAN Device Driver were designed to improve the capability of communication.

Key words: embedded system; uClinux; CAN bus; device driver program

0 引言

CAN(Contoller Area Network)现场总线是一种支持分布式控制系统或实时性控制的串行通信网络,具有成本低、可靠性高、抗干扰能力和实时性强等特性,是最普及的工业现场总线之一。随着电子商务的发展,人们对工业现场与 Internet 的整合提出了新的要求,希望通过现有的技术使工厂管理深入到控制现场。嵌入式 Linux 由于开发周期短、源代码开放、可以依据需要进行配置、强大的网络功能、可应用于多种硬件平台等特性,在嵌入式领域受到很大关注。基于嵌入式 Linux,可以开发出高性价比以太网与 CAN 现场总线网间的嵌入式网关,实现工业现场与 Internet 互联。

嵌入式 Linux 中有现成的 TCP/IP 协议栈,开发嵌入式 Linux 上的 Internet 互联程序比较容易。尽管 CAN 总线应用较广泛,但国内外基于嵌入式 Linux 上的 CAN 驱动程序开发也只是从近几年开始,开发出的一些 Linux 上的 CAN 驱动程序仅实现了 CAN 控制器的较弱功能,如简单通信控制功能,很难实现稍复杂的通信需求。但 CAN 网络通信的报文实时需求不同,不能实现对信息报文进行分组管理,确定报文发送的优先级;CAN 信息帧的字节宽度为 10 个字节,当需传送较大的数据量时,没有确定合理的规范。这些问题都是现场总线通信中的关键问题之一。因此有必要针对 CAN 报文信息分组功能及大数据量发送功能进行实现。

从上述分析可以知道,设计和实现嵌入式 CAN-Ethernet 网关,其关键技术是实现嵌入式 Linux 上的 CAN 驱动程序。本文将基于嵌入式 CAN-Ethernet 网关设计与实现,针对目前已有 CAN 驱动中信息报文分组管理功能及大数据量通信的不足,以 uClinux 下 CAN 设备驱动程序的开发作为其中的关键技术进行论述。

1 嵌入式网关软硬件结构

现场总线网络要接入以太网,首先要设计高性价比的嵌入式现场总线网关。嵌入式现场总线网关连接以太网、CAN 现场总线结构如图 1 所示。

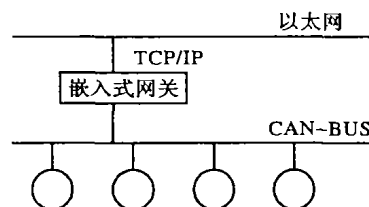


图 1 嵌入式网关的以太网和现场总线结构

1.1 嵌入式网关的硬件组成

主要硬件组成:处理器采用三星公司的 S3C4510B, CAN 采用 PHILIP 公司的 SJA1000^[1]。三星公司的 S3C4510B 是基于以太网应用系统的,内含一个由 ARM 公司设计的 16/32 位的 ARM7TDMI RISC 处理器核的微控制器,片内外围功能模

收稿日期:2004-07-23;修订日期:2004-10-27

作者简介:杨波(1974-),男,湖南城步人,硕士研究生,主要研究方向:嵌入式操作系统及现场总线研究;徐成(1962-),男,湖北蕲春人,副教授,主要研究方向:嵌入式系统。

块包含支持 100/10MB/S 工作速率的 Ethernet 控制器。SJA1000^[1] 是 PHILIP 公司开发的独立 CAN 控制器,内部集成了 CAN 协议的物理层与数据链路层,支持新的操作模式——PeliCAN,这种模式支持具有很多新特性的 CAN2.0B 协议。嵌入式网关通过以太网控制器接口与以太网相连,通过 CAN 控制器接口与现场总线相连,它的框图如图 2 所示。

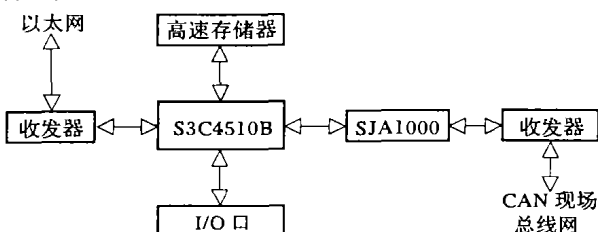


图2 嵌入式网关框图

1.2 嵌入式网关的软件部分

嵌入式操作系统采用 uClinux,它是一个符合 GNU/ GPL 的开放源代码项目,专为微控制领域而设计的。标准的 Linux 内核支持虚拟存储器,控制着处理器的内存管理单元(MMU)将虚拟内存映射为物理内存。嵌入式 CPU 为了降低硬件成本,提高系统性能,取消了 MMU 部件,本文用到的 S3C4510B 属于这类处理器,而 uClinux 正是为这类 CPU 而设计的。uClinux 可以说是标准 Linux 的小型化修正版,它支持 TCP/IP 协议,同标准 Linux 一样具有非常强大的网络功能,我们可以直接采用其自带的以太网驱动程序。uClinux 上的 CAN 控制器设备驱动程序需要自己编写。

2 CAN 设备驱动程序的设计

控制器局域网 CAN 是德国 Bosch 公司为汽车应用而开发的一种多主机局部网络系统,采用双线串行通信方式工作。具有较强的检错功能,可在高噪声干扰环境中使用,其最高通信速率可达 1MB/s。CAN 协议^[2]规定:收发报文帧长为 10 个字节,其数据长度为 0~8 个字节;工作时通过报文标识符确定总线访问优先权,高优先级报文具有较低延迟时间;发送期间若丢失仲裁或由于出错而被破坏的报文可自动重发;具有广播和成组报文的能力。

在嵌入式 Linux 操作系统下使用 CAN,必需要设计 Linux 上的 CAN 驱动程序。操作系统的作用之一就是向用户掩盖硬件的特殊性,使应用程序与底层具体的物理设备无关,设备驱动程序就是应用程序与具体硬件的桥梁。Linux 支持三类硬件设备:字符、块、网络设备,它们驱动程序的编写方法基本相同。SJA1000 CAN 控制器集成了 CAN 协议的物理层与数据链路层,CAN 控制器属于字符型设备。当 SJA1000 接收一个报文时,数据保存在接收缓存器中,并产生一个接收中断;发送报文,先将数据送入 SJA1000 的发送缓冲器中,再将数据串行化发送到 CAN 总线上。

2.1 CAN 设备驱动程序的处理流程

CAN 设备驱动程序中最重要的是中断处理例程。中断处理例程首先是被来自 CAN 控制器的硬件中断唤起,然后中断处理例程分辨中断类型(发送或接收)。驱动程序处理流程^[3,4]如图 3 所示。

如果中断类型为接收数据,则中断处理例程调用接收数据处理函数从 CAN 控制器读取数据到接收 FIFO,最后用户

使用系统调用从接收 FIFO 中读到完整的一帧数据。当中断类型为发送数据时,则中断处理例程调用数据发送处理函数,从发送 FIFO 中读取数据送入 CAN 控制器。发送 FIFO 为空且应用程序向发送 FIFO 写入数据时,数据发送处理函数将自动调用。

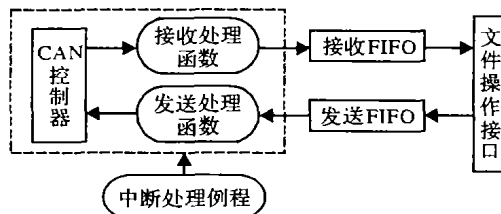


图3 CAN 设备驱动程序处理流程

2.2 报文分组

根据实际应用,通过 11 位 CAN 标识 ID 将网络上传递的信息报文分成四组,分别是为信息组 1~4,这四组信息报文格式如图 4。

| 标识 ID 各个位的意义 | | | | | | | | | | | 范围 | 标识用途 | |
|--------------|----------|----------|---|---|----------|---|---|----------|---|---|---------|---------|-------|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |
| 0 | 组 1 信息标识 | | | | 源 MAC ID | | | | | | 000~3ff | 信息组 1 | |
| 1 | 0 | MAC ID | | | | | | 组 2 信息标识 | | | | 400~5ff | 信息组 2 |
| 1 | 1 | 组 3 信息标识 | | | 源 MAC ID | | | | | | 600~7bf | 信息组 3 | |
| 1 | 1 | 1 | 1 | 1 | 组 4 信息标识 | | | | | | 7c0~7ef | 信息组 4 | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | × | × | × | × | 7f0~7ff | 无效标识 | |

图4 分组信息报文标识 ID 格式

根据 CAN 通信协议的无损位仲裁算法,信息组 1 的优先级是最高的,依次类推,信息组 4 的优先级最低。这样在设计系统时,可根据实时性要求将信息分配到相应的优先级信息组中,实现了信息分组功能。

2.3 多帧报文发送

由于 CAN 信息帧只有 10 个字节,它由 1 个字节的标识位,1 个字节的 RTR 和 DLC 位,8 个字节的数据区组成。这样就限制了一次传递信息的大小,为了传输较大的数据,应设计多帧传输格式。将较大的数据由发送方将数据拆分成多个子信息报文顺序发送出去,数据的接收方检测子报文接收是否完整,如果完整且对子报文进行组合,否则认为出错,放弃已接收的报文。设计的多帧报文格式如图 5。

| | | | | | | | | |
|---|----------------|-----|--------|-----|-----|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 11 位标识（包括 CID） | | | | | | | |
| 1 | | | | RTR | DLC | | | |
| 2 | Frag | XID | MAC ID | | | | | |
| 3 | 多帧类型 | | 帧计数 | | | | | |
| 4 | 完成服务所需参数 | | | | | | | |
| 到 | | | | | | | | |
| 9 | | | | | | | | |

图5 CAN 信息报文多帧格式

多帧类型表示该子报文在全帧中的位置(0 表示单帧,1

表示第一帧,2表示中间帧,3表示结束帧),帧计数表示当前发送的帧的编号,接收方必须判断其连续性,一旦不连续就认为有帧丢失,然后放弃所有已接收的报文。

2.4 收发缓冲区管理

收发缓冲区一般都采用环形 FIFO 队列,使得读写可以并发执行。每一次读写操作都要判断队列是否可操作。依据 CAN 通信协议,CAN 控制器每次只接收或发送一帧数据,一帧 CAN 数据是 10 个字节,所以我们在每次需要内存缓冲区时,直接分配长度为 10 个字节数据块,这 10 个字节地址是线性连续的。在向缓冲区读写数据时,只要判断一次缓冲区可否操作,并获得块首地址,就可以一次读写 10 个字节的数据,减少了重复性条件判断,提高了效率。在 CAN 设备驱动程序里,设立这样一个 CAN_FIFO_buffer 数据结构作为收发数据缓冲区:

```
struct CAN_FIFO_Buffer
{
    int head;
    int tail;
    canmsg_t data[CAN_BUFFER_SIZE];
    int usedbytes;
}
```

canmsg_t 代表 CAN 信息帧数据结构,usedbytes 代表当前缓冲区有多少字节被占用,使用 usedbytes 可以很方便判断缓冲区满或空,usedbytes = 0 为空,usedbytes = CAN_BUFFER_SIZE 为满。

3 uClinux 上 CAN 驱动程序的实现

Linux 的一个基本特点是它抽象了对硬件设备的管理,为用户程序提供一个统一的、抽象的、虚拟的文件系统界面(VFS),这个抽象的界面主要由一组标准的、抽象的文件操作构成^[5],以系统调用的形式提供给用户程序,如 read()、write()、open()等等。系统中每一个设备都用一个特殊设备文件名来表示。这个抽象界面的主体就是一个 file_operations 数据结构。每种文件系统都有自己的 file_operations 数据结构,结构中的成份几乎全是函数指针,所以实际上是个函数跳转表,如 read 就指向具体文件系统用来实现读文件操作的入口函数。如果具体的文件系统不支持某种操作,相应的函数指针就设为 NULL。可以说 file_operations 实现了标准文件操作到硬件设备操作的映射,所以设计 CAN 设备驱动程序必需要实现这个接口所定义的部分函数。根据系统需求,CAN 设备驱动程序实现了 file_operations 结构部分重要的设备方法,采用标记化格式声明它的 file_operations 结构如下:

```
struct file_operations CAN_fops =
{
    read: CAN_read,
    write: CAN_write,
    ioctl: CAN_ioctl,
    open: CAN_open,
    release: CAN_release,
};
```

CAN_read 负责从接收 FIFO 读取数据;CAN_write 负责向发送 FIFO 写入数据;CAN_release 负责关闭 CAN 控制器;CAN_ioctl 负责向 CAN 控制器发各种操作命令;CAN_open 负责打开 CAN 控制器,检查 CAN 控制器是否打开,并使用函数 request_irq()向系统登记,安装中断处理程序。CAN_open 函

数代码片断编写如下:

```
int CAN_open(struct inode *inode, struct file *file)
{
    .....
    if (request_irq(CAN_IRQ, &CAN_handler, SA_INTERRUPT,
        "CAN - irq", "sja1000")) {
        printk("s3c4510 - sja1000: Can't get irq %d\n", CAN_IRQ);
        return -EAGAIN;
    }
}
```

3.1 驱动程序初始化

在实现了 CAN_fops 结构体内的各个入口点函数后,还要编写一个 CAN 设备驱动程序初始化函数,在内核启动时登记这个驱动程序,并完成 CAN 控制器的初始化设置。初始化程序代码片断编写如下:

```
void init_CAN_bus(viod)
{
    .....
    //CAN 控制器内部寄存器初始化
    if (result = register_chrdev(254, "CAN_bus", &CAN_fops))
        printk("Error: %d init_CAN_bus() can't get Major\n", result);
}
```

函数 register_chrdev 是设备驱动程序登记函数,负责向系统注册字符型设备驱动程序。同时,我们需在/dev下创建设备文件名,可用命令 mknod /dev/CAN_bus c 254 0 创建。为了让内核在启动时初始化程序,在/uClinux /linux/drivers /char /mem.c 中的 chr_dev_init()函数中加入对初始化函数的调用。

3.2 设备驱动程序编译

最后一步对设备驱动程序进行编译,嵌入式 Linux 不能像桌面 Linux 那样灵活使用命令动态加载/卸载设备模块,所以要将设备驱动程序以静态的方法编译进内核。具体操作如下,先将编写的驱动程序文件(假设名为 CANbus.c)复制到/uClinux/linux/drivers /char/目录下,然后修改/uClinux /linux/drive - rs/char 目录下的 makefile 文件^[6],在里面添加一行:L_OBJS + = CANbus.o。

4 结语

本文基于嵌入式 CAN-Ethernet 网关的设计与实现,并以 uClinux 下 CAN 设备驱动程序的开发作为其中的关键技术进行论述。考虑到嵌入式系统的特点及已有 CAN 驱动的不足,为 CAN 报文设立四个分组,可依据报文的实时性需要放入相应的优先级组中发送,添加了信息分组功能;同时设计了多帧报文发送格式,方便了大数据量的传送;为收发数据缓冲区设计了合理的数据结构和管理方法,提高了 CAN 设备通信的吞吐量。由于嵌入式 Linux 及 CAN 的众多优越性,将在工业及生活的各个领域具有更广泛的应用前景。

参考文献:

- [1] PHILIPS Semiconductors. SJA1000 Stand-alone CAN controller[Z], 1999, 8.
- [2] BOSCH. CAN SPECIFICATION(Version 2.0)[S], 1991, 9.
- [3] CANpie User Manual[EB/OL]. http://www.micr-ocontrol.net/CANpie/download/cp_user_manual.pdf. 2003-01.
- [4] MENDOZA P, VIA J. Developing CAN based network on RT-Linux[J]. IEEE, 2001, 2(8): 161-167.
- [5] RUBINI A, CORBET J. Linux Device Drivers[M]. O'REILLY, USA, 2002.
- [6] 李驹光,聂雪媛. ARM 应用开发系统详解[M]. 北京:清华大学出版社, 2003.