基于 Arm CAN 现场总线的油库实时监控系统①

王博文

(重庆大学 通信工程学院，重庆 400000)

摘　要：针对油库管理的实时性和可靠性等要求，在论述 CAN 总线驱动程序的基础上提出了基于 CAN 总线的分布式控制系统方案。主要介绍了在核心数据处理单元 S3C2410 上利用扩展设备 CAN 控制器实现总线报文的过滤与收发。其中包括 CAN 控制器 SJA1000 的工作原理，以及 CAN 总线访问控制的一般方法，CAN 总线驱动程序的设计。实验证明，所论述的基于 ARM CAN 的驱动程序设计和报文收发过滤技术具有普遍意义。

关键词：油库监控；ARM Linux；CAN 总线；Linux 驱动程序；SJA1000 控制器

Real-Time Oil Depot Monitoring System Based on the Arm and Can Bus

WANG Bo-Wen

(Department of Communication Engineering, Chongqing University, Chongqing 400000, China)

Abstract: According to the safety of oil and real-time reliability management requirements, this paper puts forward the CAN Bus distributed control system scheme. It introduces the core data processing unit S3C2410 in the use of expansion devices CAN Controller realizing sending, receiving and filtering bus packet, including the working principle of CAN Controller SJA1000, the methods of the general CAN Bus access controlling, and the design of CAN Bus Driver Program. Experimental results show that the technology in this paper, based on ARM CAN driver design and message transceiving and filtering is universal.

Keywords: depot monitoring; ARM Linux; CAN bus; Linux driver program; SJA1000 controller

1　引言

CAN（Controller Area Network）是一种支持分布式控制和实时控制的网络。它具有结构简单、传输距离远、可进行数据错误监测、通信方式灵活等特点。在工业控制和自动化领域应用十分广泛。而 Arm linux 是一种源代码开放的操作系统。其功能强大、结构简单，可以按照需求配置，可以根据实际需要裁减。因而 Arm linux 做为 CAN 总线的软硬件环境相对其他微处理器和操作系统具有无可比拟的优势。

本文所论述的基于 Arm linux 操作系统的 CAN 总线分布式控制网络充分利用 CAN 总线的特点和 Arm linux 系统的优势来解决油库管理系统中存在的信息管理和传输问题。主要针对 CAN 控制器的工作原理、CAN 总线在多主通信模式下的报文传输、以及 CAN 总线驱动程序的设计、节点之间数据命令信息的发送接收等问题展开了论述。最后提出了系统的整体构建方案，以及节点在多传感器的情况下，节点之间数据与命令信息的收发技术。

2　系统的软硬件结构原理与设计

选用双绞线作为传输介质，采用 CAN 控制器和 CAN 收发器实现数据帧的收发、差错监测、总线仲裁等功能。利用 ARM 处理器作为微控制器，控制 CAN 控制器的工作和传感器的工作。在 ARM 处理器上，根据 SJA1000 的工作方式[1]实现 CAN 总线驱动程序，实现应用层的人机管理功能。

2.1 嵌入式系统的硬件组成

系统的硬件主要有三星公司的 S3C2410 处理器，SJA1000 CAN 控制器，以及高速 can 收发器 TJA1050，周立工的 ZLG7290 键盘、LED 控制芯片等。

2.2 SJA1000 的设置

SJA1000 兼容 Can2.0 版本，有两种工作模式：

① 收稿时间:2010-07-23;收到修改稿时间:2010-08-20

BASICCAN 模式和 PELICAN 模式。BASICCAN 有 0-31 共 32 个寄存器可用。PELICAN 共有 0-127 个共 128 个寄存器可用。要实现 can 总线的通信，主要就是如何配置这些寄存器，本系统设置 SJA1000 在 BASICCAN 模式下工作。

图 1 为 SJA1000 的工作结构图。时钟分频寄存器 OCR 用于设定 SJA1000 工作于 BASICCAN 模式还是 PELICAN 模式。在工作模式下，控制寄存器 CR 用于控制 can 控制器的行为，可读可写；命令寄存器 CMR 只能写；状态寄存器 SR 只能读；中断寄存器 IR 用于中断源的识别，只读。中断允许寄存器 IER 用于允许或禁止不同的中断源产生，可读可写。验收代码寄存器 ACR 和验收屏蔽寄存器 AMR 共同构成验收滤波器。

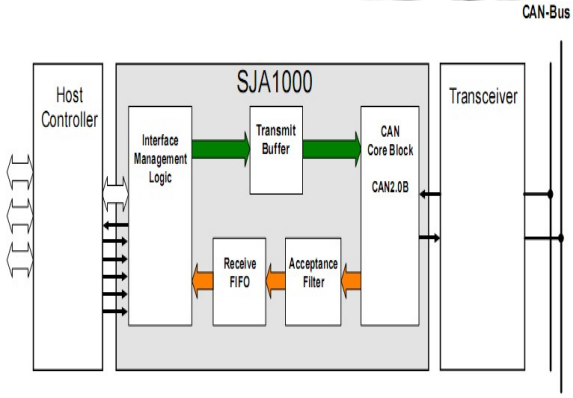


图 1 SJA1000 工作结构图

当 SJA1000 工作在 BASICCAN 模式下，当需要发信息时，若发送缓冲器 TX-Buffer 空闲，则由 CPU 控制信息写入 TX-Buffer，再由 CMR 控制发送。当接收缓冲器 RX-Buffer 未满时，且信息通过过滤后，则接收到的信息被写入 RX-Buffer。可以通过两种方法读取接收到的信息，一种方法是在中断未被使能的情况下，通过 SJA1000 向 CPU 发出中断信号，CPU 通过 SR 及 IR 识别该中断，并读取数据释放接收缓冲器。另一种方法是直接读取 SR 查询 RXFIFO 的状态，当有信息接收时，读取该信号并释放接收缓冲器。

2.3 can 总线的报文传输

报文传输由四个不同的帧表示和控制：数据帧、远程帧、错误帧、过载帧。数据帧和远程帧需要在 CPU 的控制下发出，而出错帧和过载帧则是在错误发生或者超载发生时自动进行的。数据帧有 7 个不同的位场组成：帧起始、仲裁场、控制场、数据场、CRC 场、应答场、帧结尾。远程帧和数据帧相比，仅仅少了数据场。当请求数据源节点需要目的节点给它发送数据时，先向目的节点发送一数据帧，目的节点收到该远程帧后，把所要发送的数据以数据帧的形式发送给请求数据源节点，完成数据的请求发送[2]。

由于 CAN 总线上传送的消息是经过 CAN 控制器的报文。CAN 总线可工作于多主模式，也就是任何一个节点向其他节点发送信息。CAN 报文的格式以 BASICCAN 模式为例，CAN 报文标识符 ID 的长度为 11，位于要发送信息序列的最前面（在数据链路层，标识符 ID 位于数据帧或远程帧的仲裁场里），而接收滤波器根据 SJA1000 在复位模式下设定的初始值，来判断是否对报文接收或者抛弃。

在 BASICCAN 模式下，假若 ACR 的值为 01010101，而 AMR 的值为 00111000，11 位 ID 信息为 01XXX101XXX。在验收屏蔽寄存器 1 的位置上，ID 在这位的任何值都被允许，对于三个最低位同样。因此 64 个不同的 ID 就被接收，其他位置的位必须等于验收代码寄存器所在的位的值。因此验收滤波器有效实现报文的过滤，从而根据报文 ID 判断是否对此报文接收或者过滤。

2.4 can 总线驱动程序的设计

由于 Arm Linux 内核没有包含 CAN 控制器的驱动程序，所以必须根据 SJA1000 的工作原理，以及 CAN 总线的特点设计驱动程序。驱动程序的设计是该系统架构的关键技术所在。

Linux 驱动程序是 Linux 内核的一部分，其主要功能是直接操作硬件设备，并为应用程序提供入口，透明访问硬件的机制。Linux 驱动程序支持 3 种类型的设备：字符设备、块设备、网络设备。字符设备驱动是 Linux 中最常见的设备驱动程序，诸如键盘、LED、LCD 等设备驱动都属于字符设备驱动，即应用程序对字符设备每次的 I/O 操作，都会直接传递给内核对应的驱动程序。独立的 CAN 控制器 SJA1000 实现了 CAN 协议的物理层与数据链路层，属于字符设备的范畴。

2.4.1 驱动程序的初始化

CAN 驱动程序的初始化除了对设备进行注册，配置总线带宽外，另一个重要的步骤就是对 SJA1000 进行初始化，调用下列函数完成对驱动程序的初始化。在 BASICCAN 模式下，对 SJA1000 的初始化必须在复位模式下进行，通过将 0x01 送控制寄存器进入复位

模式。进而根据需要配置时钟分频寄存器、验收代码寄存器、验收屏蔽寄存器、总线定时器、输出控制寄存器等[3]。所有寄存器配置完毕，将 ox1e 送控制寄存器进入工作模式。

```
int init_sja1000(void)
{int i;
 write_can_reg(0x01, ControlReg);
 udelay(10);
 write_can_reg(0x48, ClockDivideReg);
 udelay(10);
 write_can_reg(0, AcceptCodeReg);
 write_can_reg(0xff, AcceptMaskReg);
 write_can_reg(0x85, BusTiming0Reg);
 write_can_reg(0xb4, BusTiming1Reg);
 write_can_reg(0x1a, OutControlReg);
 write_can_reg(0x1e, ControlReg);
 udelay(10);
 udelay(10);
 return 0;}
```

2.4.2 驱动程序文件系统接口的实现

CAN 驱动程序最终给应用程序提供一个流控制接口，主要包括：open、close、release、read、write、icctl 等操作。而虚拟文件系统的主题就是结构体 file_operations，每一种驱动程序都有自己的 file_operations。结构体中的每一个成员都是一个函数指针，实现不同的操作功能。例如：open 函数实现打开一个设备文件，返回该设备的 ID 号。说 file_operations 函数实现了标准文件操作到硬件设备操作的映射。在 2.6.16 内核里，结构体 file_operations 的定义在 Include/Linux/fs.h 文件里：

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*open) (struct inode *, struct file *);};
```

根据 can 控制器的原理，can 设备驱动程序实现了上述数据结构中的一部分操作。

```
struct file_operations can_fops = {
    owner:          THIS_MODULE,
    open:           can_open,
    read:       can_read,
    write:          can_write,
    ioctl:          can_ioctl,
    release:        can_release,    /* a.k.a. close */
};
```

在 can_fops 中声明的每个成员的功能如下：

(1) can_open 负责初始化等待队列、限制 can 总线打开次数、清空发送缓冲和接收缓冲区。

(2) can_read 负责从接收寄存器读取数据。接收数据之前用 enable_irq 函数使能总线中断，然后用 disable_irq 函数屏蔽总线中断。用 read_can_reg 函数把 SJA1000 数据寄存器 RxBuffer1- RxBuffer10 的 ID 及数据信息读取到 CAN_BUS_BUF 数组。最后调用 write_can_reg 函数向命令寄存器写入命令，清空数据寄存器的内容。然后用 copy_to_user 函数把 CAN_BUS_BUF 数组的数据放到用户空间。具体代码如下：

```
enable_irq(IRQ_CAN_BUS);
READ_DATA_STATUS                = DATA_GONE;
sleep_on(&CAN_wq);
  disable_irq(IRQ_CAN_BUS);
CAN_BUS_BUF[1]                  = read_can_reg(RxBuffer1);      // Read the data
CAN_BUS_BUF[2]                  = read_can_reg(RxBuffer2);
```

```
…
CAN_BUS_BUF[10]=
read_can_reg(RxBuffer10);
write_can_reg(0x04, CommandReg);
copy_to_user(buf,CAN_BUS_BUF,11);
return 0 ;
```

(3) can_write 负责向发送缓冲区写入数据，然后通过配置命令寄存器将数据发送出去，具体代码实现和 can_read 雷同。

(4) can_ioctl 负责读写操作之外的工作特性的配置，主要完成设置工作模式、总线频率等。

(5) can_release 负责 can 设备关闭时的操作，如还原打开次数，关闭中断等。

3 系统总体设计

该方案把整个油库管理归纳为一个 CAN 总线系统，包括一个总的监测控制中心(主节点)和各个子监测控制中心(子节点)。设一工作室作为主监测控制中心，各个油罐作为子监测控制中心。子节点和主节点之间通过总线方式连接。

3.1 系统网络的物理实现

采用 S3C2410 作为主控制器，SJA1000 作为 CAN 控制器。所有的微处理器直接连接到 CAN 控制器。而 CAN 控制器则通过 CAN 收发器连接到 CAN 总线网络[4]上。CAN 收发器实现 CAN 数据的过滤和接收，它将 CAN 控制器连接到物理总线上（双绞线、光纤等）。在图 2 的系统总体图中，各节点监测控制系统由 TJA1050, SJA1000,S3C2410 处理器三项组成。

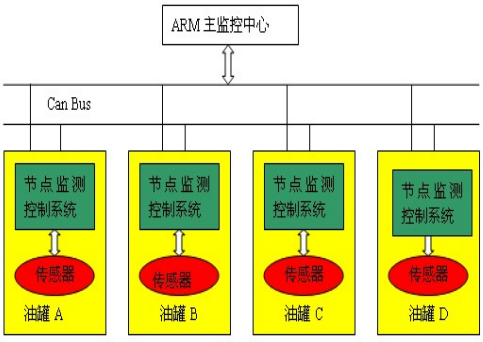


图 2 系统总体结构图

各个子节点上面含有如：温度、压力、湿度等传感器，使用 ARM 控制传感器工作。

在 LCD 上面实时的读出诸如温度、压力、湿度等数据。可以用键盘设置报警门限参数，如设置温度>50 度自动报警，报警器鸣笛。报警的同时把该油罐的位置、油的状态参数传递给主监控中心。主监控中心控制 LCD 显示该油罐的位置、油量等信息方便工作人员处理。

主监控中心实时监测各个油罐的油的信息诸如：油量、温度、压力等。并将这些数据存储起来，方便查阅。根据实际情况设置各个子结点传感器的工作状态，实现打开、关闭等操作。

对于子节点之间数据命令信息的传输，可以在根据实际情况需要的情况下进行设置。

3.2 节点间数据命令信息的传输

对于节点之间数据命令的传输可以通过设置收发缓冲来完成。收发缓冲器的设置[5]根据在 BASICCAN 模式下 SJA1000 的原理设置如下表 1。

表 1 BASICCAN 模式下的缓冲器

收发缓冲器地址	所含的字节数	具体 bit 位的说明
TX-Buffer：10H RX-Buffer：20H	标志符 ID 的第 1 个字节	标识符 ID 的前 8 bit 位
TX-Buffer：11H RX-Buffer：21H	标志符 ID 的第 2 个字节	前 3 位是 11 位标识符 ID 的后三位，1 个远程传输请求 bit 位，后 4bit 位表示数据的长度：如 0100=（4 位）
TX-Buffer：12-19H RX-Buffer：22-29H	数据位 1-8 字节	最大数据长度 8 字节

根据标识符 ID 接收需要的报文，报文的 ID 决定接收的报文是来自哪个节点油罐。在此系统里，我们对收发缓冲器的数据单元部分做了改进如表 2。

表 2 按照系统需求定义的缓冲器

数据位	地址	前四位	后四位
TX[0]	TX-Buffer：12H	标识命令或数据	标识传感器类型
TX[1]-TX[7]	TX-Buffer：13-19H	数据信息	
RX[0]	RX-Buffer：22H	标识命令或数据	标识传感器类型
RX[1]-TX[7]	RX-Buffer：23-29H	命令信息	

我们规定收发数据缓冲器的第一个数据位 TX[0]、RX[0]的前四位为标识命令或数据的位。如果源节点向目的节点发送的是数据，其值为 0000XXXX。如果发送的是命令，其为 1111XXXX。X 表示 bit 位的值不定。在接收节点，CPU 读取接收缓冲器数据位 RX[0]，根据其值与 F0H 按位与运算，根据结果判断接收的是数据还是命令指令(00H 时，为数据信息。F0H 时，为命令信息)。

发送数据指令时，设置 TX[0]前四位的值为 0000，同时设置 TX[1]后四位的值为传感器类型标识符，温

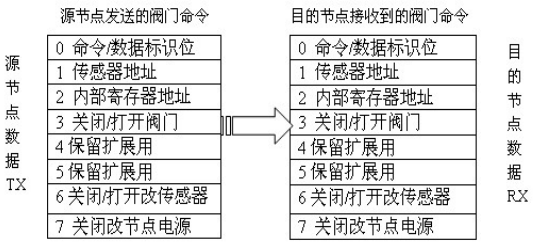
度传感器为 1000、湿度 1001，压力 1010、油量为 1011 等等。最多可以标识 16 个传感器。设置 TX[1]-TX[7]为所标识传感器的参数值，如传感器 I/O 地址、传感器内部数据寄存器地址、温度、压力、油量、湿度等。在接收节点，CPU 读取 RX[1]-RX[7]的值得到传感器的参数。

发送命令指令时，设置 TX[0]前四位的值为 1111。同时设置收发缓冲器的后四位的值为传感器类型标识符，阀门传感器 1000、灭火器传感器 1001 等等。设置 TX[1]-TX[7]为对目的节点传感器进行操作的命令指令。目的节点的 CPU 读取 RX[2]-RX[7]内的值判断对节点油罐的操作。对所有传感器的命令指令，为了在传感器失灵的情况下保证数据通信的质量，当源节点设置 TX[7]=FFH 时，目的节点读取到 RX[7]=FFH 时，关闭目的节点的电源。

假若其中一个油罐出现泄露、起火等较大险情，无须到距离比较远的主控制中心关闭该油罐的阀门，也无须冒险冲到该油罐处处理险情。只需找到最近一安全的节点处，便对此油罐做险情处理。当工作人员走到油罐 B 时，发现油罐 A 着火。此时不需要冲到油罐 A 处将阀门关闭，只需要在 B 处操作相应的节点控制器便实现关闭油罐 A 阀门的操作。这主要是因为 CAN 总线网络中的节点可以在多主模式下工作，节点之间的关系是对等的。

在 B 处向 A 发送带有命令的报文，如下表 3 所示。在 A 通过接收滤波器正确接收 B 发送的报文，CPU 读取 RX[0]=11111000，判断改指令为命令信息，相应的传感器为阀门传感器 1000。读取 RX[1]=70H，RX[3]=FFH 时，执行对 I/O 地址为 70H 的传感器的操作，从而关闭 A 油罐的阀门。

表 3　命令报文信息格式



由于线路在传输过程中有衰减，所以采用 HUB 来放大信号，保证数据的 QOS。

4　结束语

本文主要论述了在油库管理系统中基于 Arm Linux 的 CAN 通信网络报文收发过滤技术及多节点之间通信技术的软硬件实现方法。该方法能有效的解决油库管理中信息传输的实时性和有效性问题。由于 Arm Linux 的诸多优点，其代码还会被不断完善，CAN 网络和 Arm linux 的组合将会在工业领域有更为广阔的应用前景。CAN 总线也将在电子、航天、消费等各个领域发挥越来越重要的作用。

参考文献

1 陈赜.ARM9 嵌入式技术及 Linux 高级实践教程.北京:北京航空航天出版社,2005.23－27.
2 吴明晖.基于 ARM 的嵌入式系统开发与应用.北京:人民邮电出版社,2004.98－102.
3 孙福春,宁滨.ARM7 嵌入式系统在车辆调度中的应用.微计算机信息(测量自动化),2005,21(3):78－79.
4 马忠梅.ARM 嵌入式处理器结构与应用基础.北京:北京航空航天大学出版社.35－38.
5 邬宽明.CAN 总线原理和应用系统设计.北京:航空航天大学出版社,1996.102－103.