计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):231−234 [doi: 10.15888/j.cnki.csa.006159]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

波特率自适应的 CAN 驱动在嵌入式 Linux 下的实现①

史小燕, 朱建鸿

(江南大学 物联网工程学院, 无锡 214122)

摘　要: 基于三星 S3C2410 芯片为主控制器的实验平台, 设计实现了一款匹配效率更高的波特率自适应 CAN 驱动. 文中介绍了 CAN 驱动结构并详细解释了波特率自适应 CAN 驱动的原理, 结合常用的轮询法和采样法进行自适应. 同时方案提出了新的改进, 将用户影响加入波特率自适应流程来提高驱动工作效率. 最后本文对该驱动进行了数据通讯测试以及性能分析.

关键词: 嵌入式 Linux; 波特率自适应; CAN 总线; 设备驱动

引用格式: 史小燕,朱建鸿.波特率自适应的 CAN 驱动在嵌入式 Linux 下的实现.计算机系统应用,2018,27(1):231–234. http://www.c-s-a.org.cn/1003-3254/6159.html

Realization of Baud Rate Adaptive CAN Driver Under Embedded Linux

SHI Xiao-Yan, ZHU Jian-Hong

(School of IoT Engineering, Jiangnan University, Wuxi 214122, China)

Abstract: Based on the experimental platform whose main controller is Samsung S3C2410 chip, this article designs and implements a more efficient baud rate adaptive CAN driver. In this paper, the CAN drive structure is introduced and the principle of baud rate adaptive CAN driver is explained in detail. This scheme is combined with the common polling method and sampling method. At the same time, a new scheme is proposed to improve the efficiency of the drive by adding the user's input to the baud rate adaptive process. Finally, this paper carries out the data communication test and the performance analysis for this driver.

Key words: embedded Linux; baud rate adaptive; CAN bus; device driver

控制器局部网 (CAN-CONTROLLER AREA NETWORK) 是 BOSCH 公司推出的一种多主机局部网[1]. 它是一种非常有效的分布式控制串行通信网络, 并且具有传输速度快、可靠性高、通信方式灵活等特点, 因此在工业控制现场被广泛应用[2]. CAN 总线工作时要求通讯双方波特率一致, 实现 CAN 波特率自适应将有效提高 CAN 总线的灵活性和使用效率. 本文实现了一种基于嵌入式 Linux 的波特率自适应方法, 该方法通过融合直接测量法和波特率表轮询法来加快波特率的配适速度; 同时, 加入波特率优先级的概念, 保存用户选择数据并将其利用到波特率初始值设置中, 增加了一次性命中正确波特率的概率.

1 CAN 总线硬件平台介绍

硬件平台采用三星 S3C2410 芯片作为系统微控制器. CAN 总线规范定义了 OSI 模型的数据链路层和物理层. 这两层通常由 CAN 总线控制器和 CAN 总线收发器实现[3]. 本文 CAN 总线控制器和收发器分别采用 Microchip 的 MCP2510 和 Philips 的 P82C50, 二者通过 SPI 总线实现与控制器的数据传输.

S3C241X 系列是由 Samsung 公司设计生产的低功耗, 高集成微处理器芯片, 能够对 WINCE, EPOC32, LINUX 系统提供支持. 芯片接口资源丰富, 可通过 SPI 同步串行接口和 MCP2510 相连. MCP2510 是带有 SPI 接口的 CAN 总线控制器, 支持回环、正常、监听、睡

① 收稿时间: 2017-04-11; 修改时间: 2017-04-26; 采用时间: 2017-05-10; csa 在线出版时间: 2017-12-22

眠、配置 5 种工作模式, 满足 CAN 总线使用需求. P82C50 是 Philips 公司生产的 can 总线收发器, 芯片作为 CAN 控制器与物理总线间的接口, 提供对总线的差动发送和接收功能. 当使用 P82C50 作为 CAN 收发器时, 同一网络中允许挂接 110 个节点. CAN 收发器, 控制器与微控制单元之间的关系如图 1 所示.

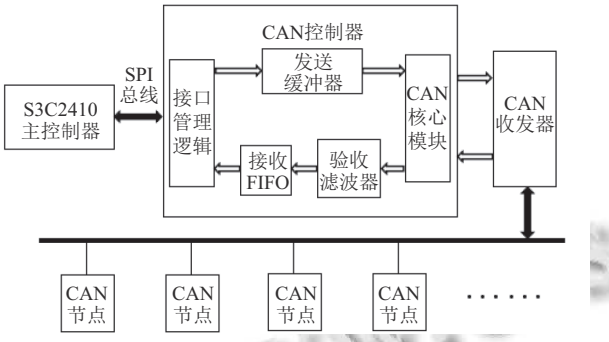

图 1 CAN 总线器件示意图

2 Linux 下 CAN 驱动软件结构

Linux 设备驱动是硬件设备和应用程序之间的接口, 它为用户提供了统一的设备访问方式. Linux 内核一般把驱动程序分为 4 种类型, 而 CAN 总线设备属于其中的字符型设备, 它以设备文件的方式建立在文件系统中的/dev 目录下, 并且可以像文件一样被访问[4].

Linux 下设备驱动程序可以以模块的形式动态的加载和卸载, 字符设备驱动程序一般包含 5 个部分: 头文件、file_operation 结构体变量、中断函数、加载函数以及卸载函数. file_operation 结构体变量存储驱动内核模块提供的对设备进行各种操作的函数的指针, 该结构体的每个域都对应着驱动内核模块用来处理某个被请求的事务的函数的地址. <linux/fs.h>对 file_operations 数据结构中各个变量做了详细定义[5]. 当用户调用如 open(), read() 等 API 接口时, 系统将参数传递给 file_operation 结构体的成员函数, 最终获得参数实控权的将是系统调用函数.

加载函数以及卸载函数为驱动提供了动态使用模式, 系统通过调用 module_init() 来加载驱动, 通过调用 module_exit() 来卸载驱动. module_exit() 只在动态编译时有意义, 因为静态加载时驱动已经被编译进内核, 无法进行卸载.

中断函数即中断服务程序, 它响应系统中断并作出相应的处理. 本文所设计的波特率自适应方式就是通过中断服务程序来实现. Linux 中断处理程序可以分为上半部和下半部[6]. 上部中断处理紧急中断任务, 它应该尽量短小简单以便程序能迅速处理完中断并及时返回. 底部中断用来处理较长的非紧急任务, linux2.6 版本内核提供了三种底部中断处理方式: softirq、tasklet 和 work queue. tasklet 和 softirq 都是基于软中断机制实现, 而 work queue 则是依靠内核线程实现. 这三种处理方式各有优缺点, 如果专注性能提高应使用 softirq, 对底部中断处理的调度有特殊要求则必须使用 work queue, 除此之外, 最优选择 tasklet 处理方式.

3 波特率自适应 CAN 驱动软件设计

CAN 波特率自适应的一般方法有两种: 轮询法和探测法. 本方案将结合这两种常用检测法来设计自适应 CAN 驱动.

轮询法即将常用波特率记录进一张波特率轮询表中, 然后将测试波特率按序填入 CAN 驱动, 通过判断能否正确接收 CAN 数据帧决定是否进行下一次轮询.

探测法即对接收数据进行采样, 通过采样数据来获得正确波特率. 由于 CAN 波特率有一定容错空间, 所以当 MCU 自带捕捉功能时, 即使采样波特率有一定误差, 很大程度上也能获得正确波特率.

这两种方法各有利弊, 前者可以保证在波特率表范围内找到正确值, 但因为要对数据依次轮询, 所以往往耗时较长. 后者虽然时耗短, 但对设备有特殊要求, 且如果偏差值超过波特率容错范围, 则会造成无法获得正确波特率的错误. 基于此本文将两种方法相结合, 实现一种平均耗时短、匹配成功率高的波特率自适应 CAN 驱动.

3.1 CAN 控制器 MCP2510

MCP2510 是一款独立 CAN 控制器, 它的存在大大降低了软件开发的难度. MCP2510 具有 8 个中断源, 寄存器 CANINTE 是中断使能寄存器, 它包含了各个中断源的中断使能位; 寄存器 CANINTF 是中断标志寄存器, 它包含了各个中断源的中断标志位. 寄存器各个位设置如表 1 所示.

表 1 中断寄存器 CANINTE, CANINTF

	Bit7	Bit6	Bit5	Bit4
INTE	MERRE	WAKIE	ERRIE	TX2IE
INTF	MERRF	WAKIF	ERRIF	TX2IF
	Bit3	Bit2	Bit1	Bit0
INTE	TX1IE	TX0IE	RX1IE	RX0IE
INTF	TX1IF	TX0IF	RX1IF	RX0IF

Bit0–Bit1: 接收中断位. MCP2510 有两个接收缓冲寄存器, 缓存器 0 和缓存器 1, 驱动将接收数据读入这两个寄存器再传递进用户层. 在这两个位使能的情况下, 若 RX0IF 置位, 则缓存器 0 已满; 若 RX1IF 置位则缓存器 1 已满.

Bit2–Bit4: 发送中断位. MCP2510 有三个发送缓存器, 缓存器 0, 1 和 2. 在这三个位使能的情况下, 若 TX0IF 置位, 则发送缓存器 0 是空; 以此类推.

Bit5–Bit6: 错误中断位和唤醒总线活动位.

Bit7: 报文错误中断位. 如果报文发送和接收过程中发生错误, 将触发该位中断. 该中断功能在与监听模式联用时被用来加快波特率的确定.

MCP2510 提供五种工作模式: 配置模式, 正常模式, 睡眠模式, 监听模式, 回环模式. 这五种模式通过设置 CANTCRL 寄存器的值来切换, 其中波特率必须在配置模式下才能成功设置; CAN 总线的通信运行在正常模式下; 监听模式仅接受数据帧而不能发送数据, 应该在该模式下进行波特率自适应.

3.2 波特率自适应软件原理

本文所使用的硬件平台将外部中断 4 作为 CAN 专用中断引脚, SPI 总线将差分信号传输到 MCU 的外部中断引脚, 驱动对该中断做出一系列响应从而得到传输数据. 首先对 MCP2510 的 CANINIE 进行设置, 使能中断. 当信号来临时会根据相应中断对中断标志寄存器进行设置, 在中断服务程序中读取 CANINIF 寄存器的值并进行判断, 完成中断响应. 通讯双方波特率相匹配时, 中断服务程序将会设置工作模式为正常模式并接收数据; 波特率设置有误时, 中断服务程序将会设置工作模式为正常模式并进入波特率自适应阶段.

在波特率自适应阶段, 驱动首先对数据进行采样以获得预判波特率. 中断触发方式有边沿触发和电平触发, 边缘触发分为高电平触发和低电平触发. 驱动中可调用 set_irq_type 函数对触发方式进行设置, do_gettimeofday 函数可在高低电平触发时分别获得一个当前时间值. do_gettimeofday 是系统函数, 用户层函数 gettimeofday 实际调用的便是该函数, 它可以实现微秒级的时间获取. 将获得的当前时间与前一次获得的时间相减并不断比较可获得一个最小时间差 T_{min}. 由于 CAN 总线采用反向不归零编码 (NRZ), 该种编码方式采用位填充的方法确保至少每 6 位时间发生一次跳变, 即 T_{min} 可以是实际码元长度的 1–5 倍. 从提高驱动效率的角度出发, 实际出现长时间不跳变的情况较少, 所以本文仅考虑 T_{min} 就是实际码元长度的情况.

获得预判波特率以后即使用该波特率进行测试, 若再次进入报文错误中断, 则依据该波特率所在位置对其周围波特率进行左右轮询直至波特率最终匹配成功. 波特率自适应流程图如图 2.

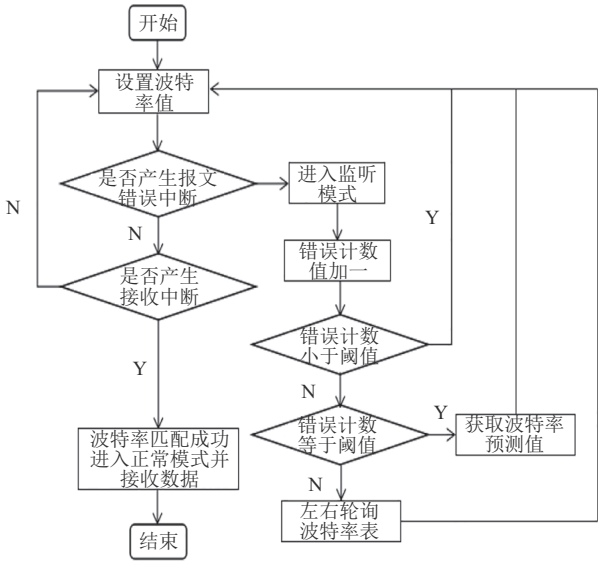

图 2 CAN 波特率自适应流程图

当产生报文错误中断后, 首先获取波特率预测值并对该波特率进行测试. 该预测值往往有一定的误差, 将波特率表中的各值之间的中间值作为判定该值的边界, 可使获得的波特率归束到波特率表中存在的值中. 若再次进入报文错误中断, 则进行波特率轮询. 确定该波特率在波特率表中的位置, 根据错误计数值的奇偶性向左右两个方向进行轮询, 即一回合向左, 下一回合向右. 若一方先达到边界值, 则专注向另一方向轮询直至找到正确值. 轮询部分交由 Linux 底部中断完成. 波特率匹配成功后进入接收中断, 对 CANINIF 寄存器的接收缓存标志位进行问询, 若缓存器 0, 1 有空闲空间, 则读取数据; 若缓存器满, 则唤醒等待队列, 将数据传递到用户层. 在中断服务程序执行完毕后, 需要手动清除中断标志位, 否则中断会一直响应陷入死循环. 清除中断标志位最好使用按位清除的方式, 以防未被响应的中断标志被一并清除.

3.3 用户优先级的设置

CAN 波特率的设置与距离有关, 波特率越高, 能够稳定通信的距离越短. 即在距离一定的情况下, 往往波特率的设置区间也受到了一定的限制. CAN 总线在不同波特率下允许的最大通讯距离是: 6.7 km(10 Kbps)、

1.3 km(50 Kbps)、620 m(100 Kbps)、530 m(125 Kbps)、270 m(250 Kbps)、130 m(500 Kbps)、40 m(1Mbps)[7]. 在 CAN 驱动中, 波特率初值固定地设置为波特率表的第一个值, 这对于波特率自适应来说是一种资源的浪费. 基于此, 本文将用户使用频率的影响添进波特率初始值的设置, 增加了一次碰撞即可获得正确波特率的概率, 提高了波特率匹配效率.

用户数据需要具有能够反复读取、即使驱动关闭也能继续存留数据的功能, 因此仅将其作为变量写入驱动是不够的. 在本方法中用户数据被写入文件, 在驱动打开时读入数据, 接收中断里进行优先级判定, 驱动关闭时保存数据. 驱动层调用读写程序之前需要使用 set_fs() 函数对内核空间进行保护.

用户数据中保存一个二维数组, 该数组放置了波特率表中的各波特率及其所属优先级, 程序每进入一次接受中断, 证明该次波特率匹配成功, 对该波特率进行判断并将其对应的优先级增加 1, 再进行排序后便可获得一个新的波特率优先级表. 该表存放在一个掉电不擦除的目录下, 每次初始化时读取到驱动中的数组里, 使用后在关闭驱动时原路保存.

4 实验结果分析

本设计方案为用户设定了 16 个常见波特率, 满足一般情况下的通讯需求. 通过实验平台串口调试功能检测驱动运行情况, 驱动初始化阶段打印优先级信息, userlist 是波特率枚举值, priority 为对应优先级信息. 用户层运行 CAN 数据收发程序, 进入波特率调试阶段, 驱动首先获得波特率探测值再根据该值进行轮询.

图 3 首先以十六进制形式向 ARM 开发板发送一组随机数据 AA 76 07 54 65 34 53 55, 成功匹配波特率后顺利接收; 然后 ARM 端向 PC 端发送如图 4 所示三组数据.

图 4 接收数据为 ASSIC 码转换而成. 经过测试, 本文所设计 CAN 驱动匹配时间基本维持在 3 秒以内, 而仅使用轮训方式匹配波特率会有首尾波特率轮询时间差距大的情况出现, 如文献[6]中波特率自适应时间从 4 s 到 12 s 时间不等. 且本设计增加了用户选择频率的影响, 当所设置波特率为常用波特率时, 有很大几率会一次命中正确波特率. 在实际实验中, 当用户优先级第一的波特率被选中时, 匹配时间会被缩短到 1 s 以内, 驱动能够在毫秒级时间内完成波特率匹配, 这进一步提高了波特率匹配效率.

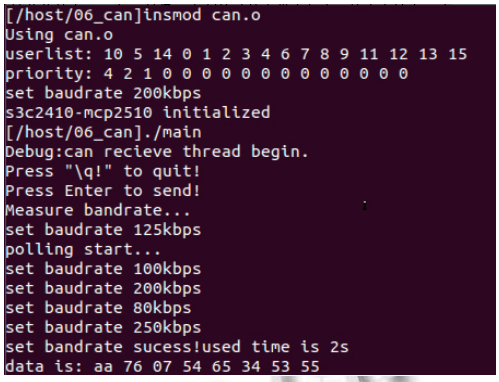

图 3 驱动匹配测试图

图 4 驱动数据收发测试图

5 结语

本文在对 CAN 驱动结构进行分析的基础上设计了一款可以实现 CAN 波特率自适应的驱动. 以 S3C2410 微处理器为主控制器、MCP2510 芯片为 CAN 控制器的嵌入式设备为实验硬件平台. 设计方案融合了两种常用波特率自适应方法并增加了用户使用频率对波特率自适应的影响, 文章对这部分内容进行了详细的解释. 经过实验验证了该方案比原本的方法耗时更短, 该驱动正常工作时数据能正确传输.

参考文献

1 冯军, 王耀南, 刘宏立. Linux 系统下 CAN 总线通信的设计及实现. 微计算机信息, 2008, 24(32): 71–72, 81. [doi: 10.3969/j.issn.1008-0570.2008.32.031]
2 田小刚, 蔡启仲, 邬军伟. 基于嵌入式 Linux 下 CAN 设备驱动程序设计. 微计算机信息, 2009, 25(17): 155–157.
3 黄辉, 范霁月, 张明. CAN 总线接口设计与应用. 轻工科技, 2013, (1): 61–62.
4 陈在平, 许东辉. Linux 下 S3C2440 微控制器的 CAN 驱动设计与实现. 化工自动化及仪表, 2011, 38(8): 985–988.
5 张雪松, 王鸿磊, 徐钊. 嵌入式 Linux2.6 内核的 CAN 驱动设计与实现. 计算机工程与设计, 2010, 31(15): 3396–3398, 3426.
6 曾祥文, 宋树祥, 宾相邦. 嵌入式 Linux 下波特率自适应的 CAN 总线驱动的实现. 测控技术, 2015, 34(8): 104–107.
7 谢云山, 杨安种, 龚建宇, 等. 自适应 CAN 总线波特率转换器设计. 自动化与仪器仪表, 2013, (5): 62–63.