基于 ARM 嵌入式系统的 SPI 驱动程序设计

李 琦,贺 明,董利民,董 健

(北京工业大学 集成电路与系统集成实验室,北京 100124)

摘 要:以微处理器 S3C2440 和嵌入式 Linux 操作系统组成的嵌入式系统作为主要开发平台,根据 SPI 通信原理和 S3C2440 电路接口的特点,设计了一款基于 ARM 嵌入式系统的 SPI 驱动程序。讨论了 SPI 驱动程序的基本开发方法和实现过程,通过编写简单的测试程序进行仿真验证。验证结果表明该驱动程序稳定可靠,可实现嵌入式系统的数据通信。

关键词:S3C2440;嵌入式 Linux;SPI;驱动程序

中图分类号:TN402 **文献标识码**:A **文章编号**:1674-7720(2011)05-0005-04

Design of SPI driver based on ARM embedded system

Li Qi, He Ming, Dong Limin, Dong Jian

(Very Large Scale Integrated Circuits & System Laboratory of Beijing University of Technology, Beijing 100124, China)

Abstract: This paper applied S3C2440 microprocessor and embedded Linux operating system as a development platform, designed an SPI driver based on SPI communication principle and SPI interface in S3C2440. And the paper mainly described the basic development method and implementation process. The correction of SPI driver was verified by writing simple testing codes. It is proved that the driver is stable and reliable, enabling communication in embedded system.

Key words: S3C2440; embedded Linux; SPI; driver

嵌入式系统已被广泛应用于国防电子、数字家庭、工业自动化、汽车电子等多种领域[1]。在嵌入式开发过程中,许多系统通常使用串口驱动来满足通信要求,但在实际应用中,使用 SPI 通信方式会更加高效和快捷[2]。SPI 接口是一种高速、高效的串行接口技术,因而 SPI 设备在数据通信应用中十分方便[3]。本文基于 ARM9 芯片的 S3C2440 和 Linux 操作系统,设计了一种 SPI 驱动程序,该驱动程序功能可靠灵活、易于移植,可应用于多种嵌入式平台,实现 ARM 与设备之间的通信。

1 硬件说明

1.1 S3C2440 开发平台

采用三星公司的 SoC 芯片 S3C2440[4]作为核心处理器,主频为 400 MHz,并与 64 MB SDRAM 和 64 MB NAND Flash 共同组成核心部分。此外,该平台也为用户提供了大量的通信、显示、调试以及 I/O 接口。为满足设计需要,将 Linux2.6.21 版内核移植于该平台上。

1.2 SPI 硬件模块

S3C2440 具有两个 SPI,每个 SPI 具有两个 8 位移位寄存器用于独立地发送和接收数据,并兼容 SPI ver. 2.11 协议,支持 8 位逻辑预分频,系统可用 polling、中断、DMA 三种方式判断 SPI 发送及接收状态。此 SPI 模块共包含以下信号线[5]:

(1)SCK:数据同步时钟信号,由主设备驱动,向从设备输出,使得从设备按照同步时钟的步调来接收或发送数据。

(2)nCS(由用户指定 GPIO):从设备选择信号线(Slave Select,SS)由主设备发出,用来选择激活某个从设备,低电平有效。

(3)MISO(SPIMISO0):主入从出信号线,表示该信号在主设备中作为输入,在从设备中作为输出。

(4)MOSI(SPIMOSI0):主出从入信号线,表示该信号在主设备中作为输出,在从设备中作为输入。

(5)/SS(nSS):多主错误检测。

2 Linux 下的 SPI 设备驱动程序设计

Linux 设备驱动在 Linux 内核中扮演着重要的角色。它可使某些特定硬件响应一个定义良好的内部编程接

口，这些接口完全隐藏了设备工作的细节。用户操作可通过一组标准化的调用来执行，这些调用在形式上完全独立于特定的驱动程序，而将这些调用映射到实际硬件设备的特有操作上，则是驱动程序的任务[6]。本设计的SPI驱动主要定义了初始化、读和写三个操作。其中初始化操作用于驱动程序第一次加载到内核运行时，对一些内核机制及存储器进行初始化。写操作负责将用户数据拷贝至内核缓冲区，控制本地主SPI发送数据至从SPI寄存器中。读操作将按照用户要求读取的字节数，连续读取本地主SPI中接收到的数据，并将其拷贝至用户空间。驱动程序将采用中断的方式通知系统SPI数据是否发送完毕，即当SPI硬件模块每发送完毕一个数据，都会通过中断线向系统发起中断，系统响应中断后，驱动程序将调用中断处理例程。

2.1 SPI初始化

(1)申请中断。此驱动设计通过中断判断数据是否发送完毕，所以需要申请SPI0相关的中断，并注册相应的中断处理函数。此驱动程序的中断处理函数声明如下：

```
static irqreturn_t s3c2440_isr_spi (int irq, void*dev_id, struct pt_regs*reg)
```

利用request_irq向内核申请中断号并注册中断处理函数：

```
request_irq (IRQ_SPI0, s3c2440_isr_spi, SA_INTERRUPT, DEVICE_NAME, s3c2440_isr_spi);
```

(2)虚拟地址映射。驱动程序可以直接通过访问内核中的虚拟地址来访问设备物理地址所对应的寄存器，对其进行操作。SPI设备的地址映射过程如下：

```
request_mem_region(S3C2440_PA_SPI,0x30,"s3c2440-spi");
base_addr = ioremap(S3C2440_PA_SPI, 0x30);
```

其中S3C2440_PA_SPI为SPI的物理地址（在/asm-arch/arch-s3c2440/map.h中定义），从S3C2440_PA_SPI开始分配0x30大小的内存区域，此后将其移至内核空间。

(3)相关寄存器的设置。通过配置SPI功能寄存器设置SPI工作模式。以ioremap返回的虚拟地址为基址，通过增加不同偏移量访问相应寄存器。本次设计将本地SPI设为主设备，开启SCK信号使能，设定CPOL和CPHA均为0，SPI工作在普通模式下。设置波特率预分频寄存器(SPPRE)中的分频比为8。具体设计如下：

```
__raw_writel((S3C2440_SPCON_SMOD_INT|S3C2440_SPCON_ENSCK|
S3C2440_SPCON_MSTR), s3c2440_SPCON);
DPRINTK(DEVICE_NAME"SPCON initialize\n");
__raw_writel ((S3C2440_SPPIN_ENMUL | S3C2440_SPPIN_KEEP),
s3c2440_SPPIN);
DPRINTK(DEVICE_NAME"SPPIN initialize\n");
__raw_writel(0x07, s3c2440_SPPRE);
DPRINTK(DEVICE_NAME"SPPRE initialize\n");
```

(4)初始化发送和接收数据缓冲区。数据缓冲区使用环形缓冲区结构，通过头尾指针的循环移动，实现对缓冲区的动态管理。其定义如下：

```
typedef struct
{
    spi_buf buf[MAX_SPI_BUF];
    unsigned int head, tail;
    wait_queue_head_t wq;
} SPI_BUF; static SPI_BUF spi_Tx_buf;static
    SPI_BUF spi_Rec_buf;
```

其中spi_buf表示char型，MAX_SPI_BUF为缓冲区大小，设为1 024 B。head、tail分别表示头尾数组下标，wq为等待队列头。此结构依靠以下宏进行管理：

```
#define SPI_Tx_BUF_HEAD(spi_Tx_buf.buf[spi_Tx_buf.head])
#define SPI_Tx_BUF_TAIL(spi_Tx_buf.buf[spi_Tx_buf.tail])
#define INCBUF(x, mod)((++(x))&((mod)-1))
```

前两个宏用于引用缓冲区中的元素，最后一个宏用于对头尾下标进行前移，并保证头尾下标数值可循环变化，不发生溢出。

在初始化时，分别对接收和发送缓冲区的头尾指针进行清零操作，具体如下：

```
spi_Tx_buf.head =spi_Tx_buf.tail =0; spi_Rec_buf.head = spi_Rec_buf.tail = 0;
```

(5)内核机制相关的数据结构初始化。本设计所使用的内核机制包括了中断上下半部的操作和睡眠等待机制，因此需要对发送、接收等待队列以及tasklet结构进行初始化，并注册tasklet处理函数。初始化过程如下：

```
init_waitqueue_head(&(spi_Tx_buf.wq));
init_waitqueue_head(&(spi_Rec_buf.wq));
tasklet_init(&spi_tasklet, spi_tasklet_handler, data);
```

(6)初始化相应端口。根据S3C2440外部管脚配置，将与SPI功能引脚复用的GPIO设定为SPI相应功能。具体操作如下：

```
s3c2440_gpio_cfgpin
        (S3C2440_GPE11, S3C2440_GPE11_SPIMISO0);
s3c2440_gpio_cfgpin
        (S3C2440_GPE12, S3C2440_GPE12_SPIMOSI0);
s3c2440_gpio_cfgpin
        (S3C2440_GPE13, S3C2440_GPE13_SPICLK0);
s3c2440_gpio_cfgpin
        (S3C2440_GPG2, S3C2440_GPG2_INP);//设置 nSS
s3c2440_gpio_cfgpin (S3C2440_GPB10,
        S3C2440_GPB10_OUTP); //设置片选信号
s3c2440_gpio_setpin(S3C2440_GPB10, 1);
```

2.2 SPI写操作

写操作主要是将上层应用部分的用户空间中的数据拷贝到内核空间中的环形缓冲区中，此后将缓冲区的数据送到SPI发送寄存器中，在SPI发送完一个数据后，系统产生中断，中断例程中的下半部将调用tasklet判断缓冲区状态。若缓冲区中有相应的空间，可以将下一数

据填入 SPI 发送寄存器中，直至将缓冲区数据全部发送完毕。

本设计的写操作实现了环形缓冲区的动态管理，即在缓冲区删除数据、尾指针前移的情况下，允许向缓冲区添加数据，头指针前移。此设计可以使用户空间任务与内核空间的数据发送同时进行，提高了用户空间任务执行效率，并且当利用 copy_from_user 函数将数据从用户空间拷贝至内核空间时，数据发送仍在进行，即数据从用户空间至内核空间拷贝过程与数据发送过程并发，提高了驱动程序效率。

为了实现环形缓冲区动态管理，定义了copy_to_Tx_buf_init 和 copy_to_Tx_buf 两个函数完成数据向缓冲区的复制操作。

(1)copy_to_Tx_buf_init 函数。本函数主要用于两种情况：

①如果缓冲区为空，当有一组数据到来且此数据的大小小于缓冲区的空间大小时，直接将此数据放到缓冲区中。

②如果发送数据的大小大于剩余缓冲区的空间，则只复制缓冲区大小的数据到缓冲区。

缓冲区满，该进程进行睡眠操作，直到缓冲区所有数据发送完毕，缓冲区再次为空，当前进程被唤醒，将此组用户数据的未发送部分复制到缓冲区，继续发送。

(2)copy_to_Tx_buf 函数。此函数主要用于缓冲区正在发送且未发送完毕的情况，将新一组用户数据 copy 至缓冲区。首先计算缓冲区剩余空间，若剩余空间大于本组用户数据大小，则直接将用户数据全部 copy 至缓冲区；若剩余空间小于本组数据大小，则 copy 与剩余空间大小相同的用户数据至缓冲区。

写操作的具体流程如图 1 所示，首先用户数据从空间态转换到内核态，并设置相应的接收标志位。此后判断数据大小。若数据大于缓冲区空间，数据发生溢出，写操作结束；若没有溢出，为了保证进程间的数据，使得该进程获得自旋锁，此时判断缓冲区是否为空。根据上面两个函数的介绍，在不同情况下分别调用不同的函数，在数据写入环形缓冲区后，将数据发送到 SPI 的发送寄存器。当 SPI 发送寄存器发送数据时，环形缓冲区依旧接收数据，如果此时缓冲区为满，则释放自旋锁，并设置进程等待标志位(wait_Tx_done)，将此进程休眠，直到发送寄存器中的数据发送完毕，再唤醒进程，判断数据是否全部发送完毕。若仍有数据等待发送，则调用 copy_to_Tx_buf_int；若数据已全部发送完毕，则写操作结束。若缓冲区不为满，则判断数据是否发送完毕。数据全部发送完毕，发送操作结束。

2.3 SPI 读操作

读操作是连续读取主 SPI 发送到从 SPI 的接收缓冲区中的数据，并将其传送给用户空间。具体流程如图 2 所示。首先判断操作标志位 spi_Rec_en，若此位为 0，说明此时驱动正处于发送状态，则将发送进程等待标志位(wait_Tx_done)置 1，读进程进入休眠状态即放入等待队列中，等待中断处理函数中相关发送程序唤醒。若操作标志位不为 1，读进程首先获得自旋锁，判断数据大小。若数据大小不为 0 且不超过缓冲区大小，则按照 S3C2440 接收数据的要求，向 SPI 发送寄存器写入第一个 dummy 数据 (0xff)。此后，将接收进程等待标志位(wait_Rec_done)置 1，释放自旋锁，并将此进程加入等待队列进行休眠，直到用户要求的所有数据已发送至接收缓冲区后，由中断处理函数唤醒该进程，最后将接收区中的数据放到临时接收缓存中，以便于其他操作读取。

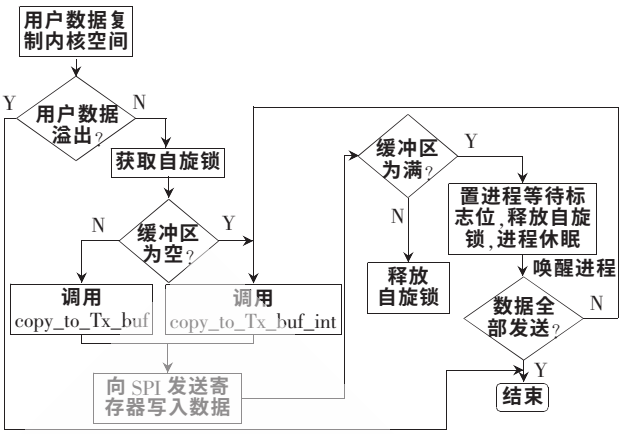

图 1 写操作流程图

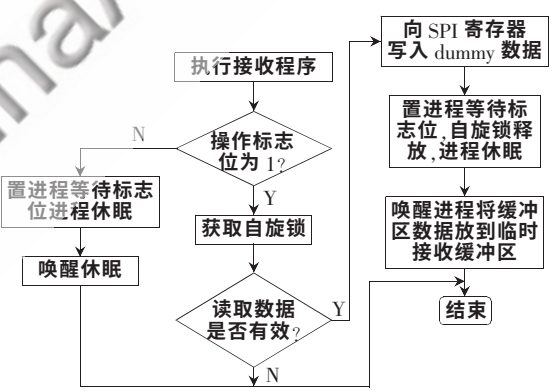

图 2 读操作流程图

3 SPI 驱动程序测试

SPI 驱动程序主要通过调用写操作，使 SPI 连续发送数据 0x55，此后再调用 SPI 读操作，将 MISO 上的串行数据读入用户缓冲区，并与实际数据进行比较。图 3 为示波器测试 MOSI 引脚波形。图中波形 1 为 SCK 信号，ARM 系统时钟为 40 MHz，SPI 的 SCK 信号为系统时钟的 256 分频，约为 156 kHz；波形 2 为 MOSI 信号，SPI 从低位向高位串行移位。通过波形可以看出，SPI 驱动能够准确地完成读写操作，验证了其正确性。

本文以 S3C2440 为硬件开发平台，采用嵌入式 Linux 操作系统驱动设计方法，设计了一款通用的 SPI 驱动程

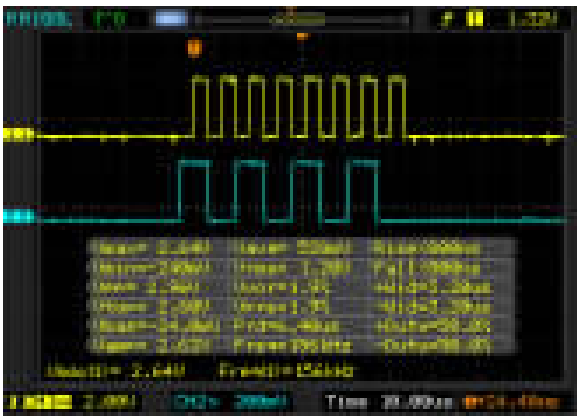

图 3 SPI 输出波形图

序，并通过编写简单测试程序，观察示波器输出波形验证。该驱动程序可以使微处理器和外设之间进行稳定可靠的数据传输，具有功能灵活、可移植性强、可靠性高等特点，有一定的使用价值和借鉴意义。

参考文献

[1] 孙天泽，袁文菊. 嵌入式设计及 Linux 驱动开发指南—基于 ARM9 处理器（第 3 版）[M]. 北京：电子工业出版社，2009.

[2] 张晓雷，陈晓宁，郭剑. 嵌入式 Linux 下基于 SPI 总线的网络设备驱动设计与实现 [J]. 计算机工程与设计，2008，29(23).

[3] 沃尔瓦诺. 嵌入式微机算机系统：实时接口技术[M]. 李森译. 北京：机械工业出版社，2008.

[4] S3C2440A 32-Bit CMOS Microcontroller User's Manual, Revision 2. Samsung Electronics. 2008.

[5] 徐英慧. ARM9 嵌入式系统设计—基于 S3C2440 与 Linux [M]. 北京：北京航空航天大学出版社，2008.

[6] CORBET J, RUBINI A, KROAH-HARTMAN G. LINUX 设备驱动程序(第三版)[M]. 魏永明，耿岳，钟书毅，译. 北京：中国电力出版社，2009.

(收稿日期：2010-11-02)

作者简介：

李琦，男，1986 年生，硕士研究生，主要研究方向：集成电路与系统集成。