基于华邦 W90P710 的嵌入式 Linux 串口驱动的实现方法

肖铁航
(深圳市拓邦自动化科技股份有限公司,广东 深圳 518108)

摘 要:基于华邦 W90P710 处理器的 Linux 内核应用,详细介绍了 Linux 串口驱动的实现方法。同时对 Linux 文件系统操作入口函数及内核的编译做了详细的说明。

关键词:ARM;Linux;UART;文件系统;串口驱动程序
中图分类号:TP311.567 **文献标识码**:A **文章编号**:1674-7720(2011)24-0065-04

Implementation of embedded Linux serial port driver based on Winbond W90P710

Xiao Tiehang
(Shenzhen Topband Automation Technology ., Ltd., Shenzhen 518108, China)

Abstract: This paper introduces the implementation of embedded Linux serial port driver based on the implementation of Linux in Winbond W90P710 processor in great detail. Entry function of Linux file system operation and kernel compilation are also illustrated with detailed description.

Key words: ARM; Linux; UART; file system; serial port driver

嵌入式 Linux 是一种很受欢迎的操作系统,具有开放源码、不存在黑箱技术、内核小、功能强大、运行稳定、效率高、易于定制裁减等特点[1],广泛应用于工控产品。很多工控产品需要和外部设备进行信息交换,而串口通信是最简单快捷的实现方法。在不同的工控产品中,由于对所选用的串口元件或者串口通信的数据格式、波特率等有不同的需求,需要对串口驱动进行开发。华邦 W90P710 采用 ARM 的 ARM7TDMI 微处理器核心,采用 μCLinux-2.4.20 内核,支持 4 组通用异步接收发送口(UART),下面基于华邦 W90P710 的串口驱动详细分析串口驱动的实现方法,实现嵌入式设备通过串口对外通信。

1 华邦 W90P710 UART 介绍

华邦 W90P710 支持 4 组 UART,串口的控制主要通过以下寄存器实现[2]:

(1)行寄存器(UART_LCR):设置数据位长度、奇偶校验、停止位数。

(2)波特率除数寄存器(UART_DLL、UART_DLM):波特率发生器的公式为:BaudOut=crystal clock/16×[Divisor +2],Divisor 为当前波特率。

(3)Modem 控制寄存器(UART_MCR):控制 RTS、CTS 等信号。

(4)FIFO 控制寄存器(UART_FCR):设置 FIFO 的长度,复位 FIFO 等控制。

(5)接收超时寄存器(UART_TOR):收到首个字节后接收器启动本超时,之后每收到一个字节后都会重置该值,在此超时时间内不再收到数据时,接收器会产生一个接收中断。

(6)中断控制器(UART_IER):设置接收、发送、行中断等。

在使用 RXDn、TXDn 前必须对 GPIO 进行配置,使能 RXDn、TXDn,串口才可正常运行。GPIO 配置对应表如表 1 所示。

2 Linux 系统驱动介绍

设备驱动程序是操作系统内核和机器硬件之间的接口。设备驱动程序为应用程序屏蔽了硬件的细节,这样在应用程序看来,硬件设备只是一个设备文件,应用程序可以像操作普通文件一样对硬件设备进行操作。同

表 1 GPIO 配置表

管脚定义	GPIO 名称	GPIO 寄存器	GPIO 地址	设置值
TXD0	GPIO5	GPIO_CFG5	0XFFF8_3050	0X00000001
RXD0	GPIO6	GPIO_CFG5	0XFFF8_3050	0X00000004
TXD1	GPIO7	GPIO_CFG5	0XFFF8_3050	0X00000010
RXD1	GPIO8	GPIO_CFG5	0XFFF8_3050	0X00000040
TXD2	GPIO9	GPIO_CFG5	0XFFF8_3050	0X00000100
RXD2	GPIO10	GPIO_CFG5	0XFFF8_3050	0X00000400
TXD3	GPIO3	GPIO_CFG0	0XFFF8_3000	0X000000c0
RXD3	GPIO4	GPIO_CFG0	0XFFF8_3000	0X00000300

时，设备驱动程序是内核的一部分[3]。图 1 所示为设备驱动程序接口流程图。

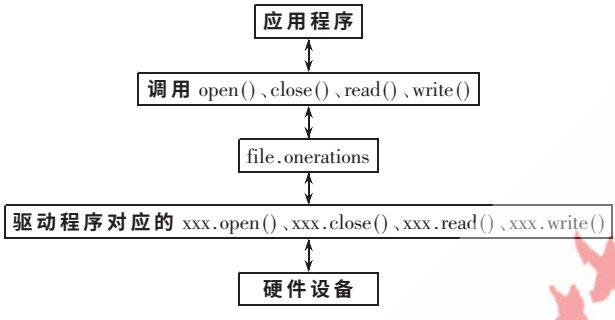

图 1 驱动程序接口流程图

Linux 系统的设备分为字符设备、块设备和网络设备三种。字符设备是指存取时没有缓存的设备，只能顺序读写。典型的字符设备包括鼠标、键盘、串行口等；块设备一般都有缓存来支持，并且块设备必须能够支持随机存取。块设备主要包括硬盘设备、CD-ROM 等；网络设备在 Linux 系统中用做专门的处理，Linux 的网络系统主要是基于 BSD Unix 的 socket 机制[4]。

3 串口驱动程序详细介绍

一般来说，Linux 的设备驱动程序包括驱动程序的注册和注销、设备的打开和释放、设备的读写操作、设备的控制操作、设备的中断和轮询处理等功能。下面就这些功能对串口驱动进行详细说明。

(1)串口设备的数据结构包括串口参数接收发送缓冲区等。串口参数包括波特率、数据位、数据起始位、奇偶校验、串口类型、发送缓冲区、接收缓冲区等，每个串口对应一个如下的数据结构：

```
typedef struct{
    int   bps;
    int   databits;
    int   stopbits;
    int   parity;
    int   siotype;                          //串口参数
    int   openflag;
    int   recvTrigTimeout;
    SIO_D_SEND_BUFFER    *pSendBuf;    //发送缓冲区
    SIO_D_RECV_BUFFER    *pRecvBuf;    //接收缓冲区
    struct fasync_struct *fasync_queue;
    wait_queue_head_t    read_wait;
}serial_dev;
static serial_dev serial_device;
```

(2)文件系统操作入口函数对应文件操作函数 read()、write()、ioctl()、open()、close()。

```
struct file_operations serial_fops = {
    owner:      THIS_MODULE,
    poll:       serial_poll,
    read:       serial_read,
    write:      serial_write,
    ioctl:      serial_ioctl,
    open:       serial_open,
    release:    serial_release,
};
```

(3)驱动程序注册和注销。驱动程序在应用前，需要在模块初始化时将设备注册到系统设备表中；不再使用时，将设备从系统中卸除。注册包括初始化定时器、初始化串口数据结构 serial_device 和字符设备注册。注销时直接调用设备注销函数[5]。

```
int __init topbandserial1_init(void)
{
    init_timer(&timer);                    //初始化定时器结构
    memset(&serial_device, 0, sizeof(serial_device));
    result=register_chrdev(SERIAL1_MAJOR, "serial1",
                                        &serial_fops);
    …
}
```

(4)串口设备打开包括分配串口的接收发送缓冲区及中断注册[5]。

```
static int serial_open(struct inode *inode, struct file *filp)
{
  dev->pRecvBuf = kmalloc(sizeof(SIO_D_RECV_BUFFER),
                                        GFP_KERNEL);
  request_irq(INT_UART1,serial_interrupt,SA_SHIRQ,
                    "TopbandSerial1",&serial_device);
  …
}
```

(5)串口设备释放包括释放内存空间、注销中断和删除定时器[5]。

```
static int serial_release(struct inode *inode, struct file *flip)
{
    serial_dev *dev = flip->private_data;  //释放内存空间
    kfree(dev->fasync_queue);
    CSR_WRITE(COM_IER_1, 0x00);        /* 中断禁止 */
    free_irq(INT_UART1, dev);                //注销中断
                        del_timer(&timer);//删除定时器
    MOD_DEC_USE_COUNT;
    dev->openflag = 0;
    …
}
```

(6)串口读数据是指返回接收缓冲区中已收到的数据。读取数据有两种方式,阻塞方式和非阻塞方式。阻塞方式[6]中用户程序执行读操作时如果没有数据可读,即让read()操作等待直到数据可读;非阻塞方式中当用户执行读操作时,不论串口是否接收到数据,设备驱动xxx_read()函数会立刻返回,read()函数系统调用也随即返回。

```
static int serial_read(struct file *filp, char *buf, size_t
                                        count, loff_t *f_pos)
{
    if(filp->f_flags & O_NONBLOCK)   /非阻塞方式读取
        retsts = serial_nonblock_read(dev,buf,count);
    else                                    /* 阻塞方式读取 */
        retsts = serial_block_read(dev,buf,count);
    …
}
```

(7)串口写数据包括把数据存放在发送缓冲区、启动硬件发送及发送中断。当发送第一个字节后,硬件会产生发送中断,剩下的数据将在中断处理程序中发送。

```
static int serial_write(struct file *filp, const char *buf,
                                    size_t count, loff_t *f_pos)
{
copy_from_user(&pSendBuf->frameData[pSendBuf->
                                    bufWritex].data[0],buf, count);
    CSR_WRITE(CMBOARD_GPIO_DATAOUT1,status1);
enable_tx_interrupt_1();
    …
}
```

(8)串口控制包括设置串口波特率、奇偶校、停止位等,还可以定义其他特殊的控制。应用程序通过ioctl()调用把串口的参数传递给驱动程序,驱动程序再通过对硬件串口控制寄存器进行设置,来满足应用层用户要求。

```
static int serial_ioctl(struct inode *inode, struct file *filp,
                        unsigned int cmd, unsigned long arg)
{
    switch(cmd){
            case SERIAL_IOC_BPS:
                …
                break;
            case SERIAL_IOC_SENDBUF:
                …
                break;
        }
}
```

(9)中断处理包括对接收中断、发送中断、异常中断的处理。读取中断寄存器的状态,根据不同的中断类型分别处理。当收到数据时,硬件会产生接收中断,驱动程序把串口的数据读取出来,放在接收缓冲区中,直到所有数据读取完成;当发送数据时,硬件会产生发送中断,驱动程序把发送缓冲区的数据发送出去,直到所有数据发送完成;当串口接收或发送发生异常时,会产生异常中断,驱动程序根据情况把串口重新初始化,以便串口恢复正常。

```
static void serial_interrupt(int irq, void * dev_id,
                                    struct pt_regs *regs)
{
    status = CSR_READ(COM_IIR_1);
    while(status & UART_IIR_STATUS_NO) == 0)
    {
        switch(status)
        {
            case UART_IIR_STATUS_RDA:
            case UART_IIR_STATUS_TOUT:
                receive_chars(dev,status);
                break;
            case UART_IIR_THRE:
                transmit_chars(dev);
                break;
        }
        status = CSR_READ(COM_IIR_1);
    }
}
```

(10)定时器处理。中断接收程序只负责把数据读取到缓冲区,并没有指示缓冲区的数据可被用户使用,这时需要在超时程序中把可用标志置上,当用户调用read()函数时就可把接收缓冲区的数据返回。

```
static void serial_timer(unsigned long dummy)
{
    …
    serial_device.pRecvBuf->frameData
        [serial_device.pRecvBuf->bufWritex].finished = 1;
    mod_timer(&timer,jiffies+2);      /* 20 ms 进一次 */
}
```

通过以上几个函数的处理,实现了串口的驱动。

4 驱动程序编译进 Linux 内核

以下以UART1为例,介绍驱动程序编译进Linux内核的过程,步骤如下:

(1)添加主次设备号。

主次设备号用来标识一个具体设备。主设备号用于标识设备类型,每种类型的设备需要一个对应的设备驱动程序。一个主设备可以有多个具体的设备与之对应。次设备号用于区分使用同种驱动程序的同类设备中多个不同的设备实例[7]。

在W90P710-μClinux/μClinux-dist\linux-2.4.x/include/linux目录下的major.h中定义主设备号,添加如下代码:

```
#define SERIAL1_MAJOR   230
```

在W90P710-μClinux/μClinux-dist/vendors/Winbond/W90P710目录下的makefile中建立设备主次设备号(主设

备号为 230，次设备号为 1)，添加如下代码：

```
serial1,c,230,1 \
```

(2)在 W90P710－μClinux/μClinux－dist/linux－2.4.x/drivers/char 目录下的 makefile 中添加如下代码：

```
obj-$(CONFIG_TOPBAND_SERIAL1)+=w90p710_serial_1.o
```

(3)在 W90P710－μClinux/μClinux－dist/linux－2.4.x/drivers/char 目录下的 config.in 字符设备段中添加如下代码：

```
#if [ "$CONFIG_TOPBAND_SERIAL1" = "y" ]; then
    bool 'Topband serial1 support' CONFIG_TOPBAND_
                                   SERIAL1
#fi
```

(4)在 W90P710－μClinux/μClinux－dist 目录下运行 make menuconfig，在 menuconfig 的字符设备选项中可以看见刚刚添加的“CONFIG_TOPBAND_SERIAL1”选项，选上该项。使用 make dep、make clean、make 三个命令编译 Linux 内核，生成内核文件 linux.bin[8]。

(5)在 W90P710－μClinux/romdisk/dev 目录下创建设备文件，输入命令：

```
mknod serial1 c 230 1
```

生成设备文件“serial1”，应用程序通过使用“/dev/serial1”这个设备文件名就可对串口进行操作。

最后编写简单的串口测试程序，编译生成镜像文件；再把镜像文件 romfs.img 和内核文件 linux.bin 下载到开发板，把开发板的串口和 PC 机相连，PC 机端使用串口调试工具发送测试数据，开发板能正确收发数据。

本文按驱动程序的功能详细介绍了 W90P710 微处理器实现串口驱动的方法，串口驱动程序是很典型的字符设备驱动程序，其他字符设备驱动和串口的实现方法是相同的，这对开发其他字符设备驱动程序有一定的借鉴作用。

参考文献

[1] 李岩，荣盘祥.基于 S3C44BOX 嵌入式μClinux 系统原理及应用[M].北京：清华大学出版社，2005.

[2] W90P710CD/W90P710CDG16/32－bit ARM microcontroller Product Data Sheet[Z].Winbond Electronics Corporation，2006：330－350.

[3] 刘天时，强新建，王瑞，等.ARM7 嵌入式开发基础实验[M].北京：北京航空航天大学出版社，2007.

[4] 郑灵祥.嵌入式接口技术与 Linux 驱动开发[M].北京：北京航空航天大学出版社，2010.

[5] W90P710 system library user's manual[Z].Winbond Electronics Corporation，2006：9－11.

[6] 崔更申，孙安青.ARM 嵌入式系统开发与实践[M].北京：中国电力出版社，2008.

[7] 宋宝华.Linux 设备驱动开发详解[M].北京：人民邮电出版社，2008.

[8] W90P710 μClinux user's manual[Z].Winbond Electronics Corporation，2005：10－13.

(收稿日期：2011－09－05)

作者简介：

肖铁航，女，1976 年生，工程师，主要研究方向：自动化设备的通信软件及应用软件开发。