Application Note

嵌入式 Linux SPI 总线应用编程

基于 M3352 核心板

AN15070501 V1.00 Date: 2015/7/5

产品应用笔记

类别	内容
关键词	M3352 核心板、AM3352、SPI 总线、应用编程
摘　要	本文主要介绍 M3352 的 SPI 总线应用编程

修订历史

版本	日期	原因
V0.09	2015/7/3	创建文档
V1.00	2015/7/5	文档发布

目录

1. 适用范围

本文主要介绍基于 AM3352 处理器的 SPI 总线应用编程，适用于 M3352 核心板，其使用原理也适用于基于 AM3352 处理器的工控核心板以及其他基于 AM3352 系列处理器的产品。

2. 嵌入式 Linux SPI 总线应用编程

2.1 概述

Linux 下用户空间和 SPI 设备通信有两种方式。第一种就是为 SPI 设备编写专门的内核空间驱动，由驱动负责处理所有的 SPI 协议，使用这种方式用户空间不需要关心具体的硬件知识，大大降低了用户空间程序的复杂度，但是用户空间程序的功能受驱动的限制，不够灵活；还有一种方式就是使用户空间的进程直接访问 SPI 设备，使得内核的编码变得简单，驱动的功能转移到了用户空间，用户空间的程序可以比较灵活的实现各种功能，但要求用户空间的开发者对芯片的工作方式有全面的了解。

本章不介绍如何编写 SPI 内核驱动，仅仅介绍第二种方法，也就是直接访问 SPI 设备来通信。这种方法访问 SPI 设备的流程如图 2.1 所示。

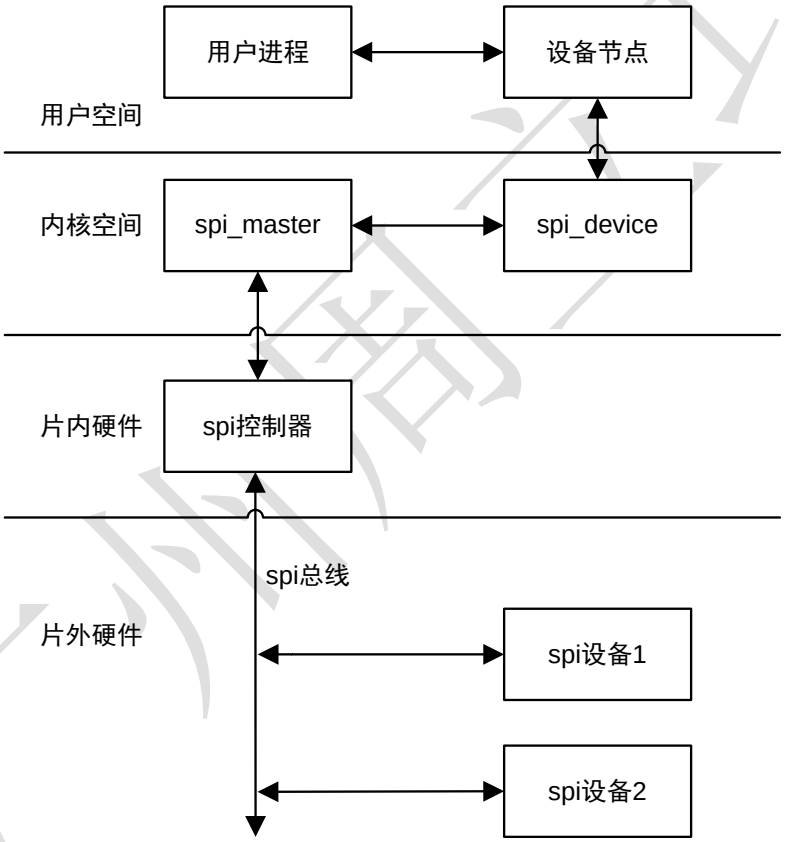

图 2.1 用户空间直接访问 SPI 设备

2.2 重要的数据结构

1. spi_device

虽然用户空间不需要直接用到 spi_device 结构体，但是这个结构体和用户空间的程序有密切的关系，理解它的成员有助于理解 SPI 设备节点的 IOCTL 命令，所以首先来介绍它。

在内核中，每个 spi_device 代表一个物理的 SPI 设备。它的成员如程序清单 2.1 所示。

程序清单 2.1 spi_device

```
struct spi_device {
```

```
    struct device       dev;
    struct spi_master *master;
    u32                 max_speed_hz;           /* 通信时钟最大频率                    */
    u8                  chip_select;            /* 片选号                              */
    u8                  mode;                   /* SPI 设备的模式，下面的宏是它各 bit 的含义 */
#define     SPI_CPHA 0x01                       /* 采样的时钟相位                      */
#define     SPI_CPOL 0x02                       /* 时钟信号起始相位：高或者是低电平    */
#define     SPI_MODE_0    (0|0)
#define     SPI_MODE_1    (0|SPI_CPHA)
#define     SPI_MODE_2    (SPI_CPOL|0)
#define     SPI_MODE_3    (SPI_CPOL|SPI_CPHA)
#define     SPI_CS_HIGH   0x04                  /* 为 1 时片选的有效信号是高电平       */
#define     SPI_LSB_FIRST       0x08            /* 发送时低比特在前                    */
#define     SPI_3WIRE     0x10                  /* 输入输出信号使用同一根信号线        */
#define     SPI_LOOP 0x20                       /* 回环模式                            */
    u8                  bits_per_word;          /* 每个通信字的字长（比特数）          */
    int                 irq;                    /* 使用到的中断                        */
    void                *controller_state;
    void                *controller_data;
    char                modalias[32];           /* 设备驱动的名字                      */
};
```

由于一个 SPI 总线上可以有多个 SPI 设备，因此需要片选号来区分它们，SPI 控制器根据片选号来选择不同的片选线，从而实现每次只同一个设备通信。

spi_device 的 mode 成员有两个比特位含义很重要。SPI_CPHA 选择对数据线采样的时机，0 选择每个时钟周期的第一个沿跳变时采样数据，1 选择第二个时钟沿采样数据；SPI_CPOL 选择每个时钟周期开始的极性，0 表示时钟以低电平开始，1 选择高电平开始。这两个比特有四种组合，对应 SPI_MODE_0～SPI_MODE_3。

另一个比较重要的成员是 bits_per_word。这个成员指定每次读写的字长，单位是比特。虽然大部分 SPI 接口的字长是 8 或者 16，仍然会有一些特殊的例子。需要说明的是，如果这个成员为零的话，默认使用 8 作为字长。

最后一个成员并不是设备的名字，而是需要绑定的驱动的名字。

2. spi_ioc_transfer

在用户使用设备节点的 IOCTL 命令传输数据的时候，需要用到 spi_ioc_transfer 结构体，它的成员如程序清单 2.2 所示。

程序清单 2.2 spi_ioc_transfer

```
struct spi_ioc_transfer {
    __u64           tx_buf;                     /* 写数据缓冲                          */
```

```
    __u64       rx_buf;                 /* 读数据缓冲                          */

    __u32       len;                    /* 缓冲的长度                          */
    __u32       speed_hz;               /* 通信的时钟频率                      */

    __u16       delay_usecs;            /* 两个 spi_ioc_transfer 之间的延时    */
    __u8        bits_per_word;          /* 字长（比特数）                      */
    __u8        cs_change;              /* 是否改变片选                        */
    __u32       pad;
};
```

每个 spi_ioc_transfer 都可以包含读和写的请求，其中读和写的长度必须相等。所以成员 len 不是 tx_buf 和 rx_buf 缓冲的长度之和，而是它们各自的长度。SPI 控制器驱动会先将 tx_buf 写到 SPI 总线上，然后再读取 len 长度的内容到 rx_buf。如果只想进行一个方向的传输，把另一个方向的缓冲置为 0 就可以了。

speed_hz 和 bits_per_word 这两个成员可以为每次通信配置不同的通信速率（必须小于 spi_device 的 max_speed_hz）和字长，如果它们为 0 的话就会使用 spi_device 中的配置。

delay_usecs 可以指定两个 spi_ioc_transfer 之间的延时，单位是微妙，一般不用定义。

cs_change 指定这个 cs_change 结束之后是否需要改变片选线。一般针对同一设备的连续的几个 spi_ioc_transfer，只有最后一个需要将这个成员置位。这样省去了来回改变片选线的时间，有助于提高通信速率。

2.3 获得与 SPI 设备通信的设备节点

首先，要让 linux3.2 内核支持 SPI 设备，进入内核的 menuconfig 对 linux3.2 内核进行配置，依次进入 Device Drivers→SPI supprot 后选择合适的配置项目，如图 2.2 所示。

```
-*- SPI support
        *** SPI Master Controller Drivers ***
< >     Altera SPI Controller
< >     Utilities for Bitbanging SPI masters
< >     GPIO-based bitbanging SPI Master
< >     OpenCores tiny SPI
<*>     McSPI driver for OMAP
< >     Xilinx SPI controller common module
< >     DesignWare SPI controller core support
        *** SPI Protocol Masters ***
<*>     User mode SPI device driver support
< >     Infineon TLE62X0 (for power switching)
```

图 2.2 SPI 内核配置

重新编译好内核后，下载到开发板并从新的内核启动开发板。

为了在用户空间获得和 SPI 设备直接通信的设备节点，必须有两个条件要满足：首先要有 SPI 控制器驱动，其次是要在内核初始化的时候注册一个 spi_board_info，它的 modalias

成员必须为“spidev”。有了这两个条件，就可以和 SPI 设备进行通信了。控制器的驱动一般由芯片厂家提供，开发者只需提供第二个条件。

spi_board_info 的定义如程序清单 2.3 所示。

程序清单 2.3 struct spi_board_info

```
struct spi_board_info {
    char        modalias[32];                   /* 要绑定的驱动的名字                */
    const void  *platform_data;
    void        *controller_data;
    int         irq;

    u32         max_speed_hz;                   /* 通信时钟最大速率                  */

    u16         bus_num;                        /* 总线编号                          */
    u16         chip_select;                    /* 片选号                            */

    u8          mode;                           /* 和 spi_device 中的 mode 成员类似  */
};
```

要了解这个结构体各个成员的意义请参考程序清单 2.1。

首先要初始化好对应端口的功能，如程序清单 2.4 所示。

程序清单 2.4 SPI 引脚初始化

```
static struct pinmux_config spi0_pin_mux[] = {
    {"spi0_sclk.spi0_sclk",OMAP_MODE0 | AM33XX_PULL_ENBL | AM33XX_INPUT_EN},
    {"spi0_d0.spi0_d0", OMAP_MODE0 | AM33XX_PULL_ENBL | AM33XX_INPUT_EN},
    {"spi0_d1.spi0_d1", OMAP_MODE0 | AM33XX_PULL_ENBL | AM33XX_INPUT_EN},
    {"spi0_cs0.spi0_cs0", OMAP_MODE0 | AM33XX_PULL_ENBL | AM33XX_INPUT_EN},
    { NULL ,0}
};
```

定义并注册 struct spi_board_info 的位置一般是内核的 arch/xxx/mach-xxxx/board-xxxx.c，比如 3352 的内核，这个文件是 arch/arm/mach-omap2/board-m3352.c。定义并注册 struct spi_board_info 的代码如程序清单 2.5 所示。

程序清单 2.5 定义并注册 spi_board_info

```
static struct spi_board_info iam335x_spi0_nfo[] =
    [0] = {
            .modalias = "spidev",
            .max_speed_hz = 5000000,
```

```
                .bus_num = 1,
                .chip_select = 0,
        },
};
static void spi0_init(int evm_id, int profile)
{
        setup_pin_mux(spi0_pin_mux);
        return spi_register_board_info(am335x_spi0_boardinfo,ARRAY_SIZE(am335x_spi0_info));
}
```

由于 3352 内核代码在 arch/arm/mach-omap2/board-m3352.c，程序清单 2.5 注册了一个挂在 0 号（注意不是 1 号）SPI 总线上的设备信息，它的片选号为 0。

最后在 zy_m3352_dev_cfg 数组中把 spi0 的初始化函数添加进去，如程序清单 2.6 所示。

程序清单 2.6 zy_m3352_dev_cfg 数组

```
static struct evm_dev_cfg zy_m3352_dev_cfg[] = {
        ......
        {spi0_int , DEV_ON_BASEBOARD , PROFILE_NONE},
        ......
};
```

增加完这段代码后将内核重新编译。在内核启动的时候，会为这个设备建立一个 spi_device 并和 0 号 SPI 总线的驱动进行绑定。同时内核会为这个设备申请一个主设备号为 153 的的设备号，次设备号和注册的顺序有关，最多支持 32 个同类设备。

内核重新编译并重启之后，如果系统中运行了 udev，/dev 下就会生成一个 spidevX.D 设备节点，其中 X 是总线编号（即 bus_num），D 是片选号。对于程序清单 2.5 的代码应该自动生成的设备节点是 spidev1.0。

一般 SPI 控制器驱动由芯片厂商提供，开发者所要在内核做的工作就是添加类似程序清单 2.5 的内容。这样内核空间的工作减少了，用户空间的工作量加大了，因为用户空间的开发者需要全面了解 SPI 设备的工作方式和接口协议。

2.4 用户空间同设备节点的接口

对于/dev/spidevX.D 设备节点，可以进行各种操作，这一小节介绍它支持的函数接口。

1. open/close

打开和关闭设备节点没有特别之处，直接使用 open/close 就可以了。

2. read/write

读写 SPI 设备可以直接使用 read/write 函数，但是每次读或者写的大小不能大于 4096Byte。

3. IOCTL 命令

用户空间对 spidev 设备节点使用 IOCTL 命令失败会返回-1。

- SPI_IOC_RD_MODE

读取 SPI 设备对应的 spi_device.mode，mode 的含义请参考程序清单 2.1。使用的方法如下：

```
ioctl(fd, SPI_IOC_RD_MODE, &mode);
```

其中第三个参数是一个 uint8_t 类型的变量。

- SPI_IOC_WR_MODE

设置 SPI 设备对应的 spi_device.mode。使用的方式如下：

```
ioctl(fd, SPI_IOC_WR_MODE, &mode);
```

- SPI_IOC_RD_LSB_FIRST

查看设备传输的时候是否先传输低比特位。如果是的话，返回 1。使用的方式如下：

```
ioctl(fd, SPI_IOC_RD_LSB_FIRST, &lsb);
```

其中 lsb 是一个 uint8_t 类型的变量。返回的结果存在 lsb 中。

- SPI_IOC_WR_LSB_FIRST

设置设备传输的时候是否先传输低比特位。当传入非零的时候，低比特在前，当传入 0 的时候高比特在前（默认）。使用的方式如下：

```
ioctl(fd, SPI_IOC_WR_LSB_FIRST, &lsb);
```

- SPI_IOC_RD_BITS_PER_WORD

读取 SPI 设备的字长。使用的方式如下：

```
ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
```

其中 bits 是一个 uibt8_t 类型的变量。返回的结果保存在 bits 中。

- SPI_IOC_WR_BITS_PER_WORD

设置 SPI 通信的字长。使用的方式如下：

```
ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
```

- SPI_IOC_RD_MAX_SPEED_HZ

读取 SPI 设备的通信的最大时钟频率。使用的方式如下：

```
ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
```

其中 speed 是一个 uint32_t 类型的变量。返回的结果保存在 speed 中。

- SPI_IOC_WR_MAX_SPEED_HZ

设置 SPI 设备的通信的最大时钟频率。使用的方式如下：

```
ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
```

- SPI_IOC_MESSAGE(N)

一次进行双向/多次读写操作。使用的方式如下：

```
struct spi_ioc_transfer  xfer[2];

......

status = ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
```

其中 N 是本次通信中 xfer 的数组长度。spi_ioc_transfer 的信息请参考程序清单 2.2。

2.5 用户空间的测试例程

为了能够得到可以直接通信的 SPI 设备节点，按照 2.3 节的内容修改内核。定义 spi_board_info info 主要初始化四个成员，modalias 一定要是“spidev”， max_speed_hz 根据具体的 SPI 设备手册来定义，笔者的的 SPI 设备是挂在 0 号总线上，所以 bus_num 初始化为 1（注意不是 0），由于片选号为 0，所以 chip_select 初始化为 0。代码如程序清单 2.5 所示。用户自己的 spi_board_info info 初始值，应该根据自己开发板上的实际情况来定，不一定和程序清单 2.5 一样。

修改好内核代码之后，将内核重新编译并下载到开发板，重启之后文件系统的/dev 下便出现了一个名为 spidev1.0 的设备节点。可以在这个设备节点上使用 2.4 节的函数来操作这个设备节点了。

在内核源码的 documentation/spi/spidev_test.c 和 documentation/spi/spi_fdx.c 中有关于用户空间操作 SPI 设备的例程。这两个例程仅仅演示各个函数的使用方法，并不一定能在实际的开发板上运行成功。

下面为用户提供一个可以在 3352 开发板上运行的测试例程，程序通过/dev/spidev1.0 来读写核心板上 SPI 接口的 flash 芯片 93LC46B。93LC46B 的连接原理图如图 2.3 所示。

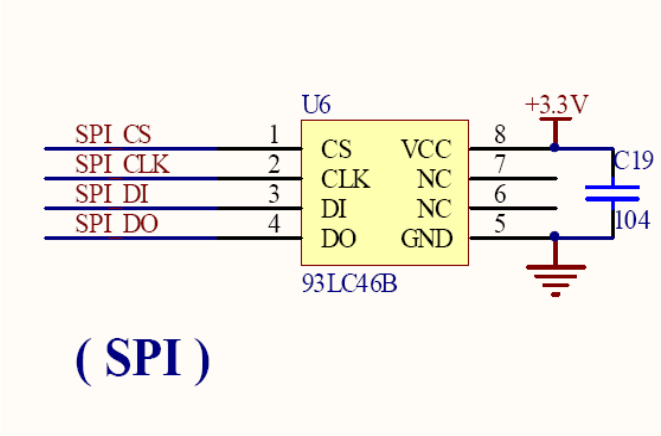

图 2.3 93LC46B 连接原理图

94LC46B 有 8 个引脚，其中和 SPI 总线直接相关的是 CS（片选）、SO（输出）、SI（输入）和 SCK（时钟）这四个引脚。值得注意的就是，linux3.2 中的 spi 控制器的 CS 片选都是低电平使能的，而 94LC46B 的 CS 片选却是高电平使能，因此，在这次测试中把 94LC46B 的 CS 端接到 3.3V 的电源上，其他三根信号线则依次与对应端口接好。

93LC46B 是 SPI 接口的 flash 芯片，容量大小为 64x16 bit (1Kbit)。由于其读写数据为 16 位，所以 93LC46B 的操作地址范围为 0x00~0x3f。94LC46B 支持的最高通信时钟速率高达 2M，只支持程序清单 2.1 中 SPI_MODE_0。

操作 93LC46B 的方式是首先传入一位的 SB 码，然后传入操作命令码，紧接着传入地址和数据（如果必要的话）。其数据格式如表 2.1 所示。

表 2.1 93LC46B 数据格式

指令	SB	操作码	地址	输入数据	输出数据	所需周期

ERASE	1	11	A5	A4	A3	A2	A1	A0	——	空闲/忙	9
ERAL	1	00	1	0	x	x	x	x	——	空闲/忙	9
EWDS	1	00	0	0	x	x	x	x	——	高阻态	9
EWEN	1	00	1	1	x	x	x	x	——	高阻态	9
READ	1	10	A5	A4	A3	A2	A1	A0	——	D15-D0	25
WRITE	1	01	A5	A4	A3	A2	A1	A0	D15-D0	空闲/忙	25
WRAL	1	00	0	1	x	x	x	x	D15-D0	空闲/忙	25

下面的程序仅仅作为用户空间操作 SPI 设备的例子，并不是完备的用户空间的 flash 驱动。对 93LC46B 操作的详细方法请参考它的数据手册。代码如程序清单 2.7 所示。

程序清单 2.7 用户空间使用 SPI 例程

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#define SPIDEV                  "/dev/spidev1.0"        /*  设备号 */
/*一系列的操作命令*/
#define FLASH_ERASE             0x07                    /* erase a sector(1K)      */
#define FLASH_ERAL              0x12                    /* erase all               */
#define FLASH_EWDS              0x10                    /* write disable           */
#define FLASH_EWEN              0x13                    /* write enable            */
#define FLASH_READ              0x06                    /* read start              */
#define FLASH_WRITE             0x05                    /* write start             */
#define FLASH_WRAL              0x11                    /* write all               */
/*相关参数宏定义*/
#define MAX_ADDR                0x3f                    /*  最大操作地址           */
#define FLASH_MODE              0                       /*  芯片模式               */
#define FLASH_BITS              16                      /*  读写数据的位数         */
#define FLASH_MAX_SPEED         2000000                 /*  最大频率               */
```

```
#define ARRAY_SIZE(a) (sizeof(a)/sizeof((a)[0]))

/*相关参数*/
static const char *device = SPIDEV;
static unsigned char mode = FLASH_MODE;
static unsigned char bits = FLASH_BITS;
static unsigned int    speed= FLASH_MAX_SPEED;

/*数据打印*/
static void dis_array(const char *info,const unsigned char *buf,unsigned int len)
{
        int i;
        int tmp=0;
        int flag=0;
        printf("%s : \n",info);
        for(i=0;i<len;i++)
        {
                printf("%x ",buf[i]);
                if(tmp != 31)
                        tmp ++;
                else
                {
                        printf("\n");
                        tmp = 0;
                }
        }
}

/*数据读写*/
static int spi_write_read(int fd, const char *wrbuf, unsigned int wrlen,
                char *rdbuf, unsigned int rdlen)
{
        int ret = 0;
        struct spi_ioc_transfer tr[2] = {
                {
```

```
                .tx_buf = (unsigned long)wrbuf,
                .rx_buf = 0,
                .len = wrlen,
                .speed_hz = 500000,
        },
        {
                .tx_buf = 0,
                .rx_buf = (unsigned long)rdbuf,
                .len = rdlen,
                .speed_hz = 500000,
        },
    };

    ret = ioctl(fd, SPI_IOC_MESSAGE(2), tr);

    return ret;
}

int main(int argc, const char *argv[])
{
    unsigned char   rdbuf[128] = {0};
    unsigned char   wrbuf[128]= {0};
    unsigned char   cmd        = 0;
    unsigned char   strg       = 0;
    int             ret;

    int fd = open(device, O_RDWR);                  /*打开设备*/
    if (fd < 0) {
        printf("open "SPIDEV"failed\n");
        goto error;
    }

    ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);        /*设置模式*/
    if(ret < 0)
        printf("can't   set spi mode\n");
```

```
ioctl(fd, SPI_IOC_RD_MODE, &mode);
printf("the flash mode : %d \n",mode);

ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD,&bits); /*设置数据位数*/
if(ret < 0)
        printf("can't set spi bits\n");

ioctl(fd, SPI_IOC_RD_BITS_PER_WORD,&bits);
printf("the flash word : %d \n",bits);

ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);/*设置读写的最大频率*/
if(ret < 0)
        printf("can't set spi speed\n", speed);

ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
printf("max speed : %d HZ\n", speed);

/* remove write protect */
cmd = FLASH_EWEN;
write(fd, &cmd, sizeof(cmd));

/* read and display */
cmd = FLASH_READ;
wrbuf[0] = cmd;
ret    = spi_write_read(fd, wrbuf, 1, rdbuf, sizeof(rdbuf));  /*读取原来数据*/
if(ret < 0)
        printf("write data error\n");
dis_array("content in flash", rdbuf, sizeof(rdbuf));            /*显示数据*/

/* write data at 0 address */
cmd        = FLASH_WRITE;
wrbuf[0]    =       cmd; //cmd
wrbuf[1]    =       0x12;
wrbuf[2]    =       0x34;
wrbuf[3]    =       0x56;
wrbuf[4]    =       0x78;
```

```
    ret = spi_write_read(fd,wrbuf,5,0,0);                    /*写入数据*/
    if(ret < 0)
        printf("write data error\n");
    else
        printf("%d bytes has been writen to %d addresss\n",4,0);

    /* read 6 bytes form 0 address */
    cmd   = FLASH_READ;
    wrbuf[1] = cmd;
    ret = spi_write_read(fd, wrbuf, 1, rdbuf, sizeof(rdbuf));    /*读取改变后的数据*/
    if(ret < 0)
        printf("read data error\n");
    dis_array("new content in flash", rdbuf, sizeof(rdbuf));

exit:
    close(fd);
error:
    return 0;
}
```

编译程序并下载到开发板上运行，输出的结果如图 2.4 所示。

```
[root@M3352 linux_spi]# ./spi_test
the flash mode : 0
the flash word : 16
max speed : 500000 HZ
content in flash :
82 6b c0 a0 82 6b c0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 bytes has been writen to 0 addresss
new content in flash :
12 34 56 78 82 6b c0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[root@M3352 linux_spi]#
```

图 2.4 SPI 测试结果

3. 免责声明

广州周立功单片机科技有限公司所提供的所有服务内容旨在协助客户加速产品的研发进度，在服务过程中所提供的任何程序、文档、测试结果、方案、支持等资料和信息，都仅供参考，客户有权不使用或自行参考修改，本公司不提供任何的完整性、可靠性等保证，若在客户使用过程中因任何原因造成的特别的、偶然的或间接的损失，本公司不承担任何责任。

销售与服务网络

广州周立功单片机科技有限公司

地址：广州市天河北路 689 号光大银行大厦 12 楼 F4
邮编：510630
传真：(020)38730925
网址：www.zlgmcu.com
电话：(020)38730916 38730917 38730972 38730976 38730977

广州专卖店

地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634 87569917
传真：(020)87578842

南京周立功

地址：南京市珠江路 280 号珠江大厦 1501 室
电话：(025)68123920 68123923 68123901
传真：(025)68123900

北京周立功

地址：北京市海淀区知春路 108 号豪景大厦 A 座 19 层
电话：(010)62536178 62536179 82628073
传真：(010)82614433

重庆周立功

地址:重庆市九龙坡区石桥铺科园一路二号大西洋国际大厦（赛格电子市场）2705 室
电话：(023)68796438 68796439
传真：(023)68796439

杭州周立功

地址：杭州市天目山路 217 号江南电子大厦 502 室
电话：(0571)89719480 89719481 89719482
89719483 89719484 89719485
传真：(0571)89719494

成都周立功

地址：成都市一环路南二段 1 号数码科技大厦 403 室
电话：(028)85439836 85437446
传真：(028)85437896

深圳周立功

地址:深圳市福田区深南中路 2072 号电子大厦 12 楼 1203
电话:(0755)83781788（5 线） 83782922 83273683
传真：(0755)83793285

武汉周立功

地址:武汉市洪山区广埠屯珞瑜路 158 号 12128 室(华中电脑数码市场)
电话：(027)87168497 87168297 87168397
传真：(027)87163755

上海周立功

地址：上海市北京东路 668 号科技京城东座 12E 室
电话：(021)53083452 53083453 53083496
传真：(021)53083491

西安办事处

地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296 83063000 87881295
传真：(029)87880865

厦门办事处

E-mail：sales.xiamen@zlgmcu.com

沈阳办事处

E-mail：sales.shenyang@zlgmcu.com