(1条消息)Linux系统自带spi驱动加载及应用程序编写方法详解 - kunkliu的博客 - CSDN博客

转载地址：http://blog.csdn.net/borntox/article/details/51871480

硬件平台：飞思卡尔IMX6，

内核版本：kernel3.0.35

Linux系统中，和I2C一样，SPI也有系统自带的设备驱动程序，位于源码目录下drivers/spi/spidev.c，以下为驱动的移植和对应应用程序编写方法

要将此设备驱动加入到内核中，要做两件事情

第一：将此驱动编译进内核

步骤：make menuconfig

Device Drivers ->

<*>SPI support ->

<*>User mode SPI device driver support

第二：在平台文件arch/arm/mach-mx6/board-mx6q_sabresd.c 中添加对spidev的设备注册

步骤：

1、准备spi_board_info变量（全局变量）

```
static struct spi_board_info spidev_ecspi2_board_info[] __initdata = {
{
/* The modalias must be the same as spi device driver name */
.modalias = "spidev",
.max_speed_hz = 20000000,
.bus_num = 1,
.chip_select = 0,

},
};
```

2、注册spi_board_info变量到内核中，要在平台硬件初始化的函数中执行本段代码

```
spi_register_board_info(spidev_ecspi2_board_info,
ARRAY_SIZE(spidev_ecspi2_board_info));
```

注意：上面两个步骤是原则，必不可少的，但是具体的平台会有一些其他更多的修改，比如笔者使用的是飞思卡尔IMX6，还需要将GPIO口进行初始化，初始化为SPI功能

具体操作见以下补丁，源码下载地址点击打开链接

在对驱动代码进行移植之后，重新编译内核，下载到开发板上，即可看到spi设备/dev/spidev1.0，标识着SPI驱动移植成功

在对驱动代码进行修改之后，需要根据驱动的架构来完成应用程序的编写，在内核源代码Documentation/spi目录下有一个spidev_test.c文件，是内核作者提供给Linux开发人员的参考文档，笔者也是参考此文件来编写的应用程序

应用程序无非是open、close、read、write、ioctl的使用。open，close没什么好说的，下面具体说下ioctl、read和write的使用。

spi应用程序编写步骤**:**

第一：open

第二：**ioctl** ，**ioctl**有九种**cmd**，分别对应不同的**arg**

a、设置或获取**SPI**工作模式

SPI_IOC_RD_MODE

用法：

mode = mode | SPI_MODE_0 | SPI_CS_HIGH | SPI_LSB_FIRST | SPI_LOOP

ioctl(fd, SPI_IOC_WR_MODE, &mode)；

注意：前面四种是对**SCK**时钟信号空闲时的电平，和采样时刻的选择，四个只能选择其中一种，后面的五种可以用或的形式选择任意几个，使用方法如上

b、设置或获取**SPI**读写是从高位还是低位开始

SPI_IOC_RD_LSB_FIRST

SPI_IOC_RD_BITS_PER_WORD

SPI_IOC_RD_MAX_SPEED_HZ

SPI_IOC_WR_MAX_SPEED_HZ

ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);

e、传输数据

SPI_IOC_MESSAGE(n)

ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);

第三：read或write

用法：和大多数的设备read函数一样的用法，但是每次读或者写的大小不能大于4096Byte。

char* buf[n];

read(fd,buf,sizeof(buf));或者write(fd,buf,sizeof(buf));

第四：close

应用程序源码

```
#include <stdint.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <getopt.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <linux/types.h>

#include <linux/spi/spidev.h>

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

staticvoid pabort(constchar *s)

{

    perror(s);

    abort();

}
```

```
staticconst char*device = "/dev/spidev1.0";
staticuint8_t mode;
staticuint8_t bits = 8;
staticuint32_t speed = 50000;
staticuint16_t delay;
unsigned charbuf_me[1] = {0x55};
staticvoid transfer(intfd)
{
    intret;
    uint8_t tx[] = {
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0x40, 0x00, 0x00, 0x00, 0x00, 0x95,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xDE, 0xAD, 0xBE, 0xEF, 0xBA, 0xAD,
        0xF0, 0x0D,
    };
    uint8_t rx[ARRAY_SIZE(tx)] = {0, };
    structspi_ioc_transfer tr = {
        .tx_buf = (unsignedlong)tx,
        .rx_buf = (unsignedlong)rx,
        .len = ARRAY_SIZE(tx),
        .delay_usecs = delay,
        .speed_hz = speed,
        .bits_per_word = bits,
    };
    ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
    if(ret < 1)
        pabort("can't send spi message");
```

```
    for(ret = 0; ret < ARRAY_SIZE(tx); ret++) {
        if(!(ret % 6))
            puts("");
        printf("%.2X ", rx[ret]);
    }
    puts("");
}
staticvoid print_usage(constchar *prog)
{
    printf("Usage: %s [-DsbdlHOLC3]\n", prog);
    puts("  -D --device   device to use (default /dev/spidev1.1)\n"
         "  -s --speed    max speed (Hz)\n"
         "  -d --delay    delay (usec)\n"
         "  -b --bpw      bits per word \n"
         "  -l --loop     loopback\n"
         "  -H --cpha     clock phase\n"
         "  -O --cpol     clock polarity\n"
         "  -L --lsb      least significant bit first\n"
         "  -C --cs-high  chip select active high\n"
         "  -3 --3wire    SI/SO signals shared\n");
    exit(1);
}
staticvoid parse_opts(intargc, char *argv[])
{
    while(1) {
        staticconst structoption lopts[] = {
            {"device",  1, 0, 'D' },
            {"speed",   1, 0, 's' },
            {"delay",   1, 0, 'd' },
```

```
        {"bpw",     1, 0, 'b' },
        {"loop",    0, 0, 'l' },
        {"cpha",    0, 0, 'H' },
        {"cpol",    0, 0, 'O' },
        {"lsb",     0, 0, 'L' },
        {"cs-high", 0, 0, 'C' },
        {"3wire",   0, 0, '3' },
        {"no-cs",   0, 0, 'N' },
        {"ready",   0, 0, 'R' },
        { NULL, 0, 0, 0 },
    };
    intc;
    c = getopt_long(argc, argv,"D:s:d:b:lHOLC3NR", lopts, NULL);
    if(c == -1)
        break;
    switch(c) {
    case'D':
        device = optarg;
        break;
    case's':
        speed =atoi(optarg);
        break;
    case'd':
        delay =atoi(optarg);
        break;
    case'b':
        bits =atoi(optarg);
        break;
    case'l':
```

```
                    mode |= SPI_LOOP;
                    break;
                case'H':
                    mode |= SPI_CPHA;
                    break;
                case'O':
                    mode |= SPI_CPOL;
                    break;
                case'L':
                    mode |= SPI_LSB_FIRST;
                    break;
                case'C':
                    mode |= SPI_CS_HIGH;
                    break;
                case'3':
                    mode |= SPI_3WIRE;
                    break;
                case'N':
                    mode |= SPI_NO_CS;
                    break;
                case'R':
                    mode |= SPI_READY;
                    break;
                default:
                    print_usage(argv[0]);
                    break;
                }
            }
        }
        int main(int argc, char *argv[])
```

```
{
    intret = 0;
    intfd;
    parse_opts(argc, argv);
    fd = open(device, O_RDWR);
    if(fd < 0)
        pabort("can't open device");
        mode = mode | SPI_MODE_1 | SPI_LSB_FIRST | SPI_LOOP;
    ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
    if(ret == -1)
        pabort("can't set spi mode");
    ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
    if(ret == -1)
        pabort("can't get spi mode");
    ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
    if(ret == -1)
        pabort("can't set bits per word");
    ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
    if(ret == -1)
        pabort("can't get bits per word");
    ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
    if(ret == -1)
        pabort("can't set max speed hz");
    ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
    if(ret == -1)
        pabort("can't get max speed hz");
    printf("spi mode: %d\n", mode);
    printf("bits per word: %d\n", bits);
    printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);
        while(1){
```

```
                write(fd,buf_me,1);
            }
        close(fd);
        returnret;
    }
```