```
/*
 * SPI testing utility (using spidev driver)
 *
 * Copyright (c) 2007  MontaVista Software, Inc.
 * Copyright (c) 2007  Anton Vorontsov <avorontsov@ru.mvista.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License.
 *
 * Cross-compile with cross-gcc -I/path/to/cross-kernel/include
 */

#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]))

static void pabort(const char *s)
{
        perror(s);
        abort();
}

static const char *device = "/dev/spidev1.1";
static uint8_t mode;
static uint8_t bits = 8;
static uint32_t speed = 500000;
static uint16_t delay;

static void transfer(int fd)
{
        int ret;
        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, };
        struct spi_ioc_transfer tr = {
                .tx_buf = (unsigned long)tx,
                .rx_buf = (unsigned long)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("");
}

static void print_usage(const char *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);
}

static void parse_opts(int argc, char *argv[])
{
        while (1) {
                static const struct option 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 },
                };
                int c;

                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[])
{
        int ret = 0;
        int fd;

        parse_opts(argc, argv);

        fd = open(device, O_RDWR);
        if (fd < 0)
                pabort("can't open device");

        /*
```

```
	 * spi mode
	 */
	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");

	/*
	 * bits per word
	 */
	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");

	/*
	 * max speed hz
	 */
	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);

	transfer(fd);

	close(fd);

	return ret;
}
```