Accessing SPI NOR flash registers in Linux user space

About this document

Scope and purpose

This application note describes how to access configuration registers in Infineon SPI NOR flash devices in Linux-based platforms. It introduces full source code and usage examples of a simple SPI NOR flash utility based on the Linux user mode SPI device driver, *spidev*.

Intended audience

This is intended for users who use Infineon SPI NOR flash devices in Linux-based platforms. It is assumed that users have knowledge and experience of software development in Linux.

Table of contents

1 Introduction

Infineon S25HL-T, S25HS-T, S25FL-L, and S25FS-S SPI NOR flash devices have separate non-volatile and volatile registers. During powerup, hardware reset, or software reset, the contents in the non-volatile registers are automatically loaded to the counterpart volatile registers. Non-volatile registers are used to apply default settings before system boot, while volatile registers are used is to change settings at system runtime. This is because the non-volatile registers are based on flash memory cells which have limited update cycles, take a longer time to update as compared to volatile registers, and are intolerant to power interruption during update.

In general, non-volatile registers can be updated by flash programmers equipped in production facilities. On the other hand, engineers who develop and evaluate the systems may need a way to update non-volatile registers in their lab, especially a way of the in-system programming.

In Linux-based platforms, Memory Technology Device (MTD) drivers and related user space commands provide access to the flash memory array, but not to flash registers. This application note introduces a simple way to access flash registers, based on the Linux user mode SPI device driver (spidev). It describes how to activate `spidev` in kernel configuration, modify the device tree, and inspect the source code of the flash utility.

See Linux kernel documentation (*Documentation/spi/spidev)* for the basics of the spidev driver; See the corresponding device datasheets for information on SPI NOR flash registers.

2 Activating spidev

2.1 Enable spidev in the kernel configuration

In menuconfig, enable user mode SPI device driver support. You can also manually enable the CONFIG_SPI_SPIDEV option in the kernel configuration file.

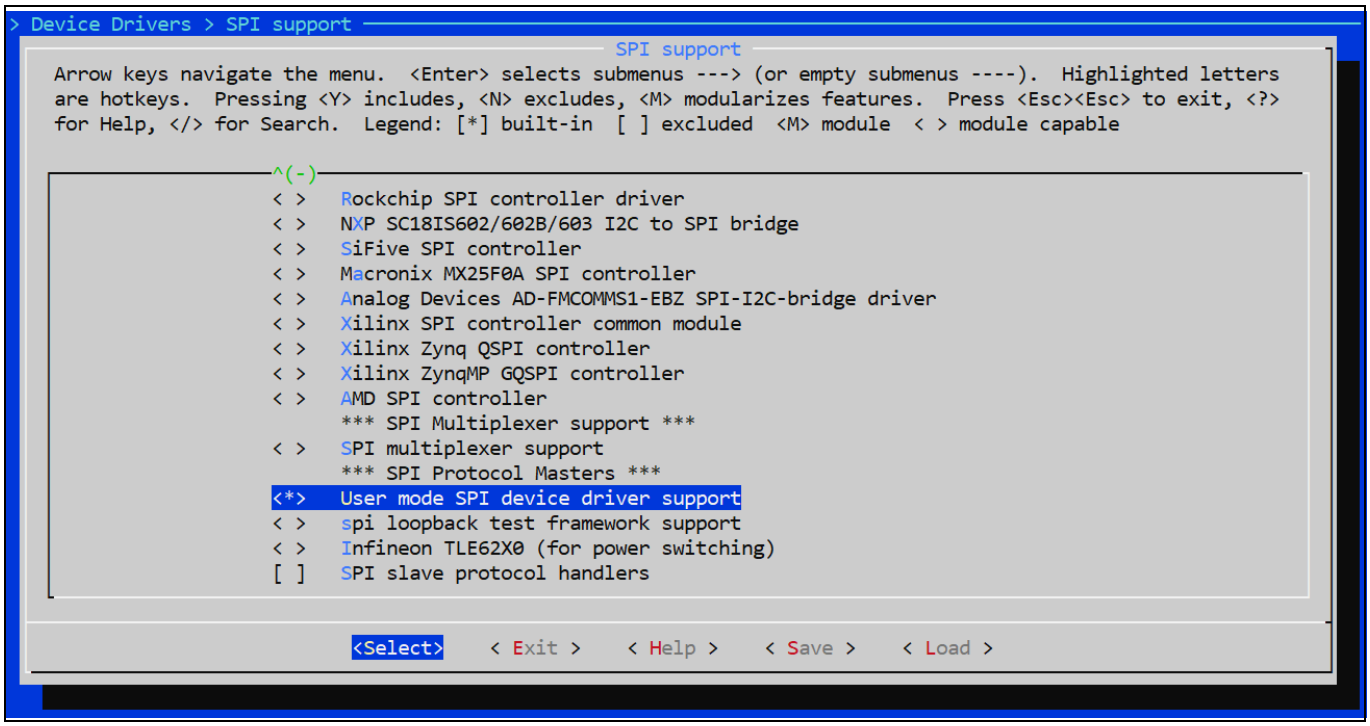

Figure 1 Kernel configuration

2.2 Bind spidev to SPI controller in device tree

In most of cases, a spi-nor flash is already binded to the SPI controller. Replace the spi-nor flash node with the spidev node because only one node can be binded at a time. Code Listing 1 shows an example.

Code Listing 1 Binding spidev to SPI controller

```
&qspi {
        status = "okay";

        /*
        flash@0 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "jedec,spi-nor";
                reg = <0>;
                spi-max-frequency = <40000000>;
                spi-tx-bus-width = <4>;
                spi-rx-bus-width = <4>;
                m25p,fast-read;
        };
        */

        spidev@0 {
                compatible = "spidev";
                reg = <0>;
                spi-max-frequency = <40000000>;
        };
};

```

2.3 Device node in sysfs

After you have enabled spidev in the kernel configuration and binded it to the SPI controller in the device tree, the sysfs node for `spidev` will appear like */dev/spidevB.C*, where *B* and *C* indicate the bus and chip select number respectively.

For example, if your platform has only one SPI bus and chip select, the sysfs node will be */dev/spidev0.0*.

3 Accessing SPI NOR flash registers via spidev

3.1 User space utility – *sf_utils*

Code Listing 2 shows the source code of a simple user space utility program named “sf_utils”. This program can access to Infineon SPI NOR Flash registers by using spidev.

Table 1 Functions

Function	Description
`main()`	Takes the sysfs device node like */dev/spidev0.0*, sub-command name, and parameters as arguments. It opens the device node, calls subroutines corresponding to sub-commands, and closes the device node.
`Rdid()`	Transmits the Read ID (RDID) instruction code (0x9F) and receives 3-byte device ID values. RDID is typically used for connectivity check between SPI controller and SPI NOR Flash.
`Rdar()`	Transmits the Read Any Register (RDAR) instruction code (0x65) followed by 3-byte register address, one dummy byte, and receives the one-byte register value. The length of the address and dummy byte can be modified depending on the SPI NOR Flash type and its configuration.
`Wrar()`	Transmits the Write Enable (WREN) instruction code (0x06), the Write Any Register (WRAR) instruction code (0x71) followed by 3-byte register address, and register value to be written to the SPI NOR Flash.
`Transfer()`	Performs transmission and reception underneath the functions above by calling `ioctl()`. An array of `struct spi_ioc_transfer` is used to point to the buffers and data lengths.

Code Listing 2 sf_utils.c

```
001  #include <stdint.h>
002  #include <stdio.h>
003  #include <stdlib.h>
004  #include <string.h>
005  #include <fcntl.h>
006  #include <unistd.h>
007  #include <sys/ioctl.h>
008  #include <linux/spi/spidev.h>
009
010  int transfer(int fd, const uint8_t *tx_buf,unsigned tx_len,
011               uint8_t *rx_buf, unsigned rx_len)
012  {
013        int ret, i = 1;
014        struct spi_ioc_transfer x[2];
015
016        memset(x, 0, sizeof(x));
017
018        x[0].tx_buf = (unsigned)tx_buf;
019        x[0].len = tx_len;
020
021        if (rx_buf) {
022              x[1].rx_buf = (unsigned)rx_buf;
023              x[1].len = rx_len;
024              i++;
```

Accessing SPI NOR flash registers via spidev

Code Listing 2 sf_utils.c

```
025                 }
026
027                 ret = ioctl(fd, SPI_IOC_MESSAGE(i), x);
028
029                 if (ret != tx_len + rx_len) {
030                       printf("spi transfer error: %d\n", ret);
031                       return -1;
032                 }
033
034                 if (rx_buf) {
035                       for (i = 0; i < rx_len; i++)
036                             printf("%02X ", rx_buf[i]);
037                       printf("\n");
038                 }
039
040                 return 0;
041           }
042
043           int rdid(int fd)
044           {
045                 uint8_t tx = 0x9F, rx[3];
046                 return transfer(fd, &tx, 1, rx, 3);
047           }
048
049           int rdar(int fd, uint32_t addr)
050           {
051                 uint8_t tx[5], rx;
052
053                 tx[0] = 0x65; /* Read Any Register */
054                 tx[1] = addr >> 16;
055                 tx[2] = addr >> 8;
056                 tx[3] = addr;
057                 tx[4] = 0; /* dummy */
058
059                 return transfer(fd, tx, 5, &rx, 1);
060           }
061
062           int wrar(int fd, uint32_t addr, uint8_t val)
063           {
064                 uint8_t tx[5];
065                 int err;
066
067                 tx[0] = 0x06; /* Write Enable */
068                 err = transfer(fd, tx, 1, NULL, 0);
069                 if (err)
070                       return err;
071
072                 tx[0] = 0x71; /* Write Any Register */
073                 tx[1] = addr >> 16;
074                 tx[2] = addr >> 8;
075                 tx[3] = addr;
076                 tx[4] = val;
077                 return transfer(fd, tx, 5, NULL, 0);
078           }
```

Accessing SPI NOR flash registers via spidev

Code Listing 2 sf_utils.c

```
079
080       void usage(void)
081       {
082             printf("usage: sf_utils <device> rdid\n"
083             "       sf_utils <device> rdar <address>\n"
084             "       sf_utils <device> wrar <address> <value>\n");
085       }
086
087       int main(int argc, char *argv[])
088       {
089             int fd, err = -1;
090
091             if (argc < 3) {
092                   usage();
093                   return -1;
094             }
095
096             fd = open(argv[1], O_RDWR);
097             if (fd < 0) {
098                   printf("cannot open device\n");
099                   return -1;
100             }
101
102             if (!strcmp(argv[2], "rdid"))
103                   err = rdid(fd);
104             else if (!strcmp(argv[2], "rdar") && argc == 4)
105                   err = rdar(fd, strtoul(argv[3], NULL, 16));
106             else if (!strcmp(argv[2], "wrar") && argc == 5)
107                   err = wrar(fd, strtoul(argv[3], NULL, 16),
108                               strtoul(argv[4], NULL, 16));
109             else
110                   usage();
111
112             close(fd);
113
114             return err;
115       }
116
```

3.2 Usage examples of *sf_utils* with Infineon S25FL256L

1. Obtain the sysfs node for spidev:

```
$ ls /dev/spidev*
/dev/spidev0.0
```

2. Read device ID:

```
$ sf_utils /dev/spidev0.0 rdid
01 60 19
```

Make sure that the 3-byte ID values are expected ones (Table 2).

Table 2 S25FL256L Device ID

Byte address	Data	Notes
00h	01h	Manufacturer ID
01h	60h	Device ID most significant byte – memory interface type
02h	19h (256Mb)	Device ID least significant byte – density and features

3. Read Configuration Register 2 Non-volatile (CR2NV -000003h). The factory default value of CR2NV is 60h.

```
$ sf_utils /dev/spidev0.0 rdar 000003
60
```

The CR2NV[6:5] controls the I/O signal output impedance (Table 3).

Table 3 Output impedance configuration in S25FL1256L

CR2NV[6:5]	Typ. impedance to V_{SS}	Typ. impedance to V_{DD}	Notes
00	18 Ω	21 Ω	
01	26 Ω	28 Ω	
10	47 Ω	45 Ω	
11	71 Ω	64 Ω	Factory default

4. Write the CR2NV register to change the output impedance from the default values (71/64 Ω to 47/45 Ω).

```
$ sf_utils /dev/spidev0.0 wrar 000003 40
```

5. Read Status Register 1 Volatile (SR1V – address 800000h) to check the completion of WRAR operation. The value should be 00h if the WRAR operation is completed successfully.

```
$ sf_utils /dev/spidev0.0 rdar 800000
00
```

6. Read Configuration Register 2 Volatile (CR2V – address 800003h) to confirm. The volatile register is updated when the non-volatile register is written.

```
$ sf_utils /dev/spidev0.0 rdar 800003
40
```

4 Conclusion

This application note introduces a utility program that runs on Linux user space to access SPI NOR flash registers. The utility supports several SPI NOR flash commands in addition to the register access commands.

Revision history

Document version	Date of release	Description of changes
**	2021-08-10	New application note

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-08-10
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to www.cypress.com/support

Document reference
002-33637 Rev.**

IMPORTANT NOTICE
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.