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

About this document.............................................................................................................................................. 1
Table of contents.................................................................................................................................................... 1
1 Introduction .................................................................................................................................................. 2
2 Activating spidev ................................................................. Enable spidev in the kernel configuration ................................................................. 3
  2.1 Enable spidev in the kernel configuration ................................................................. 3
  2.2 Bind spidev to SPI controller in device tree .......................................................... 4
  2.3 Device node in sysfs ................................................................................................. 4
3 Accessing SPI NOR flash registers via spidev ....................................................... User space utility – sf-utils ......................................................................................... 5
  3.1 User space utility – sf-utils .................................................................................. 5
  3.2 Usage examples of sf-utils with Infineon S25FL256L .......................................... 8
4 Conclusion .................................................................................................................................................. 9
Revision history...................................................................................................................................................10
Accessing SPI NOR flash registers in Linux user space

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 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 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.
Activating spidev

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.

![Kernel configuration](image-url)

Figure 1 Kernel configuration
Activating spidev

## 2.2 Bind spidev to SPI controller in device tree

In most 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

```c
&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`. 
Accessing SPI NOR flash registers in Linux user space

Accessing SPI NOR flash registers via spidev

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>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main()</td>
<td>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.</td>
</tr>
<tr>
<td>Rdid()</td>
<td>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.</td>
</tr>
<tr>
<td>Rdar()</td>
<td>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.</td>
</tr>
<tr>
<td>Wrar()</td>
<td>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.</td>
</tr>
<tr>
<td>Transfer()</td>
<td>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.</td>
</tr>
</tbody>
</table>

Code Listing 2 sf_utils.c

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

int transfer(int fd, const uint8_t *tx_buf,unsigned tx_len,
             uint8_t *rx_buf, unsigned rx_len)
{
    int ret, i = 1;
    struct spi_ioc_transfer x[2];
    memset(x, 0, sizeof(x));
    x[0].tx_buf = (unsigned)tx_buf;
    x[0].len = tx_len;
    if (rx_buf) {
        x[1].rx_buf = (unsigned)rx_buf;
        x[1].len = rx_len;
        i++;
```
Accessing SPI NOR flash registers in Linux user space

Accessing SPI NOR flash registers via spidev

**Code Listing 2**

```
sf_utils.c

025  }
026  ret = ioctl(fd, SPI_IOC_MESSAGE(i), x);
027  if (ret != tx_len + rx_len) {
028      printf("spi transfer error: %d
", ret);
029      return -1;
030  }
031  if (rx_buf) {
032      for (i = 0; i < rx_len; i++)
033          printf("%02X ", rx_buf[i]);
034      printf("\n");
035  }
036  return 0;
037 }
038
039 int rdid(int fd)
040 {
041    uint8_t tx = 0x9F, rx[3];
042    return transfer(fd, &tx, 1, rx, 3);
043 }
044
045 int rdar(int fd, uint32_t addr)
046 {
047    uint8_t tx[5], rx;
048    tx[0] = 0x65; /* Read Any Register */
049    tx[1] = addr >> 16;
050    tx[2] = addr >> 8;
051    tx[3] = addr;
052    tx[4] = 0; /* dummy */
053    return transfer(fd, tx, 5, &rx, 1);
054 }
055
056 int wrar(int fd, uint32_t addr, uint8_t val)
057 {
058    uint8_t tx[5];
059    int err;
060    tx[0] = 0x06; /* Write Enable */
061    tx[1] = addr >> 16;
062    tx[2] = addr >> 8;
063    tx[3] = addr;
064    tx[4] = val;
065    return transfer(fd, tx, 5, NULL, 0);
066 }
067
068 int scheduler(void)
069 {
070    uint8_t tx[5];
071    uint32_t addr = 0x00000000;
072    uint8_t cmd = 0x80;
073    tx[0] = cmd;
074    tx[1] = addr >> 24;
075    tx[2] = addr >> 16;
076    tx[3] = addr >> 8;
077    tx[4] = addr;
078    return transfer(fd, tx, 5, NULL, 0);
079 }
080
081
```

**Accessing SPI NOR flash registers in Linux user space**

**Code Listing 2**

```
sf_utils.c

025  }
026  ret = ioctl(fd, SPI_IOC_MESSAGE(i), x);
027  if (ret != tx_len + rx_len) {
028      printf("spi transfer error: %d
", ret);
029      return -1;
030  }
031  if (rx_buf) {
032      for (i = 0; i < rx_len; i++)
033          printf("%02X ", rx_buf[i]);
034      printf("\n");
035  }
036  return 0;
037 }
038
039 int rdid(int fd)
040 {
041    uint8_t tx = 0x9F, rx[3];
042    return transfer(fd, &tx, 1, rx, 3);
043 }
044
045 int rdar(int fd, uint32_t addr)
046 {
047    uint8_t tx[5], rx;
048    tx[0] = 0x65; /* Read Any Register */
049    tx[1] = addr >> 16;
050    tx[2] = addr >> 8;
051    tx[3] = addr;
052    tx[4] = 0; /* dummy */
053    return transfer(fd, tx, 5, &rx, 1);
054 }
055
056 int wrar(int fd, uint32_t addr, uint8_t val)
057 {
058    uint8_t tx[5];
059    int err;
060    tx[0] = 0x06; /* Write Enable */
061    tx[1] = addr >> 16;
062    tx[2] = addr >> 8;
063    tx[3] = addr;
064    tx[4] = val;
065    return transfer(fd, tx, 5, NULL, 0);
066 }
067
068 int scheduler(void)
069 {
070    uint8_t tx[5];
071    uint32_t addr = 0x00000000;
072    uint8_t cmd = 0x80;
073    tx[0] = cmd;
074    tx[1] = addr >> 24;
075    tx[2] = addr >> 16;
076    tx[3] = addr >> 8;
077    tx[4] = addr;
078    return transfer(fd, tx, 5, NULL, 0);
079 }
080
081
```
Accessing SPI NOR flash registers in Linux user space

Accessing SPI NOR flash registers via spidev

**Code Listing 2**  sf_utils.c

```c
079 080 void usage(void) 081 082    { printf("usage: sf_utils <device> rdid\n" 083 084    " sf_utils <device> rdar <address>\n" 085 086    " sf_utils <device> wrar <address> <value>\n"); 087 088 } 089 090 int main(int argc, char *argv[]) 091 092 { 093    int fd, err = -1; 094 095 if (argc < 3) { 096      usage(); 097      return -1; 098 099    } 100 101    fd = open(argv[1], O_RDWR); 102 103 if (fd < 0) { 104      printf("cannot open device\n"); 105      return -1; 106 107    } 108 109 if (!strcmp(argv[2], "rdid")) 110    err = rdid(fd); 111 else if (!strcmp(argv[2], "rdar") && argc == 4) 112    err = rdar(fd, strtoul(argv[3], NULL, 16)); 113 else if (!strcmp(argv[2], "wrar") && argc == 5) 114    err = wrar(fd, strtoul(argv[3], NULL, 16), 115    strtoul(argv[4], NULL, 16)); 116 else 117    usage(); 118 119 close(fd); 120 121 return err; 122 123 }
```
Accessing SPI NOR flash registers via spidev

**3.2 Usage examples of sf_utils with Infineon S25FL256L**

1. Obtain the sysfs node for spidev:
   
   ```bash
   $ ls /dev/spidev*
   /dev/spidev0.0
   ```

2. Read device ID:
   
   ```bash
   $ 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**

<table>
<thead>
<tr>
<th>Byte address</th>
<th>Data</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>01h</td>
<td>Manufacturer ID</td>
</tr>
<tr>
<td>01h</td>
<td>60h</td>
<td>Device ID most significant byte – memory interface type</td>
</tr>
<tr>
<td>02h</td>
<td>19h (256Mb)</td>
<td>Device ID least significant byte – density and features</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CR2NV[6:5]</th>
<th>Typ. impedance to VSS</th>
<th>Typ. impedance to VDD</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>18 Ω</td>
<td>21 Ω</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>26 Ω</td>
<td>28 Ω</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>47 Ω</td>
<td>45 Ω</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>71 Ω</td>
<td>64 Ω</td>
<td>Factory default</td>
</tr>
</tbody>
</table>

4. Write the CR2NV register to change the output impedance from the default values (71/64 Ω to 47/45 Ω).
   
   ```bash
   $ 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.
   
   ```bash
   $ 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.
   
   ```bash
   $ sf_utils /dev/spidev0.0 rdar 800003
   40
   ```
Conclusion

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

<table>
<thead>
<tr>
<th>Document version</th>
<th>Date of release</th>
<th>Description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>2021-08-10</td>
<td>New application note</td>
</tr>
</tbody>
</table>
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.