Non-volatile storage

Using external, non-volatile memory with the nRF24LE1

nAN-20

Application Note v1.0

Key words

- External flash
- EEPROM
- Non-volatile storage
- nRF24LE1

2011-01-12

Contents

1 Introduction

Many applications require non-volatile memory for storing data that should be retained even when power to a device is lost. For many applications you should also be able to update the stored data for the lifetime of a device, to account for changing operating parameters or conditions.

This data can typically be pairing and encryption information, or logged sensor data, which must be stored when batteries are depleted or the device enters low-power sleep modes.

Flash versions of the nRF24LE1 contain non-volatile flash memory that can be used for non-volatile data storage. However, if the size of this memory doesn't meet your application requirements, or if BOM cost can be reduced by using the cheaper nRF24LE1 OTP variant, you might need to add external non-volatile memory.

This application note describes the two most common types of non-volatile memory, flash and EEPROM, and the two most common interfaces used to connect the memory to the host MCU, namely I^2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface). The application note also has examples of how to connect two specific non-volatile memory devices to nRF24LE1, and provides general tips on how to write driver software for any non-volatile memory device.

Complete software examples with source code are available from Nordic Semiconductor's website at www.nordicsemi.com. The software examples are for the AT25DF021 SPI flash device and for the M24C32 I^2C EEPROM device, as well as other compatible devices. These examples will make it easy to set up the nRF24LE1 for use with any flash or EEPROM device using I^2C or SPI.

Note: While this document targets the nRF24LE1, parts of it are also relevant for the nRF24LU1+. The major difference is the lack of an I^2C interface in the nRF24LU1+, which means one should use an SPI-based memory module with the nRF24LU1+ device.

2 Memory technology

There are a wide range of memory technologies available. The types discussed in this document are currently the most popular technologies for non-volatile storage in electronic systems.

2.1 EEPROM

EEPROM (Electrically Erasable Programmable Read Only Memory) is one of the earlier non-volatile memory technologies available. It allows modification of data on a byte-to-byte basis, and while there is a limit to how many times each byte can be rewritten, it is in the order of tens of thousands in earlier devices, and as many as millions in modern devices.

2.2 Flash memory

Flash memory is a close relative of EEPROM. However, while EEPROM can delete and modify bytes individually, flash memory divides its content into pages. When data is deleted it must be deleted one page at a time; this can complicate the process of altering small parts of the memory content.

The advantage of flash memory is that it is cheaper and takes up less space on the chip. As such, flash memory provides higher capacities compared to an EEPROM device of similar price/size.

While flash memory must be erased page-by-page, many flash devices allow you to write and read bytes individually. Also, many flash devices allow you to write to the same byte several times, as long as the individual bits are only changed from ‘1’ to ‘0’. For example, if you have already written 0x0F to a byte you can change it to 0x00 afterwards without having to erase the entire page. If you want to change any of the bits back to ‘1’ the only solution is to erase the entire page. Erasing a page in flash memory means setting all the bits in the page high (and all the bytes 0xFF), and a write operation can never do anything but clear bits.

Because of these differences EEPROM is usually used where you only need small amounts of data, but where you want the possibility to change single bytes easily. Flash memory is used for larger storage of data that doesn’t have to be manipulated as much, or in applications where large portions of the memory is updated at once.

3 Non-volatile memory interfaces

The most common interfaces used to access EEPROM and flash devices are SPI and I^2C, and the major differences between them are I/O requirements and speed.

3.1 I^2C

I^2C requires only two I/Os from your MCU, and allows you to connect up to 112 devices to the same bus. It also allows multiple masters on the same bus if all the masters support it. Most I^2C devices support speeds of 100 or 400 kbit/sec, and while the I^2C specification also defines a high speed 3.4 Mbit/sec mode few devices support this.

All I^2C slaves on a bus must have a unique seven-bit address, and most I^2C devices have methods for configuring this address to allow several similar devices on the same bus. Many memory chips have a couple of I/Os which can be pulled high or low externally to configure the address.

The I/Os on an I^2C device are open-drain, and each I^2C bus requires an external pull up resistor on its lines. The value of the pull up resistors is not critical, and is typically in the 2-50 kOhm range. Low values will increase current consumption while the bus is active, but allow higher clock frequencies to be used. Inversely high resistor values will reduce the current consumption, but also the maximum clock frequency.

To initiate communication on the I^2C interface the master must issue a start condition, which is achieved by pulling SDA low while SCL stays high. Then data is clocked out on the SDA line, sampled at the rising edge of SCL, until the transmission is terminated by a stop condition. The stop condition is set by pulling SDA high while SCL stays high.

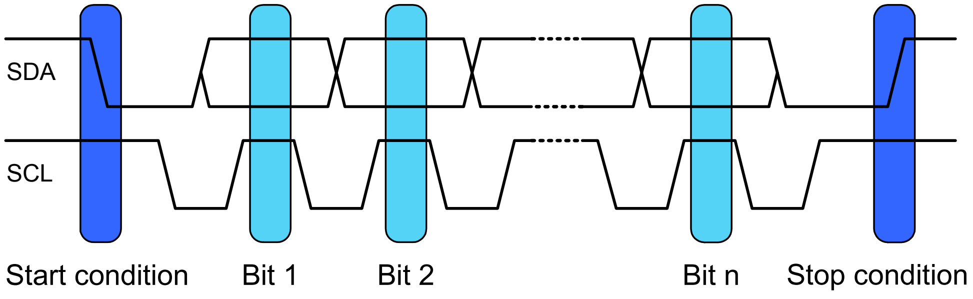

Figure 1. I^2C timing diagram

Immediately following the start condition the master will write the 7-bit slave address followed by a read/write bit (0 for write, 1 for read). The read/write bit tells the slave if the master wants to write to or read from it. The following byte(s) contains the actual data from the master (in case of a write command) or the slave (in case of a read command).

3.2 SPI

The SPI interface requires four I/Os for a single master/slave link. While an unlimited amount of slaves can be added to a single bus, the master needs an extra I/O for each added slave. This makes SPI impractical if many slaves are needed (each slave requires a dedicated Chip Select line).

SPI can usually run at much higher data rates than I^2C, and it is not uncommon for SPI devices to support bit rates in the 10 to 50 MBit/sec range. While bit rates for I^2C are standardized into six predefined values, an SPI device can have an arbitrary maximum bit rate.

An SPI master must always make sure that it uses a bit rate that the slave can support. SPI has less data overhead compared to I^2C, and supports full duplex communication (even though few memory chips make use of it).

SPI can be operated in a total of eight different configurations, depending on the state of the following three parameters:

1. Clock polarity: Active high or active low
2. Clock phase: Sample on leading or trailing edge of clock
3. Bit order: MSB to LSB, or LSB to MSB

Figure 2., Figure 3., Figure 4. and Figure 5. on page 6 show the SPI signals for all combinations of clock polarity and clock phase during transfer of a single byte. The bit order is MSB to LSB in all the figures.

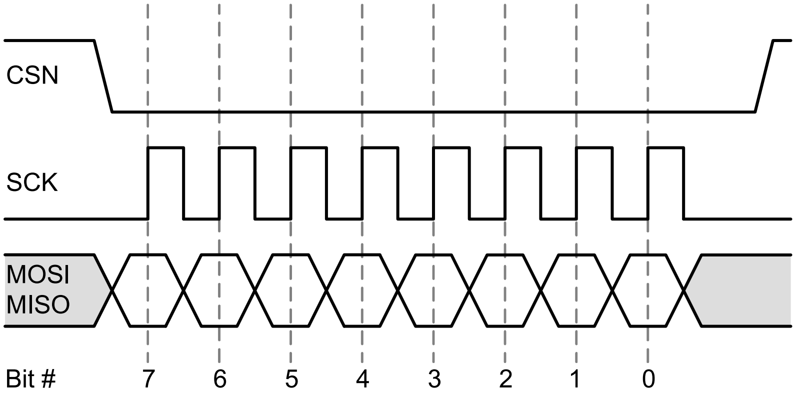

Figure 2. SCK active high, sample on leading edge, bit order MSB to LSB

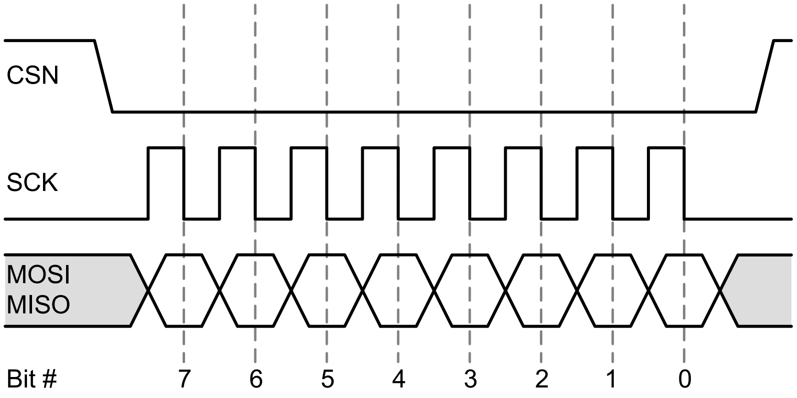

Figure 3. SCK active high, sample on trailing edge, bit order MSB to LSB

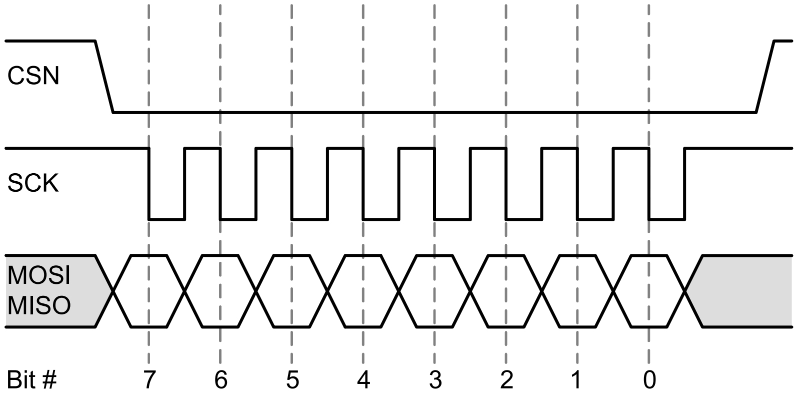

Figure 4. SCK active low, sample on leading edge, bit order MSB to LSB

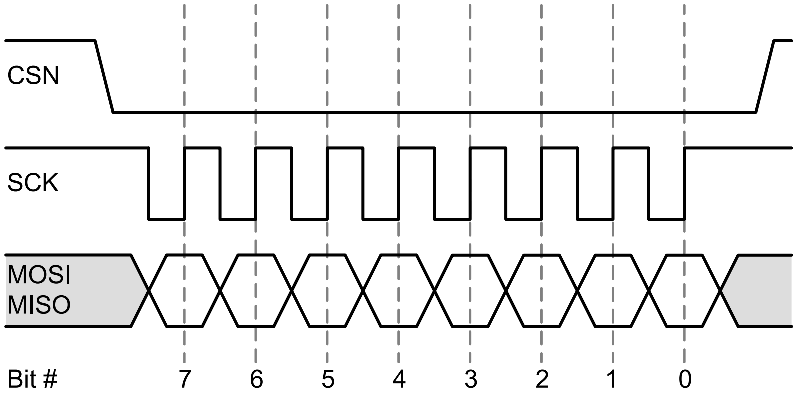

Figure 5. SCK active low, sample trailing edge, bit order MSB to LSB

4 Choosing the right memory module

The following bullet points list some factors to take into account when considering non-volatile memory:

- Price: How large a BOM can you accept?
- Capacity: What are your current and future data requirements?
- Data retention: How long does the data need to be stored and how many rewrites do you expect?
- Supply voltage: Make sure your memory module supports the supply voltage of your system.
- Current consumption: If current consumption is a concern, check the read/write/erase current and which power modes are available.
- I/O requirements: How many I/Os does your MCU have available?
- Speed/timing: What are your transfer speed and latency requirements?
- Package type: Does your hardware design put any limits on package type or size?

EEPROMs often use I^{2}C due to the low I/O requirement and the small amounts of data involved. The larger flash memory is more likely to use the faster SPI interface or faster and more I/O intensive parallel interfaces which will not be discussed in this document.

5 Hardware implementation

This section describes two scenarios for connecting memory modules to nRF24LE1. The first example describes connecting the M24C32 from ST Microelectronics, a 32-kbit EEPROM, using the I^{2}C interface. The second scenario describes connecting the AT25DF021 from Atmel, a 2Mbit Flash, using SPI.

Datasheets can be accessed from the websites of the respective vendors, at www.st.com and www.atmel.com.

Note: Memory devices like those mentioned in this section, usually rate sizes in kbit and Mbit, not kbyte and Mbyte.

5.1 I^{2}C EEPROM

Figure 6. on page 8 shows a connection between the nRF24LE1 32-pin and the M24C32. The WC signal is set high, disabling the write protect feature. By connecting the WL signal to GND, the EEPROM can be write-protected in hardware. By making the connection a solder point or switch, the hardware write-protection can be activated after a certain time (after the necessary data is written to the device).
E0, E1 and E2 are all set high, which sets the I^{2}C address of the memory module to 0x57.

Please note the pull-up resistors, Ra and Rb, which are required by the I^{2}C bus and cannot be omitted. In this case the resistor values are 10 kOhm, which represents a compromise between current consumption and maximum bus speed. It is possible to experiment with different values to improve one of the two, at the cost of the other.

The I^{2}C lines, SDA and SCL, are connected to their respective I/Os on the nRF24LE1. If a different package type is used, these signals will be located on different I/Os. See Table 1.

	nRF24LE1 24-pin	nRF24LE1 32-pin	nRF24LE1 48-pin
SCL	P0.5	P0.4	P1.2
SDA	P0.6	P0.5	P1.3

Table 1. I/Os for SDA and SCL

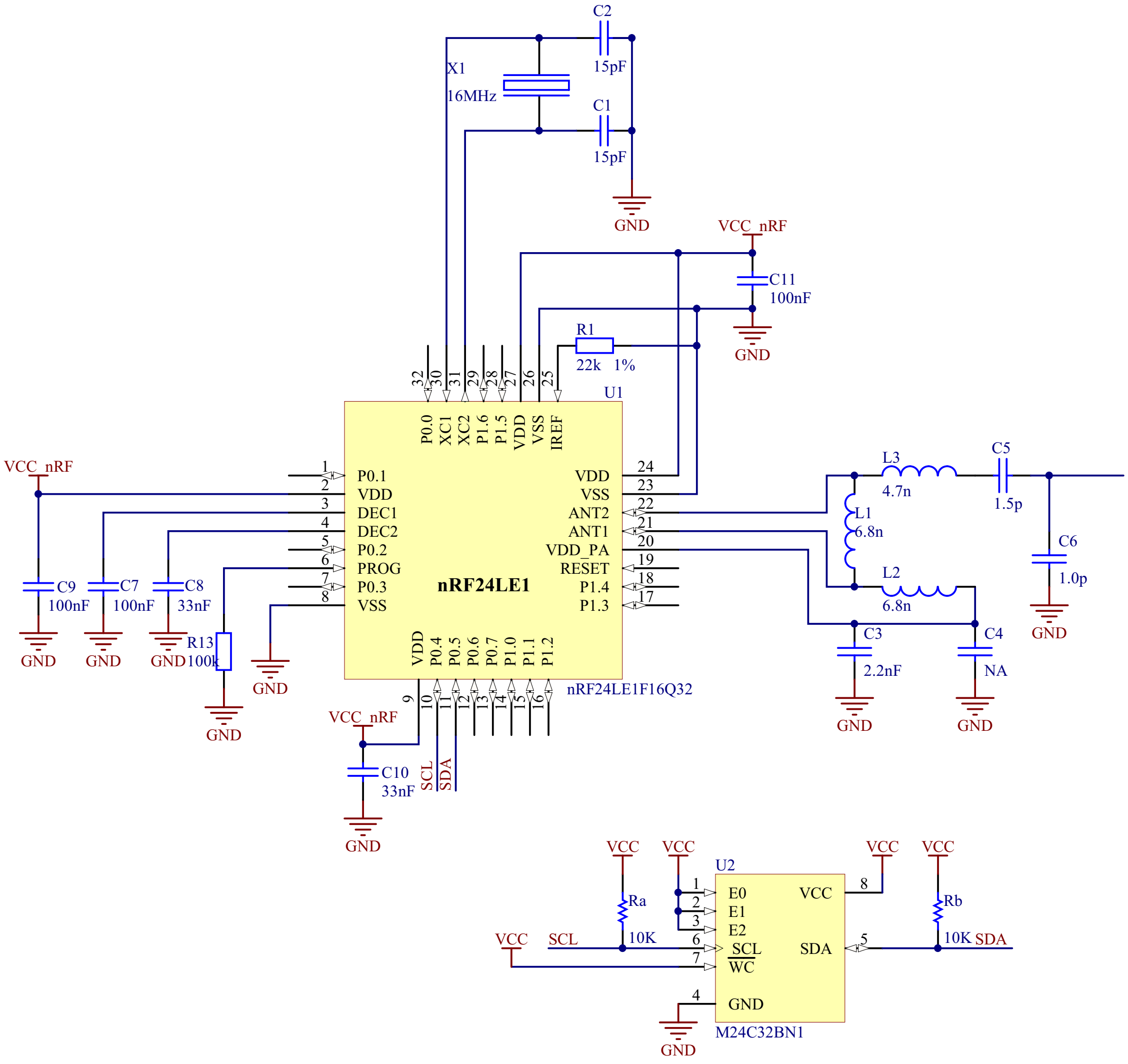

Figure 6. Schematic for nRF24LE1 with I^{2}C EEPROM

5.2 SPI flash

Figure 7. shows a possible connection between the nRF24LE1 32-pin and the AT25DF021. The WP and HOLD signals are set high, disabling these features. One or both of these can be connected to the nRF24LE1 to allow them to be controlled through the software.

The SPI lines, MOSI, MISO, MSCK and CSN, are connected to their respective I/Os on the nRF24LE1. The location of CSN can be decided by the user, and is set to P0.0 in this example. If a different package type is used, the other SPI signals will be located on the following I/Os:

Signal name	nRF24LE1 24-pin	nRF24LE1 32-pin	nRF24LE1 48-pin
MSCK	P0.2	P1.4	P0.6
MMOSI	P0.3	P1.5	P0.7
MMISO	P0.4	P1.6	P1.0

Table 2. I/Os for MSCK, MMOSI and MMISO

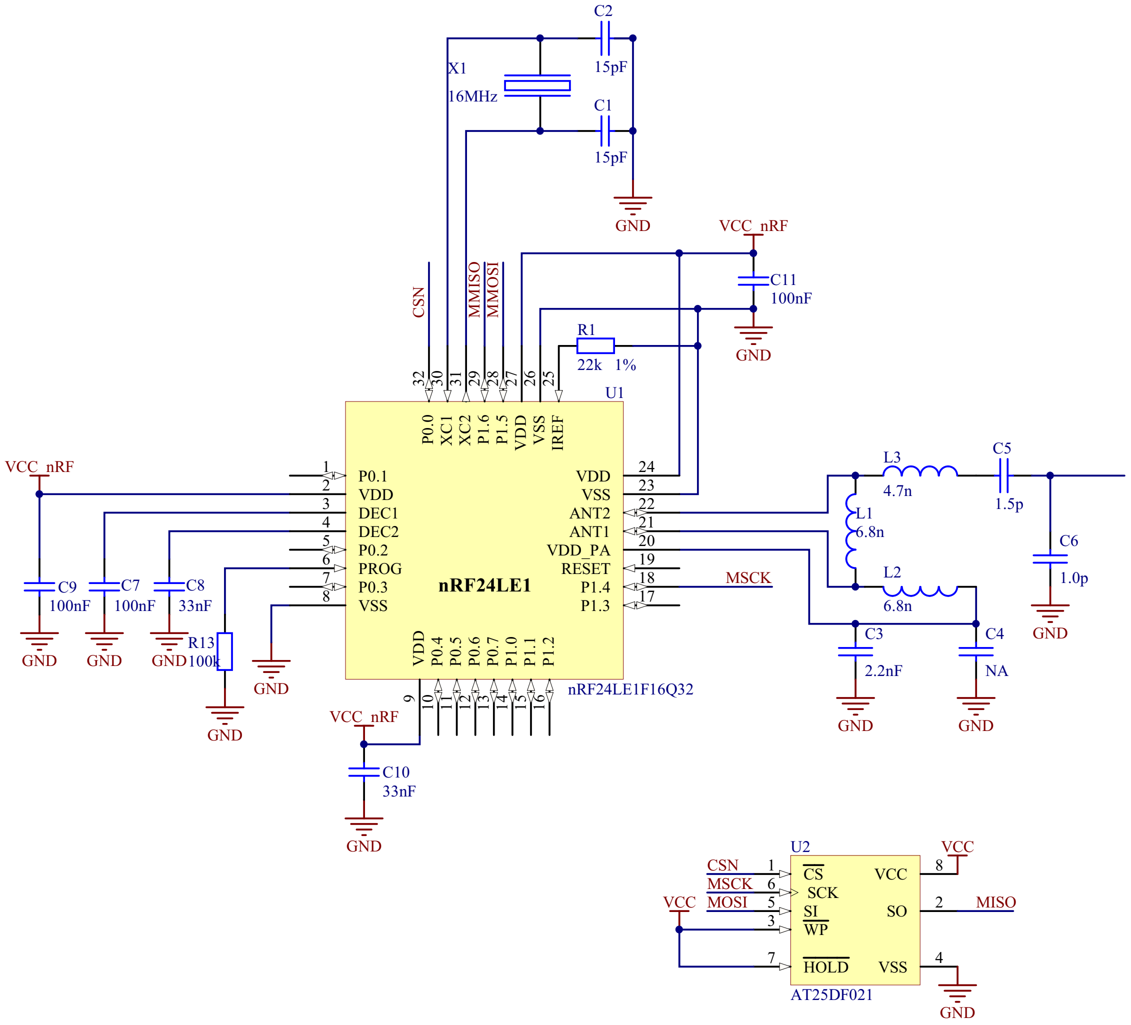

Figure 7. Schematic for nRF24LE1 with flash SPI

6 Software implementation

The software referred to in this document is intended to show a possible way to establish communication with the memory modules from the nRF24LE1. For both the I^2C EEPROM and the SPI flash, a set of library functions are provided to access the device, and a small application that implements these functions in a practical example.

6.1 I^2C EEPROM

The most important consideration when setting up an I^2C master is to verify that the I^2C slave supports the clock speed that the master is using. Both the M24C32 EEPROM and the nRF24LE1 support 400 kbit mode, and this is enabled in the nRF24LE1 firmware.

The I^2C master must then know the address of the I^2C slave with which it is trying to communicate. For the M24C32 the I^2C address is determined in part by the E0, E1 and E2 inputs, and in the chosen configuration (E0, E1 and E2 set high) the I^2C address is 0x57.

Both the supported I^2C configuration and the I^2C address are available from the device datasheet.

When the I^2C interface is configured, the easiest way to see if everything is working is to write the I^2C slave address to the I^2C bus and see if an ACK is received from the slave. The hal_w2_test_address() function in the example firmware can be used:

```
#define M24C32_I2C_ADDRESS 0x57

// Configure the 2-wire interface in 400 kHz mode.
hal_w2_configure_master(HAL_W2_400KHZ);

if( hal_w2_test_address(0x57) == 1 )
{
  // Ack received!
}
```

If an ACK is received, all is well. However if you get a NACK make sure that:

- The I^2C configuration chosen is supported by the device.
- You have chosen the correct I^2C slave address.
- The I^2C device is connected as shown in Figure 6. on page 8, and that that the pull ups are in place.

When you make contact with the device, you can start implementing higher-level functions for reading from and writing to the device. While there are some similarities between the command set of different memory chips, you should always read through the datasheet to see which commands exist and how the different commands must be structured. The M24C32 contains a relatively small set of commands and does not use any command bytes. Instead both read- and write operations are started by first writing a 16-bit address determining which part of the EEPROM to access, followed by data read/writes.

Byte 1		Byte 2	Byte 3	Byte 4	Byte 5	...	Byte (3+n)
I^{2}C-Address	W	EEPROM address (16-bit)		Data 1	Data 2	...	Data n

Table 3. *Up to 32 bytes can be written in a single command*

Byte 1		Byte 2	Byte 3	Byte 4		Byte 5	Byte 6	...	Byte (4+n)
I^{2}C-Address	W	EEPROM address (16-bit)		I^{2}C-Address	R	Data 1	Data 2	...	Data n

Table 4. Any number of bytes can be read in a single command

For an implementation of the functionality in the M24C32, see the attached **lib_m24c32.c** file.

6.2 SPI flash

When setting up an SPI interface it is important to check the maximum supported frequency of the slave, and which SPI configuration it supports. There are a total of eight different SPI configurations available, as the bit order, clock polarity and clock phase can differ from device to device.

The four different combinations of clock polarity and clock phase are usually defined as "modes" in SPI terminology, and the descriptions of these modes can be found in Table 5. By looking at the datasheet for the AT25DF021 the maximum SPI frequency is 33 MHz, one sees that the bit order is 'MSB to LSB', the clock polarity is 'active high', and the clock phase is 'sample on leading edge of MSCK'.

The maximum SPI frequency for the nRF24LE1 is 8 MHz (MCU clock / 2) which is well below the 33 MHz limit of the AT25DF021, and according to the device SPI settings the interface must be configured in HAL_SPI_MODE_0 (see Table 5. for a summary of the different SPI modes in the SPI library):

```
// Configure master SPI interface for compatibility with the AT25DF021
hal_spi_master_init(SPI_CLK_DIV2, HAL_SPI_MODE_0, HAL_SPI_MSB_LSB);
```

	Clock polarity	Clock phase
HAL_SPI_MODE_0	MSCK is active high	Sample on leading edge of MSCK
HAL_SPI_MODE_1	MSCK is active high	Sample on trailing edge of MSCK
HAL_SPI_MODE_2	MSCK is active low	Sample on leading edge of MSCK
HAL_SPI_MODE_3	MSCK is active low	Sample on trailing edge of MSCK

Table 5. Clock- phase and polarity for HAL_SPI_MODE

The SPI interface does not have any ACK functionality like I^{2}C, and the only way to know if the communication between the SPI master and slave is good is by running some commands and seeing if you get the response that you expect.

Most SPI devices have some status- or test registers that you can read to see if you have communication working. In the case of the AT25DF021 there is a command available that reads out a manufacturer- and device ID, and these values are constant for the device and defined in the AT25DF021 datasheet. A function is included in the supplied library to read this number, and the expected return bytes are: 0x1F, 0x43, 0x00, 0x00:

```
uint8_t id_string[4];
at25_read_manufacturer_and_device_id(id_string);
// Now id_string[] should contain 0x1F, 0x43, 0x00, 0x00
```

If you don't receive these bytes make sure of the following:

- The flash device is connected as described in Figure 7. on page 9.
- The SPI master interface on the nRF24LE1 is enabled and in the right mode.
- The SPI chip select pin (P00 in this example) is configured as an output, set high to default, and held low for each SPI command.
- You are not breaking any timing constraints of the chip. Some devices might require a certain delay after startup, or a certain delay after setting SPI chip select low, before starting to send data.

When you make contact with the device, you can start implementing higher-level functions for reading from and writing to the device. While there are some similarities between the command set of different memory chips, you should always read through the datasheet to see which commands exist and how the different commands must be structured. The AT25DF021 contains a fairly large set of commands allowing you to erase pages, protect/unprotect pages, enable/disable power down, read/write the status register and read/write data.

All commands start with a single command byte, and some require a 3-byte address to determine which part of the flash should be accessed. Compared to EEPROM devices most flash devices are more complex to use and require a more tedious programming procedure. For instance, data can only be erased page-wise, pages must be unprotected before they can be changed, and before each write/erase command the 'Write Enable' bit in the status register of the flash must be set.

Some of the most important commands are summarized in Table 6. to Table 11.. SPI is a full duplex interface, and data can flow both ways at any time. As such there is one row for the MOSI line (data from the nRF24LE1 to the flash), and one row for the MISO line (data from the flash to the nRF24LE1). 'xx' in the diagram means that the master must send a dummy byte to initiate the transfer, but the content of the dummy byte is irrelevant (don't care).

Signal name	Byte 1 (cmd)
MOSI	0x06/0x04
MISO	--

Table 6. Bytes for write enable/disable

Signal name	Byte 1 (cmd)	Byte 2	...	Byte n
MOSI	0x05	xx	...	xx
MISO	...	STATUS	...	STATUS

Table 7. Read status register

Signal name	Byte 1 (cmd)	Byte 2	Byte 3	Byte 4
MOSI	0x36/0x39	Flash address (24-bit)		
MISO	--	--	--	--

Table 8. Protect/unprotect sector

Signal name	Byte 1 (cmd)	Byte 2	Byte 3	Byte 4
MOSI	0x20	Flash address (24-bit)		
MISO	--	--	--	--

Table 9. Erase 4 kB block

Signal name	Byte 1 (cmd)	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	...	Byte (4+n)
MOSI	0x02	Flash address (24-bit)			Data 1	Data 2	...	Data n
MISO	--	--	--	--	--	--	...	--

Table 10. Up to 256 bytes can be written in a single write command

Signal name	Byte 1 (cmd)	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	...	Byte (4+n)
MOSI	0x03	Flash address (24-bit)			xx	xx	...	xx
MISO	--	--	--	--	Data 1	Data 2	...	Data n

Table 11. Any number of bytes can be read in a single read command

For an implementation of the functionality in the AT25DF021 look in the attached **lib_at25df021.c** file.

7 Example software

Two different software examples are included in this application note, showing a practical use case for the two referenced non-volatile memory devices.

Both software examples do the same thing, but since one targets an I^2C EEPROM device and one an SPI flash device, they go about it slightly differently.

The basic functionality of the examples is to generate a random serial number the first time the program is run and store it to the non-volatile memory device. Any subsequent run of the program will not generate a new serial number, but read the original one out from the non-volatile memory device. A specific byte in the non-volatile memory device is used to signify whether or not the serial number is present, and in the application this byte is set to 0xA3 after the serial number is programmed into the device.

At the end of the program, whether it is run for the first time or not, the serial number will be available in the 'serial_number' variable, and it is possible to use the examples as a basis for a larger application where a serial number is needed.

The software files can be found in the "code examples" folder in the application note's corresponding zip file, and Table 12. describes the files found in this folder.

File name	File type
AT25DF021/lib/lib_at25df021.h	SPI flash library header file
AT25DF021/lib/lib_at25df021.c	SPI flash library source code
AT25DF021/lib/Test project/ Flash_test.uvproj	SPI flash example project file
AT25DF021/lib/Test project/main.c	SPI flash example main file
M24C32/lib/hal_w2_mod.h	Modified I2C HAL header file[a]
M24C32/lib/hal_w2_mod.c	Modified I2C HAL source code[a]
M24C32/lib/lib_m24c32.h	I2C EEPROM library header file
M24C32/lib/lib_m24c32.c	I2C EEPROM library source code
M24C32/lib/Test project/ Eeprom_test.uvproj	I2C EEPROM example project file
M24C32/lib/Test project/main.c	I2C EEPROM example main file

[a] The existing I^2C library files (from the nRFgo SDK v2.1) were modified to simplify the I^2C EEPROM library, primarily by adding higher level functions for writing/reading over the I^2C bus.

Table 12. Software files in "code examples" folder

8 Glossary

Term	Description
BOM	Bill Of Materials. The list of required components for a hardware design. It is often associated with the cost of the design, as a smaller BOM usually means the total component cost is reduced.
CSN	Chip Select. The SPI chip select signal, used by the master to tell the slave that a command is about to start
EEPROM	Electronically Erasable Programable Read Only Memory. Byte-erasable, non-volatile memory
Flash	Page-erasable, non-volatile memory
HAL	Hardware Abstraction Layer. Set of basic library functions provided in Nordic Semiconductor's SDK for the nRF24LE1
I^2C	Inter-Integrated Circuit. Data bus used for low speed communication between computer peripherals. Another name for I^2C is 2-wire.
MISO	Master In Slave Out. The SPI signal used to send data from the SPI slave to the SPI master
MOSI	Master Out Slave In. The SPI signal used to send data from the SPI master to the SPI slave
MSCK	Master Serial Clock. The SPI clock, controlled by the SPI master
Open drain	Whereas a regular digital output pulls its signal low or high depending on the I/O state, an open-drain (or open collector) output will pull the signal low on a logical '0', and disconnect from the I/O on a logical '1'. Also known as tri-stating, this allows many devices to share the same signal, as an output in the logical '1' state will not interfere with other outputs on the same signal. Since an open-drain output is not able to pull the signal high, an external pull-up resistor is usually required on a line with only open-drain outputs connected (like the signal lines on an I^2C bus).
SPI	Serial Peripheral Interface. Data bus used for communication between computer peripherals. SPI is sometimes called 4-wire.

Table 13. Glossary

9 References

- Datasheets for MC24C32, AT25 and nRF24LE1
- http://www.robot-electronics.co.uk/htm/using the i2c bus.htm

Liability disclaimer

Life support applications

Nordic Semiconductor's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest dealer, please see http://www.nordicsemi.com.

Receive available updates automatically by subscribing to eNews from our homepage or check our website regularly for any available updates.

Main office:

Otto Nielsens veg 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.com

DNV
NORWEGIAN ACCREDITATION
No. S03
NS-EN ISO 9001 CERTIFICATED FIRM

Revision History

Date	Version	Description
January 2011	1.0	