

Technical Note

Software Device Drivers for Micron® N25Q Serial NOR Flash Memory

Introduction

This technical note provides a description of the library source code in C for Micron® N25Q serial NOR Flash memory devices, which use interfaces similar to the Flash software device driver interface specification. The N25Q.c and N25Q.h files contain libraries for accessing 128Mb, 256Mb, and 512Mb N25Q Flash memory devices.

This technical note describes the operation of the memory device and provides a basis for understanding and modifying the accompanying source code. The source code is written to be as platform independent as possible, and requires some changes by the user to compile and run.

The technical note also explains how the source code should be modified for individual target hardware. The source code contains comments throughout, which explain how it is used and why it has been written the way it has.

This technical note does not replace the N25Q data sheets. It refers to them throughout, and it is necessary to have a copy of them to follow some explanations. The software supplied with this documentation has been tested on a target platform and can be used in C and C++ environments. It is small in size and can be applied to any target hardware.

Programming Model

N25Q is a family of serial NOR Flash memory devices accessed by a high-speed serial peripheral interface (SPI) compatible bus. The memory can be written or programmed 1 to 256 bytes at a time.

Operating Features

N25Q devices memory can work in execute in place (XIP) mode. This means that the device only requires the addresses and not the instructions to output the data. This mode dramatically reduces random access time, thus enabling many applications requiring fast code execution. It is possible to enable XIP mode in two ways:

- Using the volatile configuration register, which is dedicated to applications that boot in SPI mode (extended SPI, DIO-SPI, or QIO-SPI) and then must switch to XIP mode while the application is running to directly execute code in the device.
- Using the nonvolatile configuration register, which is dedicated to applications that must boot directly in XIP mode.

N25Q memory devices can work with three different serial protocols:

- **Extended SPI protocol:** This is an extension of the standard (legacy) SPI protocol. Instructions are transmitted on a single data line, while addresses and data are transmitted by one, two, or four data lines according to the instruction.

- **Dual I/O SPI (DIO-SPI) protocol:** Instructions, addresses, and I/O data are always transmitted on two lines.
- **Quad SPI (QIO-SPI) protocol:** Instructions, addresses, and I/O data are always transmitted on four lines.

It is possible to choose among the three protocols by means of user volatile or nonvolatile configuration bits. It is not possible to mix Extended SPI, DIO-SPI, and QIO-SPI protocols. The device can operate in XIP mode in all three protocols.

The memory device has 64 one-time-programmable (OTP) bytes that can be read and programmed using two dedicated instructions: READ OTP (ROTP) and PROGRAM OTP (POTP), respectively. These 64 bytes can be permanently locked by a particular PROGRAM OTP sequence. Once they have been locked, they become read-only and this state cannot be reverted.

An internal PROGRAM/ERASE controller handles the timings necessary for PROGRAM and ERASE operations. The end of a PROGRAM or ERASE operation can be detected by polling the write in progress (WIP) bit in the status register.

Each device can be hardware protected (HPM) or software protected (SPM) against accidental PROGRAM and ERASE operations that could alter its contents. This is achieved by properly driving the write protect signal, nW/V_{PP} and properly setting the status register (legacy SPI status register).

The driver code always assumes that no protection has been applied to the target memory array when attempts to PROGRAM or ERASE are made. Protection attempts to modify the target memory array should therefore be carefully managed by the final application.

Bus Operations and Commands

Most functionality in N25Q devices is available via the instruction set. READ operations retrieve data or status information from the device. Some other operations modify data or the device behavior.

The various instructions recognized by the device are listed in the Instruction Set Table provided in the corresponding data sheet. The instructions are described in Table 1:

Table 1: Commands

Command	Description
BULK ERASE (BE)	Sets all bits to 1 (FFh). The BULK ERASE (BE) instruction is executed only if all block protect bits (TB, BP2, BP1, BP0) are 0. The BULK ERASE (BE) instruction is ignored if one or more sectors are protected.
CLEAR FLAG STATUS REGISTER (CLFSR)	Resets the error flag status register bits (erase error bit, program error bit, V _{PP} error bit, and protection error bit). It is not necessary to set the WEL bit before the CLEAR FLAG STATUS REGISTER instruction is executed.
DEEP POWER-DOWN (DP)	Executing the DEEP POWER-DOWN (DP) instruction is the only way to put the device in the lowest consumption mode (the deep power-down mode). It can also be used as a software protection mechanism while the device is not in active use. In this mode, the device ignores all WRITE, PROGRAM, and ERASE instructions.
DUAL INPUT EXTENDED FAST PROGRAM (DIEFP)	This is similar to the DUAL INPUT FAST PROGRAM (DIFFP), except that the address bits are shifted in on two pins (pin DQ0 and pin DQ1) instead of only one pin.

Table 1: Commands (Continued)

Command	Description
DUAL INPUT FAST PROGRAM (DIFP)	This is similar to the PAGE PROGRAM (PP) instruction, except that the data is entered on two pins (pin DQ0 and pin DQ1) instead of only one. Inputting the data on two pins instead of one doubles the data transfer bandwidth when compared to the PAGE PROGRAM (PP) instruction.
DUAL INPUT/OUTPUT FAST READ (DIOFR)	This is similar to the DUAL OUTPUT FAST READ (DOFR), except that the address bits are shifted in on two pins (pin DQ0 and pin DQ1) instead of only one pin.
DUAL OUTPUT FAST READ (DOFR)	This is similar to the READ DATA BYTES AT HIGHER SPEED (FAST_READ) instruction, except that the data is shifted out on two pins (pin DQ0 and pin DQ1) instead of only one. Outputting the data on two pins instead of one doubles the data transfer bandwidth compared to the READ DATA BYTES AT HIGHER SPEED (FAST_READ) instruction.
ENTER 4-BYTE ADDRESS MODE	Enters the 4-byte address mode.
EXIT 4-BYTE ADDRESS MODE	Exits the 4-byte address mode
PAGE PROGRAM (PP)	Enables bytes to be programmed in the memory (changing bits from 1 to 0).
PROGRAM OTP (PROGRAM 64 BYTES OF OTP AREA) (POTP)	Used to program at most 64 bytes to the OTP memory area (by changing bits from 1 to 0 only). To lock the OTP memory: Bit 0 of the OTP control byte (byte 64) is used to permanently lock the OTP memory array. When bit 0 of byte 64 is equal to 1, the 64 bytes of the OTP memory array can be programmed. When bit 0 of byte 64 is equal to 0, the 64 bytes of the OTP memory array are read-only and cannot be programmed. Once a bit of the OTP memory has been programmed to 0, it can no longer be set to 1. Therefore, as soon as bit 0 of byte 64 (control byte) is set to 0, the 64 bytes of the OTP memory array become permanently read-only.
PROGRAM/ERASE RESUME	Resumes the suspended PROGRAM or ERASE operation to resume after a PROGRAM/ERASE SUSPEND instruction.
PROGRAM/ERASE SUSPEND	Interrupts a PROGRAM or ERASE instruction. By design, bulk erase and program OTP cannot be suspended.
QUAD INPUT EXTENDED FAST PROGRAM (QIEFP)	This is similar to the QUAD INPUT FAST PROGRAM (QIFP) instruction, except that the address bits are shifted in on four pins (pin DQ0, pin DQ1, pin W/VPP/DQ2, and pin HOLD/DQ3) instead of only one pin.
QUAD INPUT FAST PROGRAM (QIFP)	This is similar to the DUAL INPUT FAST PROGRAM (DIFP) instruction, except that the data is entered on four pins (pin DQ0, pin DQ1, pin W/VPP/DQ2 and pin HOLD/DQ3) instead of only two. Inputting the data on four pins instead of two doubles the data transfer bandwidth compared to the DUAL INPUT FAST PROGRAM (DIFP) instruction.
QUAD INPUT/OUTPUT FAST READ (QIOFR)	This is similar to the QUAD OUTPUT FAST READ (QOFR) instruction, except that the address bits are shifted in on four pins (pin DQ0, pin DQ1, pin W/VPP/DQ2, and pin HOLD/DQ3) instead of only one.
QUAD OUTPUT FAST READ (QOFR)	This is similar to the DUAL OUTPUT FAST READ (DOFR) instruction, except that the data is shifted out on four pins (pin DQ0, pin DQ1, pin W/VPP/DQ2, and pin HOLD/DQ3 [1]) instead of only two pins. Outputting the data on four pins instead of two doubles the data transfer bandwidth when compared to the DUAL OUTPUT FAST READ (DOFR) instruction.
READ DATA BYTES (READ)	Outputs the data stored at the specified device addresses.
READ DATA BYTES AT HIGHER SPEED (FAST_READ)	Outputs the data stored at the specified device addresses at a higher speed.
READ FLAG STATUS REGISTER (RFSR)	Enables the flag status register to be read.

Table 1: Commands (Continued)

Command	Description
READ IDENTIFICATION (RDID)	Enables the device identification data to be read: manufacturer identification (1 byte), device identification (2 bytes), and a unique ID code (UID) (17 bytes, of which 16 are available upon customer request). The manufacturer identification is assigned by JEDEC and has the value 20h for Micron. The device identification is assigned by the device manufacturer and indicates the memory type in the first byte (BAh) and the memory capacity of the device in the second byte (18h).
READ LOCK REGISTER (RDLR)	Enables the lock register to be read.
READ NONVOLATILE CONFIGURATION REGISTER (RDNVCR)	Enables the nonvolatile configuration register to be read.
READ OTP (READ 64 BYTES OF OTP AREA) (ROTP)	The READ OTP (ROTP) instruction outputs the data at the OTP area.
READ STATUS REGISTER (RDSR)	Enables the status register to be read.
READ VOLATILE CONFIGURATION REGISTER (RDVCR)	Enables the volatile configuration register to be read.
READ VOLATILE ENHANCED CONFIGURATION REGISTER (RDVECR)	Enables the volatile configuration register to be read.
RELEASE FROM DEEP POWER-DOWN (RDP)	Once the device has entered the deep power-down mode, all instructions are ignored except the RELEASE FROM DEEP POWER-DOWN (RDP) instruction. Executing this instruction takes the device out of the deep power-down mode.
SECTOR ERASE (SE)	Sets to 1 (FFh) all bits inside the chosen sector. A SECTOR ERASE (SE) instruction applied to a page that is protected by the block protect bits (TB, BP2, BP1, BP0) is not executed.
SUB-SECTOR ERASE (SSE)	Sets to 1 (FFh) all bits inside the chosen sub-sector. A SUB-SECTOR ERASE (SSE) instruction issued to a sector that is hardware or software protected is not executed. Any SUB-SECTOR ERASE (SSE) instruction while an ERASE, PROGRAM, or WRITE cycle is in progress is rejected without affecting the cycle that is in progress.
WRITE DISABLE (WRDI)	Resets the write enable latch (WEL) bit. The WRITE DISABLE (WRDI) instruction is entered by driving chip select (S) LOW, sending the instruction code, and then driving chip select (S) HIGH.
WRITE ENABLE (WREN)	Sets the write enable latch (WEL) bit. The WEL bit must be set prior to each PAGE PROGRAM (PP), DUAL INPUT FAST PROGRAM (DIFP), PROGRAM OTP (POTP), WRITE TO LOCK REGISTER (WRLR), SUBSECTOR ERASE (SSE), SECTOR ERASE (SE), BULK ERASE (BE), and WRITE STATUS REGISTER (WRSR) instruction.
WRITE NONVOLATILE CONFIGURATION REGISTER (WRNVCR)	Enables new values to be written to the nonvolatile configuration register. Before it can be accepted, a WRITE ENABLE (WREN) instruction must have previously been executed. After the WRITE ENABLE (WREN) instruction has been decoded and executed, the device sets the write enable latch (WEL).
WRITE STATUS REGISTER (WRSR)	Enables new values to be written to the status register. The WRITE STATUS REGISTER (WRSR) instruction enables the user to change the values of the block protect bits (TB, BP2, BP1, BP0) and to define the size of the area that is to be treated as read-only. The WRITE STATUS REGISTER (WRSR) instruction also enables the user to set and reset the status register write disable (SRWD) bit in accordance with the write protect (nW/V _{pp}) signal. The status register write disable (SRWD) bit and write protect (nW/V _{pp}) signal enable the device to be put in the hardware protected mode (HPM). The WRITE STATUS REGISTER (WRSR) instruction is not executed once the hardware protected mode (HPM) is entered.

Table 1: Commands (Continued)

Command	Description
WRITE TO LOCK REGISTER (WRLR)	Enables bits to be changed in the lock registers.
WRITE VOLATILE CONFIGURATION REGISTER (WRVCR)	Enables new values to be written to the volatile configuration register. Before it can be accepted, a write enable (WREN) instruction must previously have been executed. After the write enable (WREN) instruction has been decoded and executed, the device sets the write enable latch (WEL).
WRITE VOLATILE ENHANCED CONFIGURATION REGISTER (WRVECR)	Enables new values to be written to the volatile enhanced configuration register. Before it can be accepted, a WRITE ENABLE (WREN) instruction must previously have been executed. After the WRITE ENABLE (WREN) instruction has been decoded and executed, the device sets the write enable latch (WEL).

Status Register

During PROGRAM or ERASE operations, a READ STATUS REGISTER (RDSR) instruction outputs the contents of the status register, which provides valuable information about the status of the ongoing operation. The status register bits are described in the corresponding data sheet. They are mainly used to determine when programming or erasing is complete and whether the operation was successful. The status bits of the status register are described in Table 2:

Table 2: Status Register Status Bits

Status Register bit	Description
Lock Protect (BPx) bits	These nonvolatile bits determine the size of the area to be software-protected against PROGRAM and ERASE instructions. The BPx bits are written with the WRITE STATUS REGISTER (WRSR) instruction, provided that the hardware protected mode has not been set. Depending on the value of the BPx bits, the corresponding memory area (as defined in the data sheet.) becomes protected against the PAGE PROGRAM (PP) and SECTOR ERASE (SE) instructions. Prior to executing the BULK ERASE (BE) instruction, it is required to reset all BPx bits to 0. If not, the BULK ERASE (BE) instruction will not be executed.
Status Register Write Disable (SRWD) bit	This is operated in conjunction with the write protect (W) signal. The SRWD bit and write protect (W) signal enable the device to be put in the hardware protected mode (when the SRWD bit is set).
Top/Bottom (TB) bit	This nonvolatile bit can be set and reset with the WRITE STATUS REGISTER (WRSR) instruction, provided that the WRITE ENABLE (WREN) instruction has been issued. The TB bit is used in conjunction with the block protect bits (BP0, BP1, and BP2) to determine if the protected area defined by the block protect bits starts from the top or the bottom of the memory array.
Write Enable Latch (WEL) bit	Indicates the status of the internal write enable latch (WEL). When set to 1, the internal WEL is set. When set to 0, the internal WEL is reset and no WRITE, PROGRAM, or ERASE instruction is accepted.

Table 2: Status Register Status Bits (Continued)

Status Register bit	Description
Write in Progress (WIP) bit	Indicates whether the memory is busy with a WRITE, PROGRAM, or ERASE cycle. When set to 1, such a cycle is in progress. When reset to 0, no such cycle is in progress.

Lock Register

There are two software protected modes (SPM1 and SPM2) that can be combined to protect the memory array as required. The first software protected mode (SPM1) is managed by specific lock registers assigned to each 64KB sector. The lock registers can be read and written using the READ LOCK REGISTER (RDLR) and WRITE TO LOCK REGISTER (WRLR) instructions. In each lock register, the bits described in Table 3 control the protection of each sector:

Table 3: Lock Register Bits

Lock Register Bit	Description
Lock Down bit	Provides a mechanism for protecting software data from simple hacking and malicious attacks. When the lock down bit is set to 1, further modification to the write lock and lock down bits cannot be performed. A power-up is required before changes to these bits can be made. When the lock down bit is reset to 0, the write lock and lock down bits can be changed.
Write Lock bit	Determines whether the contents of the sector can be modified using the WRITE, PROGRAM, or ERASE instructions. When the write lock bit is set to 1, the sector is write protected and any operations that attempt to change the data in the sector will fail. When the write lock bit is reset to 0, the sector is not write protected by the lock register and may be modified.

Nonvolatile Configuration Register

The nonvolatile configuration register (NVCR) bits affect the default memory configuration after powering on. It can be used to make the memory start in the configuration to fit the application requirements. The purpose of the NVCR is to define the default memory settings after the sequence of powering on.

Table 4: Nonvolatile Configuration Register Bits

NVCR bit	Description
Address byte (NVCR<0>)	Defines the number of address bytes (256Mb N25Q devices only).
Segment selection (NVCR<1>)	Selects an upper or lower segment (256Mb N25Q devices only).
Dual Input command bit (NVCR<2>)	Enables the DIO SPI protocol.
Quad Input command bit (NVCR<3>)	Enables the QUAD SPI protocol.
Reset/Hold disable bit (NVCR<4>)	Used to disable the hold/reset functionality. Setting the bit to 0 disables this functionality. Reset functionality is available instead of hold in devices with a dedicated part number.
Fast POR read bit (NVCR<5>)	Enables/disables the POR phase (disable is the default value).

Table 4: Nonvolatile Configuration Register Bits (Continued)

NVCR bit	Description
Output Driver Strength bits (NVCR<8:6>)	Used to set impedance at $V_{CC}/2$.
XIP enabling bit (NVCR<11:9>)	Enables XIP.
Dummy Clock Cycles bit (NVCR<15:12>)	Used to optimize instruction execution according to the clock frequency.

Volatile Configuration Register

The volatile configuration register (VCR) affects the memory configuration after every execution of a WRITE CONFIGURATION REGISTER instruction.

Table 5: Volatile Configuration Register Bits

VCR bit	Description
Dummy Clock Cycles bit (VCR<7:4>)	Used to optimize instruction execution according to the clock frequency.
Other bits (VCR<3>)	Reserved, fixed value 0.
Wrap mode (VCR<1:0>)	16, 32, 64, sequential.
XIP enabling bit (VCR<3>)	Enables XIP.

Extended Address Register

The extended address register enables access to memory beyond 128Mb. Refer to the N25Q data sheet for more information.

Table 6: Volatile Configuration Register Bits

Extended Address Register Bit	Description
<7:1>	Reserved.
<0>	EAR<0> = ADDR<24>. If EAR<0> = 1, the upper 128Mb is selected. If EAR<0> = 0, the lower 128Mb is selected.

Volatile Enhanced Configuration Register

The volatile enhanced configuration register (VECR) affects the memory configuration after each execution of the WRITE VOLATILE ENHANCED CONFIGURATION REGISTER (WRVECR) instruction. This instruction overwrites the memory configuration set during the POR sequence by the nonvolatile configuration register (NVCR).

Table 7: Volatile Enhanced Configuration Register Bits

VECR bit	Description
Dual I/O protocol (VECR<5>)	Enables or disables dual I/O protocol.
Dual Input command bit (VECR <6>)	Enables/disables the DIO SPI protocol.
Other bits (VECR <5>)	Reserved.
Output driver strength <2:0>	Output driver strength. Refer to the N25Q data sheet for more information.
Output Driver Strength bits (VECR <2:0>)	Used to set impedance at $V_{CC}/2$.

Table 7: Volatile Enhanced Configuration Register Bits (Continued)

VECR bit	Description
QIO-SPI (VECR <3>)	Used to determine whether it is possible to use the V _{pp} accelerating voltage to speed up an internal modify operation with quad PROGRAM and ERASE instructions.
Quad Input command bit (VECR<7>)	Enables/disables the QUAD SPI protocol.
Reset/Hold enable/disable bit (VECR<4>)	Enables/disables the pad hold/reset functionality.
V _{pp} accelerator (VECR<3>)	Enables or disables V _{pp} acceleration for QUAD INPUT FAST PROGRAM and INPUT EXTENDED FAST PROGRAM operations.

Flag Status Register

The flag Status register is a tool for investigating the status of the device. It is used to check information regarding what is actually done to the memory and detect possible error conditions.

Table 8: Flag Status Register Bits

Bit	Description	Note
7	P/E Controller (not WIP)	Status
6	ERASE Suspend	Status
5	ERASE	Error
4	PROGRAM	Error
3	V _{pp}	Error
2	PROGRAM Suspend	Status
1	Protection	Error
0	RESERVED	

Using the Software Driver

General Considerations

The software driver described in this technical note is intended to simplify the process of developing application code in C. This software driver is based on the Flash device driver interface that is implemented in all new software drivers. As a result, future device changes will not necessarily lead to code changes in application environments.

With the software driver interface, users can focus on writing the high-level code required for their particular applications. The high-level code accesses the Flash memory device by calling the low-level code. This means that users do not have to concern themselves with the details of the special instruction sequences. The resulting source code is both simpler and easier to maintain.

Code developed using the provided drivers can be broken down into three layers:

1. Hardware-specific bus operations
2. Low-level code
3. High-level code written by the user

The low-level code requires hardware-specific register READ and WRITE operations in C to communicate with the N25Q device. The implementation of these operations is hardware-platform dependent as it depends on the microprocessor and micro controller on which the C code runs and on the location of the memory in the microprocessor's address space. The user must write the C code that is suitable for the current hardware platform. However, a guiding framework is provided in the `Serialize.h` and `Serialize.c` files.

The high-level code written by the user accesses the Flash memory devices by calling the low-level code. In this way, the code used is simple and easy to maintain. Another consequence is that the user's high-level code is easier to apply to other Micron serial NOR Flash memory devices.

When developing an application:

1. Write a simple program to test the low-level code provided, and verify that it operates as expected in the target hardware and software environments.
2. Write the high-level code for the desired application. The application will access the serial NOR Flash memory device by calling the low-level code.
3. Thoroughly test the complete source code of the application.

Porting the Driver (User Change Area)

All sensible changes to the software driver that the user must consider can be found in the header file. A designated area called the user change area contains the items described in the following sections, which are required to port the software driver to new hardware.

Basic Data Types

Check whether the compiler to be used supports the following basic data types, as described in the source code, and change it where necessary.

```
typedef unsigned char NMX_uint8; (8 bits)
typedef char NMX_sint8; (8 bits)
typedef unsigned short NMX_uint16; (16 bits)
typedef short NMX_sint16; (16 bits)
typedef unsigned int NMX_uint32; (32 bits)
typedef int NMX_sint32; (32 bits)
```

Device Type

The correct device is automatically detected at run time based on READ ID commands. This detection occurs during the Driver_Init() function. No defines are necessary to use either a 128Mb, 256Mb, or 512Mb N25Q device.

Timeout

Timeouts are implemented in the loops of the code to provide an exit to operations that would otherwise never terminate. There are two possibilities:

1. The ANSI library functions declared in time.h exist. If the current compiler supports time.h, the define statement TIME_H_EXISTS should be activated. This prevents any change in timeout settings due to the performance of the current evaluation hardware.

```
#define TIME_H_EXISTS
```

2. The option COUNT_FOR_A_SECOND. If the current compiler does not support time.h, the define statement TIME_H_EXISTS cannot be used. In this case, the COUNT_FOR_A_SECOND value must be defined so as to create a 1-second delay. For example, if 100,000 repetitions of a loop are needed to give a time delay of 1 second, then COUNT_FOR_A_SECOND should have the value 100,000.

```
#define COUNT_FOR_A_SECOND (chosen value)
```

Note: This delay depends on the hardware performance and should therefore be updated each time the hardware is changed.

This driver has been tested with a certain configuration and other target platforms may have other performance data. It may therefore be necessary to change the COUNT_FOR_A_SECOND value. It is up to the user to implement the correct value to prevent the code from timing out too early and enable correct completion. In accordance to the corresponding data sheet, a suitable timeout value is configured in each function where required.

Additional Subroutines

```
#define VERBOSE
```

In the software driver, the define VERBOSE statement is used to activate the Flash-ErrStr() function in order to generate a text string describing the return code from the Flash memory device.

C Library Functions

The software library described in this technical note provides the user with source code for the following functions:

Table 9. C Library Functions

Function	Description
Driver_Init()	Initializes the driver and automatically detects the device. This function should be called before all other functions. If the function returns the Flash_WrongType value, the device has not been recognized. For an example, see "Getting Started (Example Quick Test)" on page 14.
BulkErase()	Erases the entire memory by sending a BULK ERASE (BE) instruction.
DeepPowerDown()	Sets the device in the lowest power consumption mode (the deep power-down mode) by sending a DEEP POWER-DOWN (DP) instruction. After calling this routine, the Flash memory device will not respond to any instruction except the RELEASE FROM DEEP POWER-DOWN (RDP) instruction.
DualInputFastProgram()	Programs 256 bytes of data or less to the memory on two pins (pin DQ0 and pin DQ1) instead of only one by sending a DUAL INPUT FAST PROGRAM (DIFP) instruction.
DualOutputFastRead()	Reads data from the memory on two pins (pin DQ0 and pin DQ1) instead of only one by sending a DUAL OUTPUT FAST READ (DOFR) instruction.
Enter4ByteAddressMode()	The ENTER 4 BYTE ADDRESS MODE command enables the 4-byte address mode (256Mb N25Q devices only).
Exit4ByteAddressMode()	The EXIT 4 BYTE ADDRESS MODE command disables the 4-byte address mode (256Mb N25Q devices only).
FastRead()	Reads the Flash memory by sending a READ DATA BYTES AT HIGHER SPEED (FAST_READ) instruction. By design, the entire Flash memory space can be read with one FAST_READ instruction by incrementing the start address and rolling to 0h automatically. This means that this function is across pages and sectors.
FlashReadDeviceIdentification()	Reads the manufacturer identification (20h) and device identification by sending a READ IDENTIFICATION (RDID) instruction. Note that the last two instructions can be unified into a single function due to the fact that only a single instruction returns the device and manufacturer identification.
PageProgram()	Programs 256 bytes or less of data to the memory by sending a PAGE PROGRAM (PP) instruction. By design, the PP instruction is effective within one page, that is XX00h to XXFFh. When XXFFh is reached, the address rolls over to XX00h automatically. This function assumes that the memory to be programmed has been previously erased, or that bits are only changed from 1 to 0.
Program()	Programs a chunk of data into the memory at one time. After successfully verifying the start address and checking the available space, this function programs data from the buffer to the memory sequentially by invoking FlashPageProgram(). This function automatically handles page boundary crosses, if any. Similar to FlashPageProgram(), this function assumes that the memory to be programmed has been previously erased, or that bits are only changed from 1 to 0.

Table 9. C Library Functions (Continued)

Function	Description
ProgramEraseResume()	Resumes the PROGRAM/ERASE operation that was suspended by sending a SPI_FLASH_INS_PER instruction.
ProgramEraseSuspend()	Resumes the PROGRAM/ERASE operation that was suspended by sending a SPI_FLASH_INS_PES Instruction
ProgramOTP()	Programs the 64-byte OTP area by sending a PROGRAM OTP (POTP) instruction.
QuadInputExtendedFastProgram()	Programs 256 bytes or less of data and address to the memory on four pins by sending a QUAD INPUT FAST PROGRAM (QIFP) instruction. Similar to FlashPageProgram(), this function assumes that the memory to be programmed has been previously erased, or that bits are only changed from 1 to 0.
QuadInputFastProgram()	Programs 256 bytes or less of data to the memory on four pins by sending a QUAD INPUT FAST PROGRAM (QIFP) instruction. Similar to FlashPageProgram(), this function assumes that the memory to be programmed has been previously erased, or that bits are only changed from 1 to 0.
QuadInputOutputFastRead()	Reads data from the memory and send addresses on four pins instead of only one by sending a QUAD OUTPUT FAST READ (QIOFR) instruction.
QuadOutputFastRead()	Reads data from the memory on four pins instead of only one by sending a QUAD OUTPUT FAST READ (QOFR) instruction.
Read()	Reads the Flash memory by sending a READ DATA BYTES (READ) instruction. By design, the entire Flash memory space can be read with one READ instruction by incrementing the start address and rolling to 0h automatically. This means that this function is across pages and sectors.
ReadLockRegister()	Reads the status register by sending a READ LOCK REGISTER (RDLR) instruction.
ReadOTP()	Reads data from the OTP area by sending a READ OTP (ROTP) instruction.
ReadStatusRegister()	Reads the status register by sending a READ STATUS REGISTER (RDSR) instruction.
ReadVolatileConfigurationRegister()	The READ VOLATILE CONFIGURATION REGISTER (RDVCR) instruction enables the volatile configuration register to be read.
ReleaseFromDeepPowerDown()	Takes the device out of deep power-down mode by sending a RELEASE FROM DEEP POWER-DOWN (RDP) instruction.
SectorErase()	Erases an entire sector by sending a SECTOR ERASE (SE) instruction.
SubSectorErase()	Erases a sub-sector by sending a SUB-SECTOR ERASE (SSE) instruction.
WriteDisable()	Resets the WEL bit by sending a WRITE DISABLE (WRDI) instruction.
WriteEnable()	Sets the WEL bit by sending a WRITE ENABLE (WREN) instruction.
WriteLockRegister()	Writes the lock register by sending a WRITE LOCK REGISTER (WRLR) instruction.
WriteStatusRegister()	Writes the status register by sending a WRITE STATUS REGISTER (WRSR) instruction.
WriteVolatileConfigurationRegister()	The WRITE VOLATILE CONFIGURATION REGISTER (WRVCR) instruction enables new values to be written to the volatile configuration register. Before it can be accepted, a WREN instruction must previously have been executed.

READ VOLATILE ENHANCED CONFIGURATION REGISTER

The READ VOLATILE ENHANCED CONFIGURATION REGISTER (RDVECR) instruction enables the volatile configuration register to be read.

WRITE VOLATILE ENHANCED CONFIGURATION REGISTER

The WRITE VOLATILE ENHANCED CONFIGURATION REGISTER (WRVECR) instruction enables new values to be written to the volatile enhanced configuration register. Before it can be accepted, a WREN instruction must previously have been executed. After the WREN instruction has been decoded and executed, the device sets the write enable latch (WEL).

WRITE NONVOLATILE CONFIGURATION REGISTER

The WRITE NONVOLATILE CONFIGURATION REGISTER (WRNVCR) instruction enables new values to be written to the nonvolatile configuration register. Before it can be accepted, a WREN instruction must previously have been executed. After the WREN instruction has been decoded and executed, the device sets the write enable latch (WEL).

READ NONVOLATILE CONFIGURATION REGISTER

The READ NONVOLATILE CONFIGURATION REGISTER (RDNVCR) instruction enables the nonvolatile configuration register to be read.

CLEAR FLAG STATUS REGISTER

The CLEAR FLAG STATUS REGISTER (CLFSR) instruction resets the error flag status register bits (erase error bit, program error bit, V_{PP} error bit, protection error bit). It is not necessary to set the WEL bit before the CLEAR FLAG STATUS REGISTER instruction is executed.

Getting Started (Example Quick Test)

The following source code shows how to use the driver. It performs the three basic Flash operations: ERASE, PROGRAM, and READ.

```
#include <stdio.h>
#include "N25Q.h"
#include "Serialize.h"

int main(int argc, char ** argv)
{
    /* flash device object */
    FLASH_DEVICE_OBJECT fdo;
    /* parameters used for all operation */
    ParameterType para;
    /* return variable */
    Returntype ret;
    /* read buffer */
    NMX_uint8 rbuffer[16];
    /* write buffer */
    NMX_uint8 wbuffer[16] =
        { 0xBE, 0xEF, 0xFE, 0xED, 0xBE, 0xEF, 0xFE, 0xED,
          0xBE, 0xEF, 0xFE, 0xED, 0xBE, 0xEF, 0xFE, 0xED };

    /* initialize your SPI interface */
    SpiDriverInit();

    /* initialize the flash driver */
    ret = Driver_Init(&fdo);
    if (Flash_WrongType == ret)
    {
        printf("Sorry, no device detected.\n");
        return -1;
    }

    /* erase first sector */
    fdo.GenOp.SectorErase(0);

    /* program 16 byte at address 0 */
```

```
para.PageProgram.udAddr = 0;
para.PageProgram.pArray = wbuffer;
para.PageProgram.udNrOfElementsInArray = 16;
fdo.GenOp.DataProgram(PageProgram, &para);

/* read 16 byte at address 0 */
para.Read.udAddr = 0;
para.Read.pArray = rbuffer;
para.Read.udNrOfElementsToRead = 16;
fdo.GenOp.DataRead(Read, &para);

/* now rbuffer contains written elements */
printf("The first device byte is: 0x%x\n", rbuffer[0]);

return 0;
}
```

Software Limitations

The described software does not implement all functionality of the N25Q device. When an error occurs, the software simply returns the error message. It is up to the user to decide what to do. The user can either try the command again or, replace the device if necessary.

Conclusion

Micron's N25Q serial NOR Flash memory device is an ideal product for embedded and other computer systems. They can be easily interfaced to microprocessors and driven with simple software drivers written in the C language.

The Flash device driver interface enables changeable Flash memory configurations, compiler-independent data types, and a unique access mode for a broad range of Flash memory devices.

Moreover, applications supporting the Flash device driver standard can implement any Flash memory device with the same interface without any code change. Recompiling with a new software driver is all that is needed to control a new device.

8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-3900
www.micron.com/productsupport Customer Comment Line: 800-932-4992

Micron and the Micron logo are trademarks of Micron Technology, Inc. All other trademarks are the property of their respective owners.



Revision History

Rev. F	02/12
• Added 512Mb device to list of supported devices	
Rev. E	10/11
• Updated the file names in the introduction	
• Removed the memory layout description from the Programming Model section	
• Added ENTER 4-BYTE ADDRESS MODE and EXIT 4-BYTE ADDRESS MODE to the commands table	
• Added Segment selection (NVCR<1>) and Address byte (NVCR<0>) to the nonvolatile configuration register bits table	
• Added the Extended Address Register section	
• Replaced the Flag Status Register Bits table with the table from the data sheet	
• Removed the obsolete sample code section	
• Updated the Device Type description in the porting section	
• Removed the Flash() and ReadDeviceIdentification() functions	
• Modified the C library function names to remove the "flash" prefix	
• Added the Driver_Init(), Enter4ByteAddressMode(), and Exit4ByteAddressMode() C library functions	
• Replaced the FlashReadManufacturerIdentification() function with FlashReadDeviceIdentification()	
• Replaced the code sample in the Getting Started (Example Quick Test) section	
• Added Wrap Mode (VCR<1:0>) to the Volatile Configuration Register Bits table	
• Added Dual I/O protocol (VECR<5>), V _{pp} accelerator (VECR<3>), and Output driver strength <2:0> to the Volatile Enhanced Configuration Register Bits table	
Rev. D	05/11
• Rebranded as a technical note	
Rev. C	07/10
• Applied branding and formatting	
Rev. B	11/09
• Updated logos and branding	
Rev. A	04/09
• Initial release of document	