Application Note
Software Device Drivers for Micron Forté™ N25Q Serial Flash Memory

Introduction
This application note provides a description of the library source code in C for Micron Forté™ N25Q serial Flash memory devices, which use interfaces similar to the Flash software device driver interface specification. The N25Q128.c and N25Q128.h files contain libraries for accessing N25Q Flash memory devices.

This application note describes the operation of the memory devices 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 application 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 application 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.
Timeout .......................................................................................................................... 15
Additional Subroutines ....................................................................................................... 16
C Library Functions Provided .............................................................................................. 16
Read Volatile Enhanced Configuration Register ............................................................... 18
Write Volatile Enhanced Configuration Register ............................................................. 18
Write Nonvolatile Configuration Register ......................................................................... 18
Read Nonvolatile Configuration Register ........................................................................ 18
Clear Flag status Register .................................................................................................. 18
Getting Started (Example Quick Test) .............................................................................. 19
Software Limitations .......................................................................................................... 20
Conclusion ............................................................................................................................. 20
Revision History .................................................................................................................. 21
Disclaimer ............................................................................................................................ 22
Programming Model

The Micron Forté™ N25Q is a family of serial 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. The memory is organized as:

- 16,777,216 bytes (8 bits each)
- 256 sectors (64KB each)
- In Bottom and Top versions: 8 bottom (top) 64KB boot sectors with 16 subsectors
- (4KB) and 248 standard 64KB sectors
- 65,536 pages (256 bytes each)

Operating Features

The N25Q 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. This 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 Flash memory device.
- Using the nonvolatile configuration register: This is dedicated to applications that must boot directly in XiP mode.

N25Q128 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 non-volatile 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 bytes (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/Vpp, 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 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 provided in the following list:

- Write Enable (WREN)
- Write Disable (WRDI)
- Read Identification (RDID)
- Read Status Register (RDSR)
- Write Status Register (WRSR)
- Read Lock Register (RDLR)
- Write to Lock Register (WRLR)
- Read Flag Status Register (RFSR)
- Clear Flag Status Register (CLFSR)
- Read Volatile Configuration Register (RDVCR)
- Write Volatile Configuration Register (WRVCR)
- Read Volatile Enhanced Configuration Register (RDVECR)
- Write Volatile Enhanced Configuration Register (WRVECR)
- Read Nonvolatile Configuration Register (RDNVCR)
- Write Nonvolatile Configuration Register (WRNVCR)
- Read Data Bytes (READ)
- Read Data Bytes at Higher Speed (FAST_READ)
- Dual Output Fast Read (DOFR)
- Dual Input Output Fast Read (DIOFR)
- Quad Output Fast Read (QOFR)
- Quad Input Output Fast Read (QIOFR)
- Read OTP (read 64 bytes of OTP area) (ROTP)
- Program OTP (Program 64 bytes of OTP area) (POTP)
- Page Program (PP)
- Dual Input Fast Program (DIFP)
- Dual Input Extended Fast Program (DIEFP)
- Quad Input Fast Program (QIFP)
- Quad Input Extended Fast Program (QIEFP)
- Sub-sector Erase (SSE)
- Sector Erase (SE)
- Bulk Erase (BE)
- Deep Power-down (DP)
- Release from Deep Power-down (RDP)
- Program/Erase Suspend (PES)
- Program/Erase Resume (PER)
Write Enable (WREN) Command

The Write Enable (WREN) instruction sets the Write Enable Latch (WEL) bit. The Write Enable Latch (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 Disable (WRDI) Command

The Write Disable (WRDI) instruction 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.

Read Identification (RDID) Command

The Read Identification (RDID) instruction allows the device identification data to be read:
- Manufacturer identification (1 byte)
- Device identification (2 bytes)
- 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 Status Register (RDSR) Command

The Read Status Register (RDSR) instruction allows the Status Register to be read.

Write Status Register (WRSR) Command

The Write Status Register (WRSR) instruction allows new values to be written to the Status Register. The Write Status Register (WRSR) instruction allows the user to change the values of the Block Protect (TB, BP2, BP1, BP0) bits and to define the size of the area that is to be treated as read-only. The Write Status Register (WRSR) instruction also allows the user to set and reset the Status Register Write Disable (SRWD) bit in accordance with the Write Protect (nW/VPP) signal. The Status Register Write Disable (SRWD) bit and Write Protect (nW/VPP) signal allow 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.

Write to Lock Register (WRLR) Command

The Write to Lock Register (WRLR) instruction allows bits to be changed in the Lock Registers.

Read Lock Register (RDLR) Command

The Read Lock Register (RDLR) instruction allows the Lock Register to be read.

Read Data Bytes (READ) Command

The Read Data Bytes (READ) instructions output the data stored at the specified device addresses.
Read Data Bytes at Higher Speed (FAST_READ) Command

Read Data Bytes at Higher Speed (FAST_READ) instructions output the data stored at the specified device addresses at a higher speed.

Dual Output Fast Read (DOFR) Command

The Dual Output Fast Read (DOFR) instruction 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.

Read OTP (read 64 bytes of OTP area) (ROTP) Command

The Read OTP (ROTP) instruction outputs the data at the OTP area.

Program OTP (Program 64 bytes of OTP area) (POTP) Command

The Program OTP instruction (POTP) is 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, that is 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.

Page Program (PP) Command

The Page Program (PP) instruction allows bytes to be programmed in the memory (changing bits from 1 to 0).

Dual Input Fast Program (DIFP) Command

The Dual Input Fast Program (DIFP) instruction 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.

Sub-sector Erase (SSE) Command

The Sub-sector Erase (SSE) instruction 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.

Sector Erase (SE) Command

The Sector Erase (SE) instruction 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 (TB,BP2, BP1, BP0) bits is not executed.
Bulk Erase (BE) Command

The Bulk Erase (BE) instruction sets all bits to 1 (FFh). The Bulk Erase (BE) instruction is executed only if all Block Protect (TB, BP2, BP1, BP0) bits are 0. The Bulk Erase (BE) instruction is ignored if one or more sectors are protected.

Deep Power-down (DP) Command

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.

Release from Deep Power-down (RDP) Command

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.

Dual Input Extended Fast Program (DIEFP) Command

The Dual Input Extended Fast Program (DIEFP) instruction is similar to the Dual Input Fast Program (DIFP), except that the address bits are shifted in on two pins (pin DQ0 and pin DQ1) instead of only one pin.

Dual Input/Output Fast Read (DIOFR) Command

The Dual I/O Fast Read (DIOFR) instruction 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.

Program/Erase Suspend Command

The Program/Erase Suspend instruction interrupts a Program or an Erase instruction. By design, Bulk Erase and Program OTP cannot be suspended.

Program/Erase Resume Command

The Program/Erase Resume instruction resumes the suspended Program or Erase operation to resume after a Program/Erase Suspend instruction.

Quad Input Fast Program (QIFP)

The Quad Input Fast Program (QIFP) instruction 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 Extended Fast Program (QIEFP) Command

The Quad Input Extended Fast Program (QIEFP) instruction 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/Output Fast Read (QIOFR) Command

The Quad I/O Fast Read (QIOFR) instruction 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) Command

The Quad Output Fast Read (QOFR) instruction 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 one doubles the data transfer bandwidth when compared to the Dual Output Fast Read (DOFR) instruction.

Write Volatile Configuration Register (WRVCR) Command

The Write Volatile Configuration register (WRVCR) instruction allows 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).

Read Volatile Configuration Register (RDVCR) Command

The Read Volatile Configuration Register (RDVCR) instruction allows the Volatile Configuration Register to be read.

Read Volatile Enhanced Configuration Register (RDVECR) Command

The Read Volatile Enhanced Configuration Register (RDVECR) instruction allows the Volatile Configuration Register to be read.

Write Volatile Enhanced Configuration Register (WRVECR) Command

The Write Volatile Enhanced Configuration Register (WRVECR) instruction allows 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).

Write Nonvolatile Configuration Register (WRNVCR) Command

The Write Nonvolatile Configuration Register (WRNVCR) instruction allows 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).

Read Nonvolatile Configuration Register (RDNVCR) Command

The Read Nonvolatile Configuration Register (RDNVCR) instruction allows the Nonvolatile Configuration Register to be read.

Clear Flag status Register (CLFSR) Command

The Clear Flag Status Register (CLFSR) instruction resets the error Flag Status Register bits (Erase Error bit, Program Error bit, VPP Error bit and Protection Error bit). It is not necessary to set the WEL bit before the Clear Flag Status Register instruction is executed.

Read Flag Status Register (RFSR) Command

The Read Flag Status Register instruction (RFSR) allows the Flag Status Register to be read.
### 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 or not the operation was successful. The status bits of the Status Register are described in the following table:

#### Table 1. Status Register Status Bits

<table>
<thead>
<tr>
<th>Status Register bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIP bit</td>
<td>The 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.</td>
</tr>
<tr>
<td>WEL bit</td>
<td>The Write Enable Latch (WEL) bit indicates the status of the internal Write Enable Latch. When set to 1, the internal Write Enable Latch is set. When set to 0, the internal Write Enable Latch is reset and no Write, Program or Erase instruction is accepted.</td>
</tr>
<tr>
<td>BPx bits</td>
<td>The Block Protect (BPx) bits are non-volatile. They determine the size of the area to be software-protected against Program and Erase instructions. The Block Protect 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 Block Protect (BPx) bits to 0. If not, the Bulk Erase (BE) instruction will not be executed.</td>
</tr>
<tr>
<td>TB bit</td>
<td>The Top/Bottom (TB) bit is non-volatile. It can be set and reset with the Write Status Register (WRSR) instruction, provided that the Write Enable (WREN) instruction has been issued. The Top/Bottom (TB) bit is used in conjunction with the Block Protect (BP0, BP1 and BP2) bits to determine if the protected area defined by the Block Protect bits starts from the top or the bottom of the memory array.</td>
</tr>
<tr>
<td>SRWD bit</td>
<td>The Status Register Write Disable (SRWD) bit is operated in conjunction with the Write Protect (W) signal. The Status Register Write Disable (SRWD) bit and Write Protect (W) signal allow the device to be put in the Hardware Protected mode (when the Status Register Write Disable [SRWD]) bit is set.</td>
</tr>
</tbody>
</table>
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 following two bits control the protection of each sector:

Table 2. Lock Register Bits

<table>
<thead>
<tr>
<th>Lock Register Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Lock bit</td>
<td>The Write Lock bit determines whether or not 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.</td>
</tr>
<tr>
<td>Lock Down bit</td>
<td>The 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.</td>
</tr>
</tbody>
</table>

Nonvolatile Configuration Register

The nonvolatile configuration register (NVCR) bits affect the default memory configuration after power-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 power-on sequence. The nonvolatile configuration register bits are as follows:

Table 3. Nonvolatile Configuration Register Bits

<table>
<thead>
<tr>
<th>NVCR bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy Clock Cycles bit (NVCR&lt;15:12&gt;)</td>
<td>Used to optimize instruction execution according to the clock frequency.</td>
</tr>
<tr>
<td>XIP enabling bit (NVCR&lt;11:9&gt;)</td>
<td>Enables XiP.</td>
</tr>
<tr>
<td>Output Driver Strength bits (NVCR&lt;8:6&gt;)</td>
<td>Used to set impedance at Vcc/2.</td>
</tr>
<tr>
<td>Fast POR read bit (NVCR&lt;5&gt;)</td>
<td>Enables/disables the POR phase (disable is the default value).</td>
</tr>
<tr>
<td>Reset/Hold disable bit (NVCR&lt;4&gt;)</td>
<td>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.</td>
</tr>
<tr>
<td>Quad Input command bit (NVCR&lt;3&gt;)</td>
<td>Enables the QUAD SPI protocol.</td>
</tr>
<tr>
<td>Dual Input command bit (NVCR&lt;2&gt;)</td>
<td>Enables the DIO SPI protocol.</td>
</tr>
</tbody>
</table>
Volatile Configuration Register

The Volatile Configuration Register (VCR) affects the memory configuration after every execution of a Write Configuration Register instruction. The Volatile Configuration Register bits are as follows:

**Table 4. Volatile Configuration Register Bits**

<table>
<thead>
<tr>
<th>VCR bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy Clock Cycles bit (VCR&lt;7:4&gt;)</td>
<td>Used to optimize instruction execution according to the clock frequency.</td>
</tr>
<tr>
<td>XIP enabling bit (VCR&lt;3&gt;)</td>
<td>Enables XiP.</td>
</tr>
<tr>
<td>Other bits (VCR&lt;3&gt;)</td>
<td>Reserved, fixed value 0.</td>
</tr>
</tbody>
</table>

Enhanced Volatile 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). The Enhanced Volatile Configuration Register bits are as follows:

**Table 5. Enhanced Volatile Configuration Register Bits**

<table>
<thead>
<tr>
<th>VECR bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quad Input command bit (VECR&lt;7&gt;)</td>
<td>Enables/disables the QUAD SPI protocol.</td>
</tr>
<tr>
<td>Dual Input command bit (VECR &lt;6&gt;)</td>
<td>Enables/disables the DIO SPI protocol.</td>
</tr>
<tr>
<td>Other bits (VECR &lt;5&gt;)</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Reset/Hold enable/disable bit (VECR&lt;4&gt;)</td>
<td>Enables/disables the Pad Hold/Reset functionality.</td>
</tr>
<tr>
<td>QIO-SPI (VECR &lt;3&gt;)</td>
<td>Used to determine whether or not it is possible to use the Vpp accelerating voltage to speed up an internal modify operation with Quad program and erase instructions.</td>
</tr>
<tr>
<td>Output Driver Strength bits (VECR &lt;2:0&gt;)</td>
<td>Used to set impedance at Vcc/2.</td>
</tr>
</tbody>
</table>
Flag Status Register

The Flag Status Register is a powerful 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. The Flag Status Register bits are as follows:

Table 6. Flag Status Register Bits

<table>
<thead>
<tr>
<th>Flag Status Register bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P/E Controller (not Write In Progress) bit (&lt;7&gt;)</td>
<td>Represents the Program/Erase Controller Status bit.</td>
</tr>
<tr>
<td>Dual Input command bit (&lt;6&gt;)</td>
<td>The bit 6 of the Flag Status register represents the Erase Suspend Status bit. It indicates that an Erase operation has been suspended or is going to be suspended.</td>
</tr>
<tr>
<td>Other bits (VECR &lt;5&gt;)</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Reset/Hold enable/disable bit (VECR&lt;4&gt;)</td>
<td>Enables/disables the Pad Hold/Reset functionality.</td>
</tr>
<tr>
<td>QIO-SPI (VECR &lt;3&gt;)</td>
<td>Determines whether or not it is possible to use the Vpp accelerating voltage to speed up an internal modify operation with Quad program and erase instructions.</td>
</tr>
<tr>
<td>Output Driver Strength bits (VECR &lt;2:0&gt;)</td>
<td>Used to set impedance at Vcc/2.</td>
</tr>
</tbody>
</table>

An Example

The Instruction Set Table provided in the corresponding data sheet describes the instruction sequences recognized as valid by the device. As an example, consider the programming of the values 94h and 65h to the addresses 03E2h and 03E3h, respectively. The required C language sequence is as follows:

```c
NMX_uint8 obuffer[2] = {0x94, 0x65};
FlashPageProgram(0x3E2h, &obuffer[0], 2);
```

where NMX_uint8 is defined as the following 8-bit value:

```c
typedef unsigned char NMX_uint8
```

The first of the two addresses (03E2h) is arbitrary. However, the address given must be within the Flash memory address space.

In this example, it is assumed that the target area has been erased before initiating the programming operation, and that it is not write protected.
How to Use the Software Driver

General Considerations

The software drivers described in this application note are 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 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 Flash memory devices.

When developing an application, the user is advised to proceed as follows:
1. Write a simple program to test the low-level code provided, and verify that it operates as expected in the user's target hardware and software environments.
2. Write the high-level code for the desired application. The application will access the serial Flash memory device by calling the low-level code.
3. Thoroughly test the complete source code of the application.
Porting the Drivers to the Target System (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

Choose the correct device by using the appropriate define statement. For example:

#define USE_N25Q128 // this macro enables support for N25Q128

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 one-second delay. For example, if 100,000 repetitions of a loop are needed to give a time delay of one 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 allow correct completion. In accordance to the corresponding data sheet, a suitable timeout value is configured in each function where required.
### Additional Subroutines

```c
#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 Provided

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

#### Table 7. C Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash()</td>
<td>This function is used to access all device functions. It acts as the main Flash memory interface. This function is available on all software drivers written in the Flash device driver format and should be used exclusively. Any functionality unsupported by the Flash memory device can be detected and malfunctions can thus be avoided. Note that the other functions are listed to offer a second-level interface when enhanced performance is required.</td>
</tr>
<tr>
<td>FlashWriteEnable()</td>
<td>This function is used to set the Write Enable Latch (WEL) bit by sending a Write Enable (WREN) instruction.</td>
</tr>
<tr>
<td>FlashWriteDisable()</td>
<td>This function is used to reset the Write Enable Latch (WEL) bit by sending a Write Disable (WRDI) instruction.</td>
</tr>
<tr>
<td>FlashReadDeviceIdentification()</td>
<td>This function is used to read the Device Identification.</td>
</tr>
<tr>
<td>FlashReadManufacturerIdentification()</td>
<td>This function is used to read the Manufacturer Identification (20h) by sending a Read Identification (RDID) instruction. Note that the last two instructions could be unified into a single function due to the fact that only a single instruction returns the Device Identification and the Manufacturer Identification.</td>
</tr>
<tr>
<td>FlashReadStatusRegister()</td>
<td>This function is used to read the status register by sending a Read Status Register (RDSR) instruction.</td>
</tr>
<tr>
<td>FlashWriteStatusRegister()</td>
<td>This function is used to write the status register by sending a Write Status Register (WRSR) instruction.</td>
</tr>
<tr>
<td>FlashWriteLockRegister()</td>
<td>This function is used to write the lock register by sending a Write Lock Register (WRLR) instruction.</td>
</tr>
<tr>
<td>FlashReadLockRegister()</td>
<td>This function is used to read the status register by sending a Read Lock Register (RDLR) instruction.</td>
</tr>
<tr>
<td>FlashRead()</td>
<td>This function is used to read the Flash memory by sending a Read Data Bytes (READ) instruction. By design, the entire Flash memory space can be read with one Read Data Bytes (READ) instruction by incrementing the start address and rolling to 0h automatically. This means that this function is across pages and sectors.</td>
</tr>
<tr>
<td>FlashFastRead()</td>
<td>This function is used to read 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 Read Data Bytes at Higher Speed (FAST_READ) instruction by incrementing the start address and rolling to 0h automatically. This means that this function is across pages and sectors.</td>
</tr>
<tr>
<td>FlashDualOutputFastRead()</td>
<td>This function is used to Read 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.</td>
</tr>
<tr>
<td>FlashReadOTP()</td>
<td>This function is used to read data from the OTP area by sending a Read OTP (ROTP) instruction.</td>
</tr>
</tbody>
</table>
Table 7. C Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlashProgramOTP()</td>
<td>This function is used to program the 64-byte OTP area by sending a Program OTP (POTP) instruction.</td>
</tr>
<tr>
<td>FlashPageProgram()</td>
<td>This function is used to program 256 bytes or less of data to the memory by sending a Page Program (PP) instruction. By design, the Page Program (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.</td>
</tr>
<tr>
<td>FlashDualInputFastProgram()</td>
<td>This function is used to program 256 bytes or less of data to the memory on two pins (pin DQ0 and pin DQ1) instead of only one by sending a Dual Input Fast Program (DIFP) instruction.</td>
</tr>
<tr>
<td>FlashSubSectorErase()</td>
<td>This function is used to erase a sub-sector by sending a Sub-sector Erase (SSE) instruction.</td>
</tr>
<tr>
<td>FlashSectorErase()</td>
<td>This function is used to erase an entire sector by sending a Sector Erase (SE) instruction.</td>
</tr>
<tr>
<td>FlashBulkErase()</td>
<td>This function is used to erase the entire memory by sending a Bulk Erase (BE) instruction.</td>
</tr>
<tr>
<td>FlashDeepPowerDown()</td>
<td>This function is used to set the device in the lowest power consumption mode (the Deep Powerdown mode) by sending a Deep Power-down (DP) instruction. After calling this routine, the Flash memory will not respond to any instruction except for the Release from Deep Power-down (RDP) instruction.</td>
</tr>
<tr>
<td>FlashReleaseFromDeepPowerDown()</td>
<td>This function is used to take the device out of the Deep Power-down mode by sending a Release from Deep Power-down (RDP) instruction.</td>
</tr>
<tr>
<td>FlashProgram()</td>
<td>This function is used to program a chunk of data into the memory at one time. After verifying the start address and checking the available space successfully, 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.</td>
</tr>
<tr>
<td>FlashProgramEraseResume()</td>
<td>This function resumes the program/erase operation that was suspended by sending a SPI_FLASH_INS_PER instruction.</td>
</tr>
<tr>
<td>FlashProgramEraseSuspend()</td>
<td>This function resumes the program/erase operation that was suspended by sending a SPI_FLASH_INS_PES instruction.</td>
</tr>
<tr>
<td>FlashQuadInputFastProgram()</td>
<td>This function is used to program 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.</td>
</tr>
<tr>
<td>FlashQuadInputExtendedFastProgram()</td>
<td>This function is used to program 256 bytes or less of data and address to the memory on four pins by sending 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.</td>
</tr>
<tr>
<td>FlashQuadInputOutputFastRead()</td>
<td>This function is used to Read data from the memory and send addresses on four pins instead of only one by sending a Quad Output Fast Read (QIOFR) instruction.</td>
</tr>
<tr>
<td>FlashQuadOutputFastRead()</td>
<td>This function is used to Read data from the memory on four pins instead of only one by sending a Quad Output Fast Read (QOFR) instruction.</td>
</tr>
</tbody>
</table>
How to Use the Software Driver

Read Volatile Enhanced Configuration Register

The Read Volatile Enhanced Configuration Register (RDVECR) instruction allows the Volatile Configuration Register to be read.

Write Volatile Enhanced Configuration Register

The Write Volatile Enhanced Configuration register (WRVECR) instruction allows 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).

Write Nonvolatile Configuration Register

The write nonvolatile configuration register (WRNVCR) instruction allows new values to be written to the nonvolatile 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).

Read Nonvolatile Configuration Register

The read nonvolatile configuration register (RDNVCR) instruction allows the nonvolatile configuration register to be read.

Clear Flag status Register

The Clear Flag Status Register (CLFSR) instruction reset the error Flag Status Register bits (Erase Error bit, Program Error bit, VPP Error bit, Protection Error bit). It is not necessary to set the WEL bit before the Clear Flag Status Register instruction is executed.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlashWriteVolatileConfigurationRegister()</td>
<td>The Write Volatile Configuration register (WRVCR) instruction allows 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.</td>
</tr>
<tr>
<td>FlashReadVolatileConfigurationRegister()</td>
<td>The Read Volatile Configuration Register (RDVCR) instruction allows the Volatile Configuration Register to be read.</td>
</tr>
</tbody>
</table>
Getting Started (Example Quick Test)

To test the source code in the target system, start by reading from the N25Q. If it is erased, only FFh data should be read. Then, read the Manufacturer and Device Identifications and verify that they are correct. If these functions work, the other functions are also likely to work. However, all functions should be tested thoroughly.

To start, write a main() function and include the C file as described in the following example. All serial Flash memory functions can be called and executed within the main() function. The following example shows a check of the device and manufacturer identifiers (Device Identification and Manufacturer Identification) and a simple Sector Erase instruction.

```c
#include "Serialize.h" /*Header file for SPI master abstract prototypes */
#include "N25Q128.h" /* Header file for Flash memory instructions */
#include <stdio.lib>

void main(void) {
    ParameterType fp; /* Contains all Flash memory Parameters */
    ReturnType rRetVal; /* Return Type Enum */
    Flash(ReadManufacturerIdentification, &fp);
    printf("Manufacturer Identification: %02Xh\r\n",
           fp.ReadManufacturerIdentification.ucManufacturerIdentification);
    Flash(ReadDeviceIdentification, &fp);
    printf("Device Identification: %04Xh\r\n",
           fp.ReadDeviceIdentification.ucDeviceIdentification);
}/* End of Function Main */
```
Software Limitations

The described software does not implement all functionality of the N25Q. 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, if necessary, replace the device.

Conclusion

The Micron Forté™ N25Q serial Flash memory device is ideal products 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 allows 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.
Revision History

Rev. C .............................................................................................................................. 07/10
  • Applied branding and formatting.

Rev. 2 .............................................................................................................................. 11/09
  • Updated logos and branding.

Rev. 1 .............................................................................................................................. 04/09
  • Initial release of document.
Disclaimer

Please Read Carefully:

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH MICRON PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN MICRON’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, MICRON ASSUMES NO LIABILITY WHATSOEVER, AND MICRON DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF MICRON PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Micron products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Micron may make changes to specifications and product descriptions at any time, without notice.

Micron Technology, Inc. may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Micron reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Contact your local Micron sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Micron literature may be obtained by visiting Micron's website at http://www.micron.com.

Micron StrataFlash™ and Axcell™ are trademarks or registered trademarks of Micron or its subsidiaries in the United States and other countries.

Other names and brands may be claimed as the property of others.

Copyright © 2010, Micron Technology, Inc., All Rights Reserved.