

Technical Note

Software Device Drivers for M28W640C Parallel NOR Flash Memory

Introduction

This technical note describes the library source code in C for M28W640CT and M28W640CB parallel NOR Flash memory devices, which are referred to as M28W640C throughout this document unless otherwise specified.

The source code is available from micron.com or your Micron distributor. The `c1663.c` and `c1663.h` files contain libraries for accessing M28W640C NOR Flash memory devices.

Also included in this technical note is an overview of the programming model for M28W640C devices. This overview outlines memory device operation 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 minimal changes by the user to compile and run. This technical note explains how to modify the source code for individual target hardware. The source code contains comments throughout that explain how it is used and why it has been written the way that it has.

This technical note does not replace the M28W640C data sheet. It refers to it throughout, and it is necessary to have a copy of the data sheet to follow some explanations. The software supplied with this documentation has been tested on a target platform and is usable in C and C++ environments. It is small in size and can be applied to any target hardware.

M28W640C Programming Model

M28W640C is a 64Mb (4Mb x 16) Flash memory that can be electrically erased at a block level and programmed in-system on a word-by-word basis through special coded command sequences on most standard microprocessor buses. The devices feature an asymmetrical block architecture. The M28W640C has an array of 135 blocks: 8 parameter blocks of 4-kilobyte words and 127 main blocks of 32-kilobyte words. M28W640CT memory devices have parameter blocks at the top of the memory address space, while M28W640CB memory devices locate the parameter blocks starting from the bottom. Each block can be erased separately. An ERASE can be suspended to either READ from or PROGRAM to another block, and then resumed. A PROGRAM operation can be suspended to read data in another block and then resumed. Each block can be programmed and erased over 100,000 cycles.

All blocks have three levels of protection. They can be locked and locked-down individually, preventing any accidental programming or erasure. The memory devices offer an additional hardware protection: when V_{PP} is lower than V_{PPLK} , all blocks are protected against an unwanted PROGRAM or ERASE. All blocks are locked at power-up. The devices include a protection register to increase the protection of a system's design.

PROGRAM and ERASE commands are written to the memory device's command interface. An on-chip PROGRAM/ERASE controller (P/E.C.) handles the timings necessary for PROGRAM and ERASE operations. The end of a PROGRAM or ERASE operation can be detected and any error conditions identified. The command set required to control the memory is consistent with JEDEC standards.

M28W640C devices offer two features to improve the programming throughput: the DOUBLE WORD PROGRAM command used to write a page of two adjacent words in parallel and the QUADRUPLE WORD PROGRAM command used to write a page of four adjacent words in parallel.

Note: Data with the current CPU data bus width is referred to as “elements” throughout the document unless otherwise specified. Due to the flexibility of the software driver, the size of an element depends on the current configuration (user change area).

Bus Operations and Commands

Most M28W640C functionality is available via the two standard bus operations: READ and WRITE. READ operations retrieve data or status information from the device. WRITE operations are interpreted by the device as commands that modify the data stored or the device's behavior. Only certain special WRITE operation sequences are recognized as commands by M28W640C devices. The various commands recognized by the devices are listed in the Commands Tables provided in the corresponding data sheets. The main commands are described in Table 1:

Table 1: Bus Operations and Commands

Command	Description
READ	This command returns the M28W640C to read mode where it behaves as a ROM. In this state, a READ operation outputs the data stored at the specified device address onto the data bus.
READ ELECTRONIC SIGNATURE	This command places the device in a mode that enables the user to read the electronic signature and block protection status. These are accessed by reading different addresses while the device is in read electronic signature mode.
ERASE	This is used to set all bits to 1 at every memory location in the selected block. The data previously stored in the erased block will be lost. The ERASE command takes longer to execute than other commands because an entire block is erased at once. Any attempts to ERASE or PROGRAM either a locked block or the memory while it is protected (for example, when V_{PP} is lower than V_{PPLK}) generate an error and leave the contents of the memory unchanged.
PROGRAM	This command is used to modify the data stored at the specified device address. Note that programming can only change bits from 1 to 0. If an attempt is made to change a bit from 0 to 1 using the PROGRAM command, the command will be executed and no error will be signaled, but the bit will remain unchanged. It may therefore be necessary to ERASE the block before programming to addresses within it. Programming modifies a single word at a time. Programming larger amounts of data must be done one word at a time by issuing a PROGRAM command, waiting for the command to complete, issuing the next PROGRAM command, and so forth.

Table 1: Bus Operations and Commands (Continued)

Command	Description
PROGRAM/ERASE SUSPEND	Issuing the PROGRAM/ERASE SUSPEND command during a PROGRAM or ERASE operation temporarily places the M28W640C device in program/erase suspend mode. While an ERASE operation is being suspended, the blocks not being erased can be read or programmed as if in the reset state of the device. While a PROGRAM operation is being suspended, the rest of the device can be read. This enables the user to immediately access information stored in the M28W640C device without having to wait for the PROGRAM or ERASE operation to complete. The PROGRAM or ERASE operation is resumed when the device receives the PROGRAM/ERASE RESUME command.
READ COMMON FLASH INTERFACE QUERY	This command enables the user to identify the number of blocks in the Flash memory device and the block addresses. The interface also contains information relating to the typical and maximum PROGRAM and ERASE times. This enables the user to implement software timeouts and prevents waiting indefinitely for a defective Flash memory device to finish programming or erasing. For further information about the CFI, please refer to the CFI specification available at http://www.jedec.org or from your Micron distributor.
BLOCK LOCK, BLOCK UNLOCK, and BLOCK LOCK-DOWN	Blocks can be protected against accidental PROGRAM and ERASE operations that could alter their contents. A block can be locked or locked-down. A locked block cannot be programmed or erased. A locked-down block cannot have its protection status changed when the WRITE PROTECT signal is LOW (V_{IL}). When the WRITE PROTECT signal is HIGH (V_{IH}), the LOCK-DOWN function is disabled and each block can be locked or unlocked independently of the others. All blocks are locked at power-up and reset.

Status Register

During PROGRAM or ERASE operations, a BUS READ operation outputs the contents of the status register. The status register, which can also be accessed by issuing the READ STATUS REGISTER command, provides valuable information about the latest PROGRAM or ERASE operation. The status register bits are described in the Status Register Bits tables in the M28W640C data sheet. They are primarily used to determine when programming or erasing is complete and whether the operation was successful.

The completion or suspension of the PROGRAM or ERASE operation is indicated by the PROGRAM/ERASE controller status bit (status register bit DQ7) going HIGH (V_{IH}). Programming or erasing errors are indicated by one or more error bits (status register bits DQ1, DQ3, DQ4, and DQ5) going HIGH. In the case of a failure, a CLEAR STATUS REGISTER command must be issued to reset the status register error bits. Otherwise, it will be not possible to determine whether subsequent operations are successful.

A Detailed Example

The Command tables in the M28W640C data sheet describe the BUS WRITE sequences recognized as valid commands by the PROGRAM/ERASE controller.

As an example, consider programming the value 9465h to address 03E2h. The required C language sequence is:

```
*(uword*) (0x0000) = 0x0040; /* 1st cycle (any block address) */  
*(uword*) (0x03E2) = 0x9465; /* 2nd cycle: address and data */
```

where uword is defined as the following 16-bit value:

```
typedef unsigned short uword
```

The first of the two addresses (0000h) is arbitrary, but must be inside the Flash memory address space. The example assumes that address 0000h in the M28W640C device is mapped to address 0000h in the microprocessor address space. In practice, Flash devices are likely to have a base offset that must be added to the address.

While the device is programming to the specified address, READ operations will access the status register bits. Status register bit DQ7 will be 0 (LOW) during programming and switch to 1 (HIGH) upon completion. If any of the status register bits DQ1, DQ3, or DQ4 goes HIGH upon completion of the PROGRAM operation, it means that the operation has failed.

Once programmed, address 03E2h cannot be reprogrammed reliably until an ERASE operation is issued to erase the entire block.

Using the Software Driver

General Considerations

The software device drivers described in this technical note are intended to simplify the process of developing an application code in C for M28W640C Flash devices.

Note: To meet compatibility requirements, the M28W640C software device driver numbers each block in a Flash memory device starting from 0 (block 0 always has address offset 0) up to the highest address block number in the device. Block numbers may be described differently in the data sheets. For example, in a Flash device containing 64 blocks, it will always refer to the block with address offset 0 as block number 0, and to the last block as block number 63.

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 so that users do not have to consider the details of the special command 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:

- Hardware-specific bus operations
- Low-level code
- High-level code written by the user

The low-level code requires hardware-specific READ and WRITE bus operations in C to communicate with an M28W640C device. The implementation of these operations is hardware-platform dependent as it depends on the microprocessor 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 drivers that are suitable for the current hardware platform. The low-level code issues the correct WRITE operation sequence for each command and interprets the information received from the devices during programming and erasing.

The high-level code written by the user accesses the memory devices by calling the low-level code. In this way, the code used is simple and easier to maintain. Another consequence is that the user's high-level code is easier to apply to other Micron Flash memory devices.

When developing an application, it is recommended to:

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 accesses the 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 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 following items 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 ubyte; (8 bits)
typedef char byte; (8 bits)
typedef unsigned short uword; (16 bits)
typedef short word; (16 bits)
typedef unsigned int udword; (32 bits)
typedef int dword; (32 bits)
```

Device Type

Use the appropriate define statement to choose the correct device:

```
#define USE_M28W640CT
#define USE_M28W640CB
```

Flash Memory Location

BASE_ADDR is the start address of the Flash memory device. It must be set according to the target system to access the Flash memory device at the correct address. This value is used by the FlashRead() and FlashWrite() functions. The default value is set to 0, and must be adjusted appropriately:

```
#define BASE_ADDR ((volatile uCPUBusType*)0x00000000)
```

Flash Configuration

Choose the correct Flash memory configuration:

```
#define USE_16BIT_CPU_ACCESSING_1_16BIT_FLASH
```

This define statement supports a board configuration containing a CPU with an external 16-bit memory bus with a single 16-bit Flash memory device connected to it.

```
#define USE_32BIT_CPU_ACCESSING_2_16BIT_FLASH
```

This define statement supports a board configuration containing a CPU with an external 32-bit memory bus with two 16-bit Flash memory devices connected to it.

Timeout

Timeouts are implemented in the loops of code to provide an exit for 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` is used. 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 100000.

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

Note: This delay depends on hardware performance and should 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. As a result, it may 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.

Additional Subroutines

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

```
#define VERBOSE
```

Additional Considerations

The access timing data for the Flash memory device can sometimes be problematic. It may be necessary to change the `FlashRead()` and `FlashWrite()` functions if they are not compatible with the timings of the target hardware. This can be solved with a logic state analyzer.

The programmer must take extra care when the device is accessed during an interrupt service routine. When the device is in read mode, interrupts can freely read from the device. Interrupts that do not access the device may be used during all functions.



C Library Functions Provided

The software library described in this technical note provides the source code for the functions described in Table 2:

Table 2: C Library Functions

Function	Description
Flash()	This is used to access all device functions and 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: The other functions are listed to offer a second-level interface when enhanced performance is required. Within the Flash device driver, the functions are always used in the same way, which means that the function interface (names, return codes, parameters, and data types) remains unchanged regardless of the Flash memory device.
FlashBlockErase()	This is used to erase a block in the device. A block cannot be erased when it is locked or V_{PP} is invalid (lower than V_{PPLK}). Attempting to do so generates an error.
FlashBlockLockDown()	This is used to lock-down a block. Once locked-down, the block is locked and when WP is LOW (V_{IL}), the lock status of the block cannot be changed by using the software commands alone. The block reverts to the locked state when the device is reset or powered down.
FlashBlockProtect()	This is used to lock (protect) a block in the Flash memory device. Once locked (protected), the data in the block cannot be programmed or erased until the block is unlocked (unprotected).
FlashBlockUnprotect()	This is used to unlock (unprotect) a block in the Flash memory device. Once the block is unlocked, the data it contains can be erased or new data can be programmed to it.
FlashCheckBlockLockDownStatus()	This is used to check whether a block is locked-down.
FlashCheckBlockProtection()	This is used to check whether a block is locked.
FlashCheckCompatibility()	This is used to check the Flash memory device for compatibility.
FlashChipErase()	This is used to erase the entire device. Locked blocks will be not erased. The device cannot be erased when V_{PP} is invalid (lower than V_{PPLK}). Attempting to do so generates an error.
FlashChipUnprotect()	This is used to unlock all blocks in the Flash memory device. Once all the blocks are unlocked, the data contained in all the blocks can be entirely erased or new data can be programmed.
FlashClearStatusRegister()	This is used to clear the status register.
FlashDoubleProgram()	This is used to program the memory by issuing the DOUBLE WORD PROGRAM command.
FlashErrorStr()	This is used to generate a text string describing the detected error.

Table 2: C Library Functions (Continued)

Function	Description
FlashProgram()	This is used to program data arrays to the Flash memory device. Only previously erased elements can be programmed reliably. Locked blocks cannot be programmed, and PROGRAM operations cannot be performed when V_{pp} is invalid.
FlashProtectionRegisterProgram()	This is used to program the protection register.
FlashQuadProgram()	Available only for M28W640C devices, this is used to program the memory by issuing the QUADRUPLE WORD PROGRAM command.
FlashReadCfi()	This is used to check if the common Flash interface (CFI) is supported and then read the CFI data at the specified offset.
FlashReadDeviceId()	This is used to read the device codes of the Flash memory device.
FlashReadManufacturerCode()	This is used to read the manufacturer codes of the Flash memory device.
FlashReadProtectionRegister()	This is used to read a location in the protection register.
FlashReadStatusRegister()	This is used to read the status register.
FlashReset()	This is used to reset the device to read array mode. Note: There should be no need to call this function under normal operation as all the other software library functions leave the device in this mode.
FlashResume()	This is used to resume the PROGRAM or ERASE operation being suspended.
FlashSingleProgram()	This is used to program a single element.
FlashSuspend()	This is used to suspend the PROGRAM or ERASE operation in progress. The functions provided in the software library rely on the user implementing the hardware-specific bus operations and on access timings to communicate properly with the Flash device. If changes in the software driver are necessary, the only two functions that need to be changed are FlashRead() and FlashWrite().
FlashRead()	This is used to read a value from the Flash memory device.
FlashWrite()	This is used to write a value to the Flash memory device.

Getting Started (Example Quicktest)

To test the source code in the target system, start by reading from the M28W640C device. If it is erased, only FFFFh data should be read. Then, read the manufacturer and device codes and verify that they are correct. If these functions work, it is likely that the other functions will also work. However, all functions should be tested thoroughly.

To start, write a function main() and include the C file as described in the following example. All Flash memory functions can be called and executed within the main function.

The following example shows a check of the device identifiers (device code, manufacturer code) and a simple BlockErase command.

```
#include "c1663.c"

void main(void) {
    ParameterType fp; /* Contains all Flash Parameters */
    ReturnType rRetVal; /* Return Type Enum */

    Flash(ReadManufacturerCode, &fp);
    printf("Manufacturer Code: %08Xh\r\n",
        fp.ReadManufacturerCode.ucManufacturerCode);

    Flash(ReadDeviceId, &fp);
    printf("Device Code: %08Xh\r\n",
        fp.ReadDeviceId.ucDeviceId);

    fp.BlockErase.ulBlockNr = 10; /* block 10 will be erased*/
    rRetVal = Flash(BlockErase, &fp); /* function execution */

} /* EndFunction Main */
```

Software Limitations

The software described in this technical note does not implement the full set of M28W640C functionality. When an error occurs, the software simply returns the error message. It is up to the user to decide what to do. They can either try the command again or replace the device, if necessary.

Conclusion

M28W640C 3V supply, parallel NOR Flash memory devices are ideal 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 M28W640C driver interface enables changeable Flash configurations, compiler-independent data types, and a unique access mode for a broad range of Flash devices.

In addition, applications supporting the software can implement any Flash device with the same interface, without any code change. A simple recompiling with a new software driver is all that is required 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. C01/12
<ul style="list-style-type: none">• Edited and formatted document• Rebranded as a technical note	
Rev. B05/11
<ul style="list-style-type: none">• Minor changes	
Rev. A02/03
<ul style="list-style-type: none">• Initial release of document	