Technical Note

Software Device Drivers for Micron® MT29F NAND Flash Memory

Introduction

This technical note explains how to use Micron's MT29F NAND Flash memory software device drivers, which are the low-level drivers (LLDs) that manage the hardware functionality of the NAND Flash memory devices.

Note: For all Micron ONFI-compliant NAND

This document describes the operation of the devices and provides a basis for understanding and modifying the accompanying source code. The source code driver is written to be as platform independent as possible, and requires minimal changes to compile and run. The technical note also explains how to modify the source code for a target system. The source code is available at www.micron.com or from your Micron distributor.

This technical note does not replace the data sheet for any NAND device. It refers to the data sheet throughout, and it is necessary to have a copy of it to follow some explanations. Refer to the data sheets for the corresponding device part numbers and densities.

Contact your Micron representative for more information about Micron's NAND Flash memory software device drivers.

MT29F NAND Overview

MT29F NAND Flash is a family of nonvolatile Flash memory devices use NAND cell technology. The devices operate either from a 1.8V or a 3V voltage supply.

The address lines are multiplexed with the data input/output signals on a multiplexed x8 or x16 input/output bus. This interface reduces the pin count and makes it possible to migrate to other densities without changing the footprint.

The devices have the following hardware and software security features:

- A WRITE protect pin ($\overline{WP}$) is available to provide hardware protection against PROGRAM and ERASE operations.
- A block locking scheme is available to provide user code and/or data protection.

The devices feature an open-drain ready/busy output that can be used to identify whether the PROGRAM/ERASE/READ (P/E/R) controller is currently active. The use of an open-drain output makes it possible for the ready/busy pins from several memory devices to be connected to a single pull-up resistor.

A COPY BACK PROGRAM command optimizes the management of defective blocks. When a PAGE PROGRAM operation fails, the data can be programmed in another page without resending the data to be programmed.

Cache program and cache read features improve the PROGRAM and READ throughputs for large files. During cache programming, the device loads the data in a cache register while the previous data is transferred to the page buffer and programmed into the memory array. During cache reading, the device loads the data in a cache register while the previous data is transferred to the I/O buffers to be read.

Micron's MT29F NAND Flash memory devices can be delivered with an optional unique identifier (serial number) that makes it possible for each device to be uniquely identified. The unique identifier option is subject to a non-disclosure agreement (NDA), and as a result is not described in the data sheet.

A lock/unlock enable input (LOCK) is used to enable and disable the lock mechanism. When LOCK is HIGH (V_{IH}), the device is in block lock mode and a BLOCK UNLOCK command is required before programming or erasing a block.

Note: The LOCK pin may not be supported. Please refer to the NAND data sheet for a specific device to check if this functionality is supported.

In addition, Micron's M68A/69A/60A NAND Flash memory devices support internal error correction code (ECC). For a full description, please refer to the "Internal ECC and Spare Area Mapping for ECC" section of the device's data sheet.

Bus Operations

The standard bus operations that control Micron MT29F NAND Flash memory devices are:

- COMMAND INPUT
- ADDRESS INPUT
- DATA INPUT
- DATA OUTPUT
- WRITE PROTECT

Table 1: Bus Operations

Bus Operation	Description
COMMAND INPUT	Sends commands to the memory device.
ADDRESS INPUT	Inputs the memory addresses. Five bus cycles are required for 2Gb and larger devices. Refer to the NAND Flash memory data sheet for a detailed description of addresses and address input cycles on x8 and x16 devices.
DATA INPUT	Inputs the data to be programmed.
DATA OUTPUT	Reads data from the memory array, status register content, block lock status, electronic signature, or unique identifier.
WRITE PROTECT	Protects the memory against PROGRAM or ERASE operations. When the WRITE PROTECT signal is LOW, the NAND memory device does not accept PROGRAM or ERASE operations. As a result, the contents of the memory array cannot be altered.

Device Operations

All bus WRITE operations to the device are interpreted by the command interface. The following commands are available on Micron MT29F NAND Flash memory devices:

- PAGE READ
- RANDOM DATA OUTPUT
- CACHE READ
- PAGE PROGRAM
- MULTIPLANE PAGEPROGRAM
- RANDOM DATA INPUT
- COPYBACK PROGRAM
- MULTI-PLANE COPYBACK PROGRAM
- CACHE PROGRAM
- BLOCK ERASE
- MULTIPLANE BLOCK ERASE
- RESET
- READ STATUS REGISTER
- READ ENHANCED STATUS REGISTER
- READ ELECTRONIC SIGNATURE
- BLOCKS LOCK
- BLOCK UNLOCK
- BLOCKS LOCK-DOWN
- BLOCK LOCK STATUS
- GET FEATURE
- SET FEATURE
- OTP MODE SUPPORT

Note: Refer to the data sheet for the NAND device for a detailed description of the device operations.

Table 2: Device Operations

Device Operation	Description
PAGE READ	After the first READ access, the page data is transferred to the page. Once the transfer is complete, the data can then be read out sequentially (from the selected column address to the last column address).
RANDOM DATA OUTPUT	The device can output random data in a page (instead of consecutive sequential data) by issuing a RANDOM DATA OUTPUT command. This command can be used to skip some data during a sequential data output.
CACHE READ	Improves the READ throughput by reading data using the cache register. When the user starts to read one page, the device automatically loads the next page into the cache register.
PAGE PROGRAM	This is the standard operation to program data to the memory array. The memory array is programmed by page. However, partial page programming is allowed where any number of bytes can be programmed.

Table 2: Device Operations (Continued)

Device Operation	Description
MULTIPLANE PAGEPROGRAM	Issues a MULTIPLANE PAGEPROGRAM command, which enables the programming of two pages in parallel, one in each plane. It can perform multi-plane page program with random data input. A MULTIPLANE PAGEPROGRAM operation requires the following two steps: 1. Serially load up to two pages of data into the data buffer. 2. Parallel programming of both pages starts after the issue of the Page Confirm command. If the MULTIPLANE PAGEPROGRAM fails, an error is signaled on bit SR0 of the status register. To determine which page of the two planes failed, the READ STATUS ENHANCED command must be issued twice, once for each plane. Refer to the NAND device's data sheet for a description of the multi-plane operation and the restrictions related to the multi-plane page program sequence.
RANDOM DATA INPUT	During a SEQUENTIAL INPUT operation, the next sequential address to be programmed can be replaced by a random address by issuing a RANDOM DATA INPUT command.
COPY BACK PROGRAM	Copies the data stored in one page and reprograms it in another page. The operation is particularly useful when a portion of a block is updated and the rest of the block must be copied to the newly assigned block.
MULTI-PLANE COPYBACK PROGRAM	This function issues a MULTI-PLANE COPYBACK PROGRAM command as explained in the data sheet. The function permits random data input. In this case, the data inserted are all 1. The multi-plane copy back program requires the same steps as the multi-plane page program command. If the MULTI-PLANE COPYBACK PROGRAM fails, an error is signaled on bit SR0 of the status register. To know which page of the two planes failed, the READ STATUS ENHANCED command must be executed twice, once for each plane. Refer to the NAND device's data sheet for a description of the multi-plane operation and specific information related to the multi-plane copy back program sequence.
CACHE PROGRAM	Improves the programming throughput by programming data using the cache register. The CACHE PROGRAM operation can only be used within one block. The cache register allows new data to be input while the previous data that was transferred to the page buffer is programmed into the memory array.
BLOCK ERASE	ERASE operations are performed one block at a time. An ERASE operation sets all bits in the addressed block to 1. All previous data in the block is lost.
MULTI-PLANE BLOCK ERASE	Issues a MULTI-PLANE BLOCK ERASE command, which erases two blocks in parallel, one in each plane. To determine which page of the two planes failed, the READ STATUS ENHANCED command must be executed twice, once for each plane. Refer to the NAND device's data sheet for a description of the multi-plane operation and specific information related to the block erase sequence.
RESET	Resets the command interface and status register. If the RESET command is issued during any operation, the operation will be aborted. If it was a PROGRAM or ERASE operation that was aborted, the contents of the memory locations being modified will no longer be valid as the data will be partially programmed or erased.
READ STATUS REGISTER	The device contains a status register, which provides information on the current or previous PROGRAM or ERASE operation. The various bits in the status register convey information and errors on the operation. The status register is read by issuing the READ STATUS REGISTER.
READ ENHANCED STATUS REGISTER	This function is used after a COPYBACK or MULTI-PLANE COPYBACK to check if there is an EDC error inside the page.
READ ELECTRONIC SIGNATURE	Enables device information, such as manufacturer code and device code, to be read. Refer to the NAND device's data sheet for a description of electronic signature values.
BLOCKS LOCK	BLOCKS LOCK is a data protection command that enables all blocks to be locked simultaneously. It is can be issued only if the PRL signal is HIGH (lock/unlock enabled). Refer to the NAND device's data sheet for a detailed description of the block protection scheme.

Table 2: Device Operations (Continued)

Device Operation	Description
BLOCK UNLOCK	BLOCK UNLOCK is a data protection command used to unlock a sequence of consecutive locked blocks to enable PROGRAM or ERASE operations. It can be issued only if the LOCK signal is HIGH (lock/unlock enabled). Refer to the NAND device's data sheet for a detailed description of the block protection scheme.
BLOCKS LOCK-DOWN	BLOCKS LOCK-DOWN is a data protection command that provides an additional level of protection. When locked-down, a block cannot be unlocked by a software command. Locked-down blocks can only be unlocked by setting the WRITE PROTECT signal to LOW. This command can be issued only if the LOCK signal is HIGH (lock/unlock enabled). Refer to the NAND device's data sheet for a detailed description of the block protection scheme.
BLOCK LOCK STATUS	Checks the block lock status of each block. It can be issued only if the PRL signal is HIGH (lock/unlock enabled). Refer to the NAND device's data sheet for a detailed description of the block protection scheme.
GET/SET FEATURE	Modifies the target's default power-on behavior, such as the timing mode and internal ECC.
OTP MODE SUPPORT	Includes a set of functions that allow users to manage the OTP area (See How to Manage OTP Area for more details): - OTP_Mode_Enter() - OTP_Mode_Exit() - OTP_Mode_Protect() - OTP_Page_Program() - OTP_Spare_Program() - OTP_Page_Read() - OTP_Spare_Read().

Enhanced Status Register

The devices contain an enhanced status register, which provides information about the addressed LUN. When a PROGRAM, ERASE, or other operation occurs, it is possible to retrieve the information about a specific LUN.

Note: Refer to the NAND device's data sheet for a description of the specific enhanced status register bits.

Status Register

The status register provides information on the current or previous PROGRAM or ERASE operation. The various bits in the status register convey information and errors on the operation.

The status register is read by issuing the READ STATUS REGISTER command. The status register information is present on the output data bus (I/O0–I/O7). After the READ STATUS REGISTER command has been issued, the device remains in read status register mode until another command is issued. If a READ STATUS REGISTER command is issued during a random read cycle, a new READ command must be issued to continue with a PAGE READ operation.

Note: Refer to the NAND device's data sheet for a description of the status register bits.

Software Driver

A low-level driver (LLD) is software that provides a first abstraction layer of the hardware to free the upper software layer from hardware management.

The MT29F NAND software drivers, based on the following two layers, are easily portable on different hardware platforms.

- A *hardware-dependent layer* (nand_hw_if.h) that uses basic functions to manage NAND memory control signals.
- A *hardware-independent layer* (nand_MT29F_lld.h and nand_MT29F_lld.c) that uses command functions to control device operations such as BLOCK ERASE, PAGE PROGRAM, and READ. The NAND_Ret function shows whether an operation is successful: Successful = 0; Failed = 1. For a list and description of each command function, refer to the Doxygen documentation provided with the device driver.

C Library: Basic and OTP Functions

Table 3: Basic Data Types

typedef unsigned char	MT_uint8	8 bits unsigned
typedef signed char	MT_sint8	8 bits signed
typedef unsigned short	MT_uint16	16 bits unsigned
typedef signed short	MT_sint16	16 bits signed
typedef unsigned int	MT_uint32	32 bits unsigned
typedef signed int	MT_sint32	32 bits signed

Notes: 1. Data types used by the software drivers are described in the common.h header file.

Table 4: C Function Return Codes

NAND_SUCCESS	0x00	The operation completed successfully.
NAND_GENERIC_FAIL	0x10	The operation failed without more details.
NAND_BAD_PARAMETER_PAGE	0x20	The content of the parameter page is invalid.
NAND_INVALID_NAND_ADDRESS	0x30	The address provided in input is invalid, out-of-bound or invalid for the current operation.
NAND_INVALID_LENGHT	0x31	Length provided in input is invalid.
NAND_ERASE_FAILED	0x40	ERASE failed.
NAND_ERASE_FAILED_WRITE_PROTECT	0x41	ERASE failed due to write protect.
NAND_PROGRAM_FAILED	0x50	PROGRAM failed.
NAND_PROGRAM_FAILED_WRITE_PROTECT	0x51	PROGRAM failed due to write protect.
NAND_READ_FAILED	0x60	READ failed.
NAND_UNSUPPORTED	0xFD	The operation is not supported for the current device.
NAND_TIMEOUT	0xFE	The operation ended due to a timeout.
NAND_UNIMPLEMENTED	0xFF	The operation is not implemented in the driver.

Table 5: Basic Hardware-Dependent Layer Functions

Return Value	Function Name	Parameter		Function Description
		Name	Description	
void	PLATFORM_Init(void);	–	–	Init function initializes the platform during driver initialization.
void	PLATFORM_SetWriteProtect (void);	–	–	Sets the WRITE PROTECT signal ($\overline{WP}$) to LOW.
void	PLATFORM_UnsetWriteProtect (void);	–	–	Sets the WRITE PROTECT signal ($\overline{WP}$) to HIGH.
void	PLATFORM_WaitTime (int nanoseconds);	nanoseconds	Time to wait.	Waits a period of time equal to the value of the nanoseconds parameter.
void	PLATFORM_Open (void);	–	–	Programs the required settings before issuing a NAND operation.
void	PLATFORM_CommandInput (bus_t ubCommand);	ubCommand	Command to be issued to the NAND device.	Command latch.
void	PLATFORM_AddressInput (int ubAddress);	ubAddress	Address to be issued to the NAND device.	Address latch.
void	PLATFORM_DataInput (bus_t ubData);	ubData	Data to be written.	Input data latch.
bus_t[1]	PLATFORM_DataOutput (void);	ubAddress	Address to be read from.	Output data latch.
void	PLATFORM_Close (void);	–	–	–

Notes: 1. These basic hardware-dependent layer functions must be implemented by the user in accordance with the platform's behavior and details..
2. The bus_t format is either MT_uint8 for x8 bus width or MT_uint16 for x16 bus width.

Table 6: OTP Functions

Function Name	Function Description
NAND_OTP_MODE_ENTER	Puts the device in OTP mode, after which the following read and program operations refer only to the OTP area.
NAND_OTP_PAGE_PROGRAM()	Performs programming of a page in the OTP area.
NAND_OTP_PAGE_READ()	Reads a page in the OTP area.
NAND_OTP_SPARE_PROGRAM()	Performs programming of a page spare area while the device is in the OTP mode.
NAND_OTP_SPARE_READ()	Performs the reading of a spare area while the device is in OTP mode.
NAND_OTP_MODE_EXIT	Puts the device in normal mode, after which read and program operations refer to the normal NAND array.
NAND_OTP_MODE_PROTECT	Permanently blocks contents of the OTP, preventing bit manipulation operations.

Notes: 1. You cannot perform any ERASE operation while the device is in OTP mode.

Sample OTP Function Sequence

Using the OTP functions listed above, a correct sequence to program a page in the OTP area is the following:

1. Enter OTP mode (NAND_OTP_MODE_ENTER).
2. Program a page (NAND_OTP_PAGE_PROGRAM) and its spare (NAND_OTP_SPARE_PROGRAM).
3. Protect the page (NAND_OTP_MODE_PROTECT).
4. Exit OTP mode (NAND_OTP_MODE_EXIT).

Using the Driver

Choosing the Device

Before using the software on a target device, the user provides an implementation for the functions in nand_hw_if.h and sets the following define according to the platform.

Table 7: Device Constants Definition

Constant Name	Description
BUS_WIDTH_16BIT	Bus data width x16.
LITTLE_ENDIAN	The system is little endian.
BIG_ENDIAN	The system is big endian.
TIMEOUT_SUPPORT	If defined, the driver uses time.h to manage the timeout.
TIMEOUT_SECOND	If TIMEOUT_SUPPORT is set, this defines the second after which the timeout occurs.
CLOCKS_PER_SEC	If TIMEOUT_SUPPORT is set, this defines the number of ticks for a second.

Addressing Blocks and Pages

When a READ or PROGRAM operation must be performed, the software device driver receives as input parameters the address (Address) and the number (Length) of bytes/words to read/write from/to the memory.

The driver configures itself with data contained in the parameter page. The driver translates an address (contained in a nand_address_t type) in five address cycles used by the NAND device. For more information, refer to the Device and Array Organization section in the NAND data sheet.

Sample Code

To test the source code in the target system, start by reading from the MT29Fx device. If the device is erased, only 0xFFh data should be read.

Then, read the codes by issuing a NAND_Read_ID and check 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 function main() and include the header file of the LLD. The following example initializes the driver and reads the parameter page. It then erases a block and verifies that it is 0xFF. Then, it programs a page with an 0xBB pattern and verifies that the program is complete.

```
#include "common.h"
#include "nand_MT29F_lld.h"

#define MAX_BUF_SIZE 0x5000

int main() {
    MT_uint8 ret;

    param_page_t device_info;
    flash_width write_buf[MAX_BUF_SIZE];
    flash_width read_buf[MAX_BUF_SIZE];

    int main_area_size = 0, spare_area_size = 0;

    /* Lun = 0, Block = 10, Page = 10, Column = 0 */
    nand_addr_t addr = { 0, 10, 10, 0 };

    /* initialize driver */
    if(DRIVER_INITIALIZED == Init_Driver())
        printf("Driver initialized\n");
    else {
        printf("ERROR: Driver not initialized\n");
        return -1;
    }
    /* read parameter page */
    printf("Read parameter page\n");
    ret = NAND_Read_Param_Page(&device_info);
    if(NAND_SUCCESS != ret) {
```

```
    printf("NAND_Read_Param_Page FAIL! (exit code=%x)\n",
    ret);
    return 1;
    }
main_area_size = device_info.data_bytes_per_page;
spare_area_size = device_info.spare_bytes_per_page;

/* erase block */
printf("Erase\n");
ret = NAND_Block_Erase(addr);
if(NAND_SUCCESS != ret) {
    printf("NAND_Block_Erase FAIL! (exit code=%x)\n", ret);
    return -1;
}
/* read */
printf("Read\n");
ret = NAND_Page_Read(addr, read_buf, main_area_size);
if(NAND_SUCCESS != ret) {
    printf("NAND_Page_Read FAIL! (exit code=%x)\n", ret);
    return -1;
}
 /* (now block contains all 0xFF) */

printf("%x\n", read_buf[0]);

/* fill the write buffer with 0xBB pattern */
memset(write_buf, 0xBB, device_info.data_bytes_per_page);

/* program */
printf("Program\n");
ret = NAND_Page_Program(addr, write_buf, main_area_size);
if(NAND_SUCCESS != ret) {
    printf("NAND_Page_Program FAIL! (exit code=%x)\n", ret);
    return -1;
}
/* read */
printf("Read\n");
ret = NAND_Page_Read(addr, read_buf, main_area_size);
```

```
    if(NAND_SUCCESS != ret) {
        printf("NAND_Page_Read FAIL! (exit code=%x)\n", ret);
        return -1;
    }
     /* (now page contains all 0xBB) */

    printf("%x\n", read_buf[0]);

    /* erase block before end */
    printf("Erase\n");
    ret = NAND_Block_Erase(addr);
    if(NAND_SUCCESS != ret) {
        printf("NAND_Block_Erase FAIL! (exit code=%x)\n", ret);
        return -1;
    }
}
```

A correct sequence to program a page in the OTP area is the following:

1. Enter OTP mode (NAND_OTP_MODE_ENTER).
2. Program a page (NAND_OTP_PAGE_PROGRAM) and its spare (NAND_OTP_SPARE_PROGRAM).
3. Protect the page (NAND_OTP_MODE_PROTECT)
4. Exit OTP mode (NAND_OTP_MODE_EXIT)

Revision History

Rev. B 05/13

Added OTP information

Rev. A 07/12

- Initial release