

Technical Note

Software Device Drivers for Small Page Micron® NAND Flash Memory

Introduction

This technical note describes how to use the Micron® small page NAND Flash memory software drivers. These drivers are low-level drivers (LLD) that manage the hardware functionality of small page NAND Flash memory devices. It also 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 be compiled and run. In addition, this technical note explains how to modify the source code for a target system. The source code is available from www.micron.com or from your Micron distributor. The c2194.c and c2194.h files contain libraries for accessing the Flash devices.

This technical note does not replace the NANDxxx-A, NAND01GWxA2B-KGD, NANDxxxxxA2D, and NANDxxxxxA2C single-level cell (SLC) small page NAND Flash memory family data sheets. The data sheets are referred to throughout this document and it is necessary to have a copy to follow some explanations. For more information on the small page NAND drivers, contact your Micron representative.

Small Page NAND Overview

Small page NAND is a family of nonvolatile Flash memory devices that use SLC NAND cell technology. The devices range from 128Mb to 1Gb and operate with either a 1.8V or 3V voltage supply. The size of a page is either 528 bytes (512 + 16 spare) or 264 words (256 + 8 spare), depending on whether the device has a x8 or x16 bus width. 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. Each block can be programmed and erased over 100,000 cycles. To extend the lifetime of NAND Flash devices, it is strongly recommended that an error correction code (ECC) be implemented. A write protect (WP) pin is available to provide hardware protection against unwanted PROGRAM and ERASE operations.

The devices feature an open-drain ready/busy output that is used to identify whether the PROGRAM/ERASE/READ (P/E/R) controller is currently active. The use of an open-drain output enables the ready/busy pins from several memory devices to be connected to single pull-up resistors. A COPY BACK 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.

For the NAND Flash family, there is a serial number that enables each device to be uniquely identified. The serial number option is subject to a non-disclosure agreement (NDA). As a result, it is not described in the data sheet. For more details on this option, contact your Micron representative.

Bus Operations

There are six standard bus operations that control the memory:

- **COMMAND INPUT:** Gives commands to the memory. Commands are accepted when chip enable is LOW, command latch enable is HIGH, address latch enable is LOW, and read enable is HIGH. They are latched on the rising edge of the write enable signal. Only I/O0 to I/O7 are used to input commands.
- **ADDRESS INPUT:** Inputs the memory address. Three bus cycles are required to input the addresses for 128Mb and 256Mb devices, and four bus cycles are required to input the addresses for 512Mb and 1Gb devices (refer to Table 2 on page 3 and Table 3 on page 3). The addresses are accepted when chip enable is LOW, address latch enable is HIGH, command latch enable is LOW, and read enable is HIGH. They are latched on the rising edge of the write enable signal. Only I/O0 to I/O7 are used to input addresses.
- **DATA INPUT:** Inputs the data to be programmed. Data is only accepted when chip enable is LOW, address latch enable is LOW, command latch enable is LOW, and read enable is HIGH. The data is latched on the rising edge of the write enable signal. It is then input sequentially using the write enable signal.
- **DATA OUTPUT:** Reads the electronic signature, serial number, and data in the memory array and status register. Data is output when chip enable is LOW, write enable is HIGH, address latch enable is LOW, and command latch enable is LOW. The data is output sequentially using the read enable signal.
- **WRITE PROTECT:** Protects the memory against PROGRAM or ERASE operations. When the write protect signal is LOW the device will not accept PROGRAM or ERASE operations and as a result the contents of the memory array cannot be altered. The write protect signal is not latched by write enable to ensure protection, even during power-up.
- **STANDBY:** When chip enable is HIGH the memory enters into standby mode, the device is deselected, outputs are disabled, and power consumption is reduced.

Each of these is described in this section Table 1. For information on timing requirements, refer to the following SLC small page NAND Flash memory family data sheets:

- NANDxxx-A
- NAND01GWxA2B-KGD
- NANDxxxxxA2D
- NANDxxxxxA2C

Table 1: Bus Operations

Bus Operation	\bar{E}	AL	CL	\bar{R}	\bar{W}	\bar{WP}	I/O0 - I/O7	I/O8 - I/O15 ¹
COMMAND INPUT	V_{IL}	V_{IL}	V_{IH}	V_{IH}	Rising	X ²	Command	X
ADDRESS INPUT	V_{IL}	V_{IH}	V_{IL}	V_{IH}	Rising	X	Address	X
DATA INPUT	V_{IL}	V_{IL}	V_{IL}	V_{IH}	Rising	X	Data Input	Data Input
DATA OUTPUT	V_{IL}	V_{IL}	V_{IL}	Falling	V_{IH}	X	Data Output	Data Output
WRITE PROTECT	X	X	X	X	X	V_{IL}	X	X
STANDBY	V_{IH}	X	X	X	X	X	X	X

- Notes:**
1. Only for x16 devices.
 2. WP must be V_{IH} when issuing a PROGRAM or ERASE command.

Table 2: Address Insertion, x8 Devices^{1, 2}

Bus Cycle	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
First	A7	A6	A5	A4	A3	A2	A1	A0
Second	A16	A15	A14	A13	A12	A11	A10	A9
Third	A24	A23	A22	A21	A20	A19	A18	A17
Fourth ³	V _{IL}	V _{IL}	V _{IL}	V _{IL}	V _{IL}	V _{IL}	A26	A25

- Notes:
1. A8 is set LOW or HIGH by the 00h or 01h command.
 2. Any additional address input cycles will be ignored.
 3. The fourth cycle is only required for 512Mb and 1Gb devices.

Table 3: Address Insertion, x16 Devices^{1, 2, 3}

Bus Cycle	I/O8 - I/O15	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
First	X	A7	A6	A5	A4	A3	A2	A1	A0
Second	X	A16	A15	A14	A13	A12	A11	A10	A9
Third	X	A24	A23	A22	A21	A20	A19	A18	A17
Fourth ⁴	X	V _{IL}	V _{IL}	V _{IL}	V _{IL}	V _{IL}	V _{IL}	A26	A25

- Notes:
1. A8 is “Don’t Care” in x16 devices.
 2. Any additional address input cycles will be ignored.
 3. The 01h command is not used in x16 devices.
 4. The fourth cycle is only required for 512Mb and 1Gb devices.

Operations

The operations are:

- **RANDOM READ:** Each time the operation is issued, the first read is RANDOM READ.
- **PAGE READ:** After the RANDOM READ access, the page data is transferred to the page buffer. Once the transfer is complete, the data can then be read out sequentially from the selected column address to the last column address.
- **PAGE PROGRAM:** This is the standard operation to program data to the memory array. The main area of the memory array is programmed by page. However, partial page programming is allowed where any number of bytes (for example, 1 to 528) or words (for example, 1 to 264) can be programmed. The maximum number of consecutive PARTIAL PAGE PROGRAM operations allowed in the same page is three. After exceeding this number, a BLOCK ERASE command must be issued before any further PROGRAM operations can take place on that page.
- **COPY BACK PROGRAM:** Copies data stored in one page and reprograms it in another page. The COPY BACK PROGRAM operation does not require external memory and as a result the operation is faster and more efficient because reading and loading cycles is not required. However, as the standard external ECC cannot be used with the COPY BACK operation, bit errors due to charge loss cannot be detected.
- **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. An ERASE operation consists of three steps:
 1. One bus cycle for setting the BLOCK ERASE command.
 2. Three bus cycles are required for 512Mb and 1Gb devices, while two bus cycles are required for 128Mb and 256Mb devices to input the block address.

3. One bus cycle is required to issue the CONFIRM command.
- **RESET:** Used to reset the command interface and status register. If the RESET operation is issued during any operation, the operation will be aborted. If a PROGRAM or ERASE operation 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 ELECTRONIC SIGNATURE:** Enables information to be read from the device, such as manufacturer code and device code.

Detailed Example

The Commands table in the NANDxxx-A, NAND01GWxA2B-KGD, NANDxxxxxA2D and NANDxxxxxA2C data sheets describes the WRITE operation sequences recognized as valid commands by the PROGRAM/ERASE controller.

For the NAND 512Mb device x8 bus of page 19h of the block 20h, the required C language sequence is the following:

1. One bus cycle is required to setup the PAGE PROGRAM (SEQUENTIAL INPUT) command (see Table 4):

```
NAND_CommandInput((NMX_uint8)0x80);
```

2. Three or four bus cycles are required to input the program address.

```
NAND_AddressInput(0x00); /* first address byte */
```

```
NAND_AddressInput(0x19); /* second address byte */
```

```
NAND_AddressInput(0x04); /* third address byte */
```

```
NAND_AddressInput(0x00); /* fourth address byte */
```

3. The data is loaded into the data registers.

```
for (j=0; j<PageSize; j++) {
    NAND_DataInput(data[j]);
}
```

4. One bus cycle is required to issue the PAGE PROGRAM confirm command to start the P/E/R controller. The P/E/R only starts if the data has been loaded in step 3.

```
NAND_CommandInput((NMX_uint8)0x10);
```

5. The P/E/R controller programs the data into the array.

Table 4: x8 NAND01G-A: Page 19h – Block 20h

Bus Cycle	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
1	0	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0	1
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0

- Notes:**
1. White cells are not specified.
 2. Green cells indicate the column address.
 3. Yellow cells indicate the page address
 4. Blue cells indicate the block address.

Once the PROGRAM operation starts, the status register can be read using the READ STATUS REGISTER command. During PROGRAM operations, the status register only flags errors for bits set to 1 that have not been successfully programmed to 0. During the PROGRAM operation, only the READ STATUS REGISTER and RESET commands are accepted. All other commands are ignored. Once the PROGRAM operation has completed the P/E/R controller, the bit SR6 is set to 1 and the ready/busy signal goes HIGH.

The device remains in read status register mode until another valid command is written to the command interface.

Software Driver

In general, LLD software provides a first abstraction layer of the hardware to free the upper software layer from the hardware management. The small page NAND software drivers enable easy migration to different hardware platforms. The drivers are based on two layers:

1. **Hardware Dependent Layer**, which uses basic functions to manage the NAND control signals.
2. **Hardware Independent Layer**, which uses NAND command functions to implement device operations such as BLOCK ERASE, PAGE PROGRAM, and READ.

Porting the drivers to adapt the hardware dependent layer to a new platform requires only a few steps.

C Library Functions

Basic Data Types

Data types used by the software driver are listed in Table 5.

Table 5: Basic Data Types

typedef unsigned char	NMX_uint8	8 bits unsigned
typedef signed char	NMX_sint8	Initial release
typedef unsigned short	NMX_uint16	8 bits signed
typedef signed short	NMX_sint16	16 bits unsigned
typedef unsigned int	NMX_uint32	16 bits signed
typedef signed int	NMX_sint32	32 bits unsigned

NAND Basic Functions – Hardware Dependent Layer

The basic functions of the hardware dependent layer are listed in Table 6.

Table 6: Basic NAND Functions

Return Value	Function Name	Parameter	Parameter Description	Function Description
void	NAND_Open(void);	–	–	–
void	NAND_CommandInput (NMX_uint8 ubCommand);	ubCommand	Command to issue to NAND.	Implements a command latch cycle.
void	NAND_AddressInput (NMX_uint8 ubAddress);	ubAddress	Address to issue to NAND.	Implements an address latch cycle.
void	NAND_DataInput (dataWidth ubData);	ubData	Data to write to NAND.	Implements a data input latch cycle.

Table 6: Basic NAND Functions (Continued)

Return Value	Function Name	Parameter	Parameter Description	Function Description
dataWidth	NAND_DataOutput(void);	ubAddress	Address of data to be output.	Implements a data output latch cycle.
void	NAND_Close(void);	–	–	–

NAND Command Functions – Hardware Independent Layer

The command functions of the hardware independent layer are listed in Table 7. The NAND_Ret value shows whether the operation was successful. A value is of 0 indicates that the operation was successful. A value of 1 indicates that the operation failed.

Table 7: Command Functions

Return Value	Function Name	Parameter	Parameter Description	Function Description
NAND_Ret	NAND_BlockErase (NMX_uint32 udAddress);	udAddress	The address of the block to erase.	Implements a BLOCK ERASE.
NAND_Ret	NAND_CopyBack (NMX_uint32 udSourceAddr, NMX_uint32 udDestinationAddr);	udSourceAddr	Address of the source page to copy.	Implements a COPY BACK operation.
		udDestinationAddr	Address of the destination page.	–
NAND_Ret	NAND_PageRead (NMX_uint32 udAddresses, dataWidth *Buffer, udword *udLength);	udAddresses	Addresses in the page where read.	Implements a PAGE READ operation.
		Buffer	Destination buffer where store the data.	–
		udLength	Lengths of the data to read.	–
NAND_Ret	NAND_PageProgram (NMX_uint32 udAddresses, dataWidth *Buffer, NMX_uint32 udLength);	udAddresses	Addresses into the page to program.	Implements a PAGE PROGRAM operation.
		Buffer	Source buffers with the data to program.	–
		UdLength	Length of the data pieces to program.	–
void	NAND_ReadElectronicSignature (dataWidth * Buffer);	Buffer	Buffer where store the read data.	Reads the electronic signature.
void	NAND_Reset (void);	–	–	–
NAND_Ret	NAND_SpareProgram (NMX_uint32 udAddress, dataWidth *Buffer, NMX_uint32 udLength);	udAddress	Address into the page to program.	Programs in the spare area of the NAND Flash array.
		Buffer	Source buffer with the data to program.	–
		UdLength	Length of the data piece to program.	–
NAND_Ret	NAND_SpareRead (NMX_uint32 udAddress, dataWidth *Buffer, NMX_uint32 udLength);	udAddress	Address in the page where read.	Reads from the spare area of the NAND Flash array.
		Buffer	Destination buffer to store the data.	–
		UdLength	Length of the data pieces to read.	–
ubyte	NAND_ReadStatusRegister (void)	–	–	Allows access to the status register.

Return Code

The functions of the software driver return values are listed in Table 8.

Table 8: Command Function Return Code

typedef ubyte NAND_Ret		
Type	Value	Description
#define NAND_PASS	0x00	The operation on NAND completed successfully.
#define NAND_FAIL	0x01	The operation on the NAND failed.
#define NAND_FLASH_SIZE_OVERFLOW	0x02	The address is not within the device.
#define NAND_PAGE_OVERFLOW	0x04	Attempt to access more than one page.
#define NAND_WRONG_ADDRESS	0x08	The address is incorrect.

Using the Driver

The driver supports all devices belonging to the small page NAND Flash memory family (see the NANDxxx-A, NAND01GWxA2B-KGD, NANDxxxxxA2D, and NANDxxxxxA2C data sheets).

Choosing the Device

Before using the software on the target device, a device must be chosen.

The device to be used is specified by choosing the relative definition. For example, when choosing the NAND512RW3A device, it is sufficient to add the line:

```
#define NAND512RW3A
```

Note: Refer to internal software documentation for a detailed list of supported devices.

The definition sets the constants listed in Table 9:

Table 9: Device Constants

Constant Name	Description
FLASH_WIDTH	Data width of the NAND Flash. DataWidth is either ubyte or uword depending on whether the device is x8 or x16 bus width.
FLASH_SIZE	NAND size (without spare area).
PAGE_SIZE	Size of main area of page.
PAGE_DATA_SIZE	Size of data area of a page.
PAGE_SPARE_SIZE	Size of spare area of a page.
NUM_BLOCKS	Number of blocks.

Note: See the NANDxxx-A, NAND01GWxA2B-KGD, NANDxxxxxA2D and NANDxxxxxA2C SLC small page NAND Flash memory data sheets for detailed descriptions of the devices.

Enabling DMA

Direct memory access (DMA) can increase the application's performance. The READ/ WRITE operations from and to the NAND device and the speed of the data transfer between the NAND page buffer and the RAM can be increased by enabling DMA.

The DMA engine is enabled by setting the following definitions:

```
#define DMA_ENABLE
```

According to the hardware environment, the code to manage the DMA engine must be inserted at every occurrence of the following definition:

```
#ifdef DMA_ENABLE
int i;
*((volatile unsigned int*) (GDMSRC0)) = (unsigned int)Buffer;
*((volatile unsigned int*) (GDMADST0)) = Base_Address;
*((volatile unsigned int*) (GDMACNT0)) = udlength;
*(volatile unsigned int*) (GDMACON0) = 0x0041;
/*Wait for DMA busy*/
do
i= *((volatile unsigned int*) (GDMACON0));
while( (i&(0x2))!=0);
}
#else
/* Write the data to the internal buffer */
for (udIndex=0;udIndex<udlength;udIndex++)
NAND_DataInput(Buffer[udIndex]);
#endif
```

Addressing Blocks and Pages

Setting Address and Length Parameters in NAND Command Functions

When issuing READ or OPERATIONS operations, the Micron software driver routine receives the address as the input parameter and the number of bytes/words to be read/written:

Figure 1: x8 Device

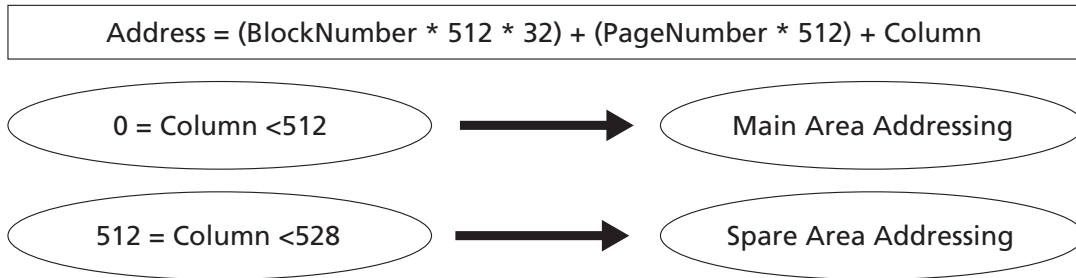


Figure 2: x16 Device

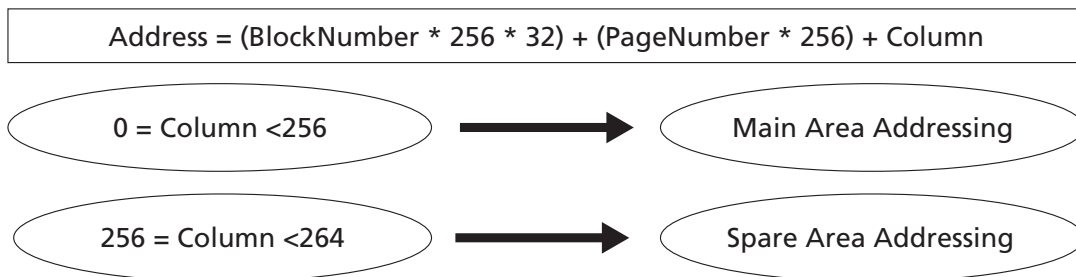


Figure 3: Address Example: x8 Device, Main Area – Area A

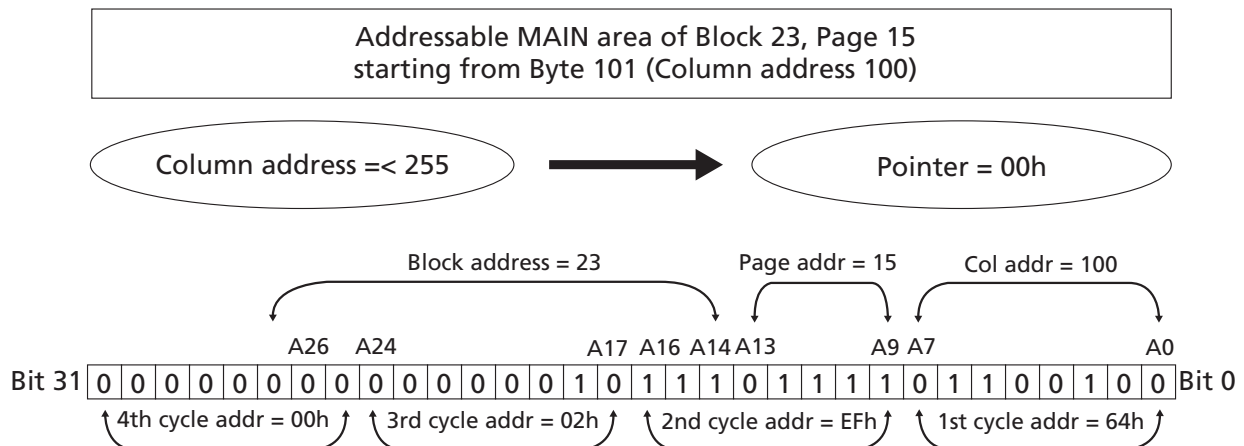


Figure 4: Address Example: x8 Device, Main Area – Area B

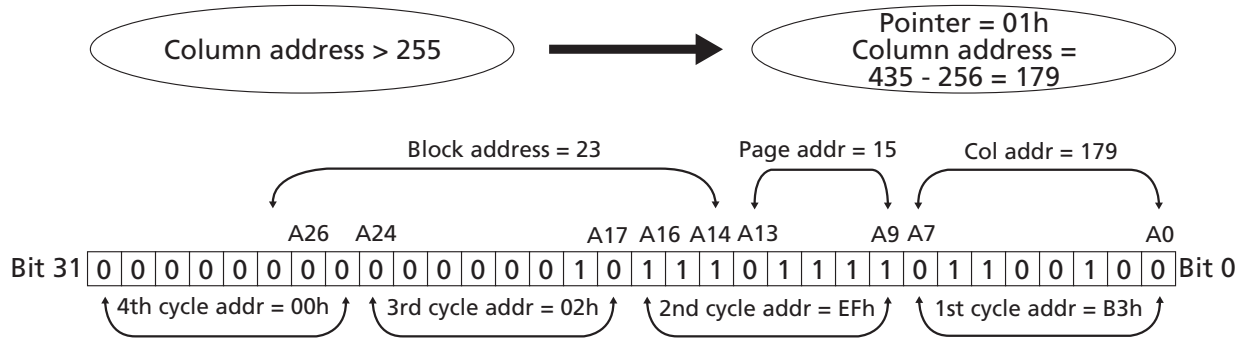


Figure 5: Address Example: x8 Device, Main Area – Area C

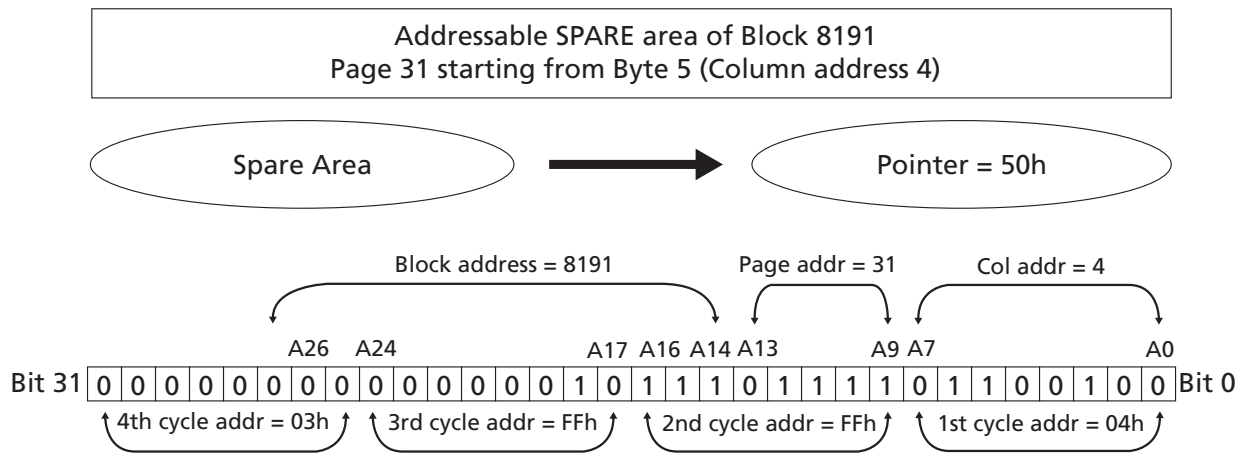
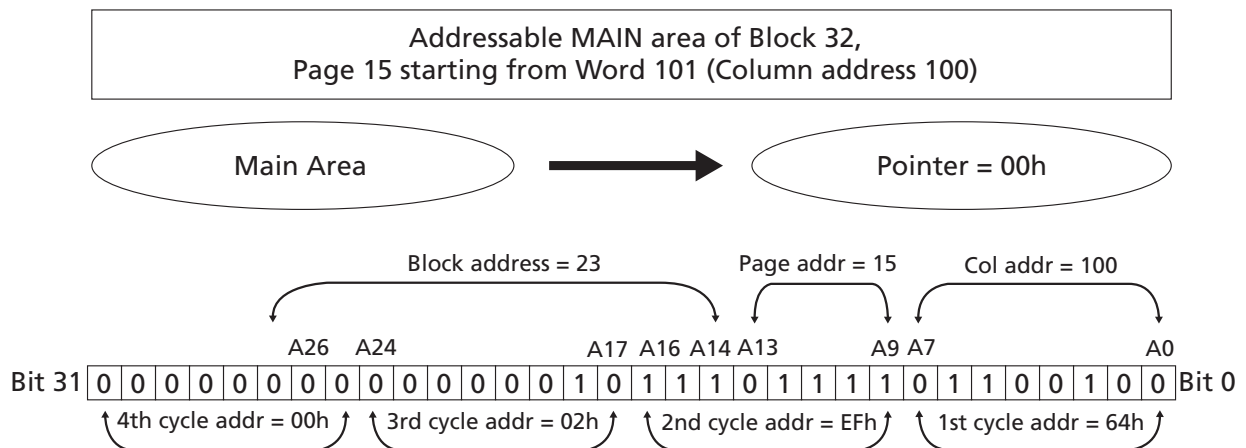


Figure 6: Address Example: x16 Device, Main Area – Area A - B



Note: For x 16 devices, the addressable areas A and B are the same.

Getting Started (Example Quick Test)

To test the source code in the target system, start by reading from the small page NAND device. If it is erased, only FFh data should be read. Next, manufacturer and device codes are read and checked. If the functions work, the other functions are also likely to work. However, all the functions should be tested thoroughly.

To start, write a function main() and include the header file of the LLD. All Flash memory functions can then be called and executed within the main() function.

The following example shows a check of the device identifiers (device code, manufacturer code), BLOCK ERASE command, and a PROGRAM main spare.

```
void main (void) {
    dataWidth DeviceID[4]; /*buffer for read manufacturer and
    devicecodes/*
    dataWidth BufferMainSpare[PAGE_DATA_SIZE+ PAGE_SPARE_SIZE];
    /*buffer for program*/
    NMX_uint32 address;
    NMX_uint32 n_block,n_page;
    NMX_uint32 udlength;
    InitBoard(); /* initialize the board*/

    memset (DeviceID, 0xFF,4); /*set the buffer to 0xFF */
    printf("Read Electronic Signature \n");
    NAND_ReadElectronicSignature(DeviceID); /* read the manufactures
    and device codes */printf("Manufacturer Code: %x\n", Devi-
    ceID[0]);
    printf("Device Code: %x\n", DeviceID[1];
    printf("Reserved Byte 3: %x\n", DeviceID[2]);
    printf("Byte 4: %x\n", DeviceID[3]);
    24
    n_block = 10; /* block 10 will be erased */
    address = BLOCK_2_ADDRESS(n_block); /* address of block 10*/
    NAND_BlockErase(address); /* erase of block 10*/

    n_block = 10;
    n_page =3; /*address of block 10 page 3*/
    address= BLOCK_2_ADDRESS(n_block)+(n_page * PAGE_DATA_SIZE);
    udlength= PAGE_DATA_SIZE+ PAGE_SPARE_SIZE; /*length of data
    to write*/

    memset (BufferMainSpare, 0xBB, udlength); /*initialize the
```

```

buffer to program to 0xBB*/
/*program main and spare area of page 3 in the block 10*/
NAND_PageProgram (address, BufferMainSpare, udlength);
}

```

Porting the Software Driver

To port the software driver for a particular platform, only the hardware dependent layer functions must be rewritten to manage the signals of NAND device connected to the platform.

The *How to Connect a Single Level Cell NAND Flash Memory to a Generic SRAM Controller* technical note describes how to use the GPIO signals that pilot the NAND device.

For example, the basic hardware dependent command input comprises two steps:

1. Set AL and CL signals by setting the GPI/O 0 (LOW) and the GPI/O 1 (HIGH).
2. Set the E and W signals with a WRITE or READ operation to the bank where the NAND Flash is mapped.

```

void NAND_CommandInput (ubyte ubCommand)
{
    NMX_uint8 * udAddress = (NMX_uint8*) Base_Address;
    /* Set Op mode Command (AL=0,CL=1) */
    *(GPIODATA)=BASE_IO_DATA|ASSERT_CL|WRITE_PROTECT_BIT;
    *(udAddress) = ubCommand; /*Transfer to NAND ubCommand */
    *(GPIODATA)=BASE_IO_DATA|WRITE_PROTECT_BIT; /*UnSet Op mode
    (AL=0,CL=0) */
}

```

The *How to Connect a Small Page NAND Flash Memory to an ARM7TDMI Core Based Microcontroller* technical note describes how to connect the NAND Flash memory to an ARM7DMI core based microcontroller with GLUE Logic. For example, the basic hardware dependent command consists of three steps:

1. **Setting the Flip-Flop:** The NAND E and CL signals are set by a dummy write to the bank area on which the flip-flop is mapped. The flip-flop latches the KS32C50100 signals on the falling edge of W and holds the NAND control signals during the memory operations. The NAND control signal status depends on the value of the least significant address (A1-A4) of KS32C50100 latched by the flip-flop.
2. **Issue a Command to the Flash:** The device is ready to accept a command. The command is issued with a WRITE operation to the NAND device. The A0 address controls the operation on the device and the NAND latches the command on the rising edge of W as described in the NANDxxx-A, NAND01GWxA2B-KGD, NANDxxxxxA2D, and NANDxxxxxA2C SLC small page NAND Flash memory data sheets.
3. **Resetting the Flip-flop:** Resetting the flip-flop resets the CL signal. When setting the flip-flop, the flip-flop is reset by a “dummy” write to the flip-flop address with A2 set to VIL.

```

void NAND_CommandInput (ubyte ubCommand)
{

```

```
NMX_uint32 udAddress;
/* Setting the address to assert on the ubChip the CL and
control the Write Protect of the glue logic */
udAddress = (Base_Address|ASSERT_CL|WRITE_PROTECT_BIT);
__asm{
    MOV R1,udAddress /* Load the address in R1 */
    MOV R2,#0
    STRB R2,[R1,#0] /* Setting the glue logic*/
}
/* Setting the address to issue the command*/
udAddress = (Base_Address|DATA_OFFSET);
__asm{
    MOV R1,udAddress /* Load the address in R1 */
    MOV R2,ubCommand /* Load the command in R2 */
    STRB R2,[R1,#0] /* Issue the command*/
}
/* Setting the address to reset the glue logic except E */
udAddress = (Base_Address|RESET_LOGIC|WRITE_PROTECT_BIT);
__asm{
    MOV R1,udAddress /* Load the address in R1 */
    MOV R2,#0
    STRB R2,[R1,#0] /* reset the glue logic except E */
```

References

- NANDxxx-A, NAND01GWxA2B-KGD, NANDxxxxxA2D, and NANDxxxxxA2C SLC small page NAND Flash memory data sheets
- *How to Connect a Single-level Cell NAND Flash Memory to a Generic SRAM Controller* technical note
- *How to Connect a Small Page NAND Flash Memory to an ARM7TDMI Core Based Microcontroller* technical note

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.