Technical Note

Software Device Drivers for Very Large Page Micron® NAND Flash

Introduction

This technical note explains how to use Micron's very large page NAND Flash memory software device drivers. These drivers are the low-level drivers (LLD) that manage the hardware functionality of the very large page NAND Flash memory devices.

This technical note 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 compile and run. The technical note 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. The files contain libraries for accessing the devices.

Refer to the data sheets for the corresponding part numbers and densities (see "References" on page 21). This technical note does not replace the Micron NAND Flash memory device data sheet. It refers to the data sheet throughout, and it is necessary to have a copy of it to follow some of the explanations.

For more information about Micron NAND Flash memory software device drivers, please contact your Micron representative.

Very Large Page NAND Overview

The NAND Flash 4224-byte page is a family of nonvolatile Flash memory devices that use NAND cell technology. The devices range from 1Gb to 8Gb, and operate either from a 1.8V or a 3V voltage supply. The size of a page is 4224 bytes for devices with either 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.

The devices have the following hardware and software security features:
- A write protect pin ($\overline{\text{WP}}$) enables hardware protection against PROGRAM and ERASE operations.
- A block locking scheme provides 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 enables 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.

All devices have the "Chip Enable Don't Care" feature, which enables code to be directly downloaded by a microcontroller, as chip enable transitions during the latency time do not stop the READ operation.

Micron NAND Flash 4224 byte page memory devices can be delivered with an optional unique identifier (serial number), which allows 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.

Micron's very large page NAND devices have a hardware mechanism to ensure data protection. A write protect pin is available to provide hardware protection against PROGRAM and ERASE operations.

Note: The PRL pin may not be supported. Please refer to Micron NAND data sheets to check if this functionality is supported.

Bus Operations

The standard bus operations that control Micron's very large page NAND Flash memory devices are:
- **COMMAND INPUT**: Sends commands to the memory device.
- **ADDRESS INPUT**: Inputs the memory addresses. Four bus cycles are required to input the addresses for 1Gb devices, whereas five bus cycles are required for 8Gb devices. Refer to the Micron very large page 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**: Enables data to be read from the memory array, the status register content, a block lock status, the electronic signature, or the 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.
- **STANDBY**: When chip enable is HIGH, the memory enters standby mode, the device is deselected, outputs are disabled, and power consumption is reduced.

Device Operations

All bus write operations to the device are interpreted by the command interface. The following commands are available:
- **PAGE READ**: After the first random read access, the page data (4224 bytes) is transferred to the page. Once the transfer is complete, the data can be read out sequentially from the selected column address to the last column address.
- **MULTI-PLANE PAGE READ**: This operation is an extension of a PAGE READ operation for a single plane. Since the device is equipped with two memory planes, a read of two pages (one for each plane) is enabled by activating two sets of 4224-byte page registers (one for each plane). A multi-plane page read operation requires the following steps:
  1. Serially load up to two pages of data (4224 bytes) into the data buffer.

2. The device can output random data from the device buffer. The RANDOM DATA OUTPUT command can be used to skip some data during a sequential data output.

- **RANDOM DATA OUTPUT**: The device can output random data in a page (instead of consecutive sequential data) by issuing this command. The RANDOM DATA OUTPUT command can be used to skip some data during a sequential data output.

- **CACHE READ**: Used to improve the read throughput by reading data using the cache register. As soon as 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 (4224) can be programmed.

- **MULTI-PLANE PAGE PROGRAM**: Issues a MULTIPLANEPAGEPROGRAM command, which enables the programming of two pages in parallel, one in each plane. It can perform a multi-plane page program with random data input. A MULTI-PLANE PAGE PROGRAM operation requires the following two steps:
  1. Serially load up to two pages of data (8448 bytes) into the data buffer.
  2. Parallel programming of both pages starts after the issue of the PAGE CONFIRM command.

  If the MULTI-PLANE PAGE PROGRAM 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 Micron very large page NAND Flash memory 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**: Used to copy the data stored in one page and reprogram 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 COPY BACK PROGRAM**: Issues a MULTI-PLANE COPYBACK command as explained in the data sheet of the 4224 byte page family. 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 COPY BACK PROGRAM fails, an error is signaled on bit SR0 of the status register. Refer to the Micron very large page NAND Flash memory data sheet for a description of the multi-plane operation and specific information related to the MULTI-PLANE COPY BACK PROGRAM sequence.

- **CACHE PROGRAM**: Used to improve the programming throughput by programming data using the cache register. The CACHE PROGRAM operation can only be used within one block. The cache register enables 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 enables the erasing of two blocks in parallel, one in each plane. 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 Micron very large page NAND Flash memory data sheet for a description of the multi-plane operation and specific information related to the block erase sequence.

- **RESET**: Used to reset the command interface and status register. If the RESET command is issued during any operation, the operation is 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 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 command.
- **READ ELECTRONIC SIGNATURE**: This device operation enables device information, such as manufacturer code and device code, to be read. Refer to the very large page Micron NAND Flash memory data sheet for a description of the electronic signature values.

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 is issued, the device remains in read status register mode until another command is issued. Therefore, 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 very large page Micron NAND Flash memory data sheet for a description of the status register bits.

Detailed Example

The Commands table provided in the very large page NAND Flash memory data sheet describes the WRITE operation sequences that are recognized as valid commands by the PROGRAM/ERASE controller.

For example, with a device configured in an 8-bit bus width, programming page 19h of block 20h requires the following C sequence to be issued:

1. The first bus write cycle sets up the PAGE PROGRAM command (command code 80h). By default, data is input sequentially. This is done by sending the following C command to the memory:

```
NAND_CommandInput((NMX_uint8)0x80); /* command code */
```

2. Four or five bus cycles are required to input the program address. The cycle sequence that must be issued to program page 19h of block 20h is described in Table 1.

```
InsertAddress(address); /* This function perform the data
insertion in 5 cycles(row & column address) */
```

3. The data is loaded into the data registers:

```
for ( j=0; j<PageSize; j++) {

NAND_DataInput(data[I]);

}
```

4. One bus cycle is required to issue the PAGE PROGRAM CONFIRM command (command code 10h) to start the P/E/R Controller. The P/E/R only starts if the data has been loaded in step 3.

```
NAND_CommandInput((ubyte)0x10);
```

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

Once the PAGE PROGRAM operation has started, 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 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.

Table 1: Page and Block Address Setting Sequence

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	V_{IL}	V_{IL}	V_{IL}	A12	A11	A10	A9	A8
Third	A20	A19	A18	A17	A16	A15	A14	A13
Fourth	A28	A27	A26	A25	A24	A23	A22	A21
Fifth	V_{IL}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	A31	A30	A29

Software Driver

LLD software provides a first abstraction layer of the hardware to free the upper software layer from hardware management. The very large page NAND software drivers are structured to be easily portable on different hardware platforms. They are based on two layers:

- A *hardware dependent layer* that uses basic functions to manage the NAND memory control signals.
- A *hardware independent layer* that uses command functions to control device operations such as BLOCK, ERASE, PAGE PROGRAM, and READ.

C Library Functions

Basic Data Types

The data types used by the software drivers are described in Table 2.

Table 2:   Data Types

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

Return Codes

The codes that can be returned by the software driver's C functions are described in Table 3.

Table 3:   Return Codes

typedef NMX_uint8 NAND_Ret	–	–
#define NAND_PASS	0x00	The operation on the NAND device completed successfully.
#define NAND_FAIL	0x01	The operation on the NAND device 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 wrong.
#define NAND_DIFFERENT_PAGES	0x10	The page is different in the data input command.
#define NAND_WRITE_PROTECTED	0x80	The device is write protected.
#define NAND_UNLOCKED_BLOCK	0x09	Block unlocked.
#define NAND_LOCKED_BLOCK	0x06	Block locked.
#define NAND_LOCK_DOWN	0x05	Block lock-down.
#define CACHE_READ_NOT_POSSIBLE	0x07	Required cache read data output with cache read not pending.
NAND_FIRSTPLANE_FAIL	0x11	The operation failed on the first plane.
NAND_SECONDPLANE_FAIL	0x13	The operation failed on the second plane.
NAND_BOTHPLANE_FAIL	0X12	The operation failed on both planes.

Hardware Dependent Layer: Basic functions

The basic functions of the hardware dependent layer are listed in Table 4.

Table 4:   Basic Functions

Return Value	Function Name	Parameter		Function Description
		Name	**Description**	
void	NAND_SetWriteProtect (void);	–	–	Sets the write protect signal ($\overline{WP}$) to LOW.
void	NAND_UnsetWriteProtect (void);	–	–	Sets the $\overline{WP}$ to HIGH.
void	NAND_WaitTime (NMX_uint8 nanoseconds);	nanoseconds	Time to wait.	Waits a period of time equal to the value of the nanoseconds parameter.
void	NAND_Open (void);	–	–	Programs the required settings before issuing a NAND operation.
void	NAND_CommandInput (NMX_uint8 ubCommand);	ubCommand	Command to be issued to the NAND device.	Command latch.
void	NAND_AddressInput (NMX_uint8 ubAddress);	ubAddress	Address to be issued to the NAND device.	Address latch.
void	NAND_DataInput (dataWidth ubData);	ubData	Data to be written.	Input data latch.
dataWidth[1]	NAND_DataOutput (void);	ubAddress	Address to be read from.	Output data latch.
void	NAND_Close (void);	–	–	Programs the required settings after issuing a NAND operation.

Notes: 1. The dataWidth format is either NMX_uint8 for x8 bus widths or NMX_uint16 for x16 bus widths.

Hardware Independent Layer: Command functions

The command functions of the hardware independent layer are listed in Table 5.

The NAND_Ret function shows whether the operation was successful. When its value is 0, the operation was successful. When its value is 1, the operation failed.

Table 5: Command Functions

Return Value	Function Name	Parameter	Parameter Description	Function Description
NAND_Ret	**NAND_MultiplanePageRead** (NMX_uint32 *FPudAddress, dataWidth *FPBuffer, NMX_uint32 *SPudAddress, dataWidth *SPBuffer, NMX_uint8 numOfChunks, NMX_uint16 *chunkSize)	FPudAddress	First plane addresses to be read.	Performs a MULTI-PLANE PAGE READ with random data output.
		FPBuffer	Source buffers containing the data to be read.	
		udAddress	Second plane addresses to be read.	
		FPBuffer	Source buffers containing the data to be read.	
		SPBuffer	Source buffers containing the data to be programmed.	
		numOfChunks	Number of addresses contained in udAddresses.	
		ChunkSize	Length of the data pieces to be programmed.	

8

Table 5:   Command Functions (Continued)

Return Value	Function Name	Parameter	Parameter Description	Function Description
NAND_Ret	**NAND_MultiplaneCopyBack** (NMX_uint32 *FPudSourceAddr, NMX_uint32 *FPudDestinationAddr, dataWidth *FPBuffer, NMX_uint16 FPnumOfChunks, NMX_uint16 *FPchunkSize, NMX_uint32 *SPudSourceAddr, NMX_uint32 *SPudDestinationAddr, dataWidth *SPBuffer, NMX_uint16 SPnumOfChunks, NMX_uint16 *SPchunkSize);	FPudSourceAddress	First plane addresses to be read.	Performs a MULTI-PLANE COPY BACK operation.
		FPudDestinationAddress	First plane addresses to be programmed.	
		FPBuffer	Source buffers containing the data to be read.	
		FPnumOfChunks	Number of addresses contained in the first plane.	
		FPChunkSize	Length of the data pieces to be programmed.	
		SPudSourceAddress	Second plane addresses to be read.	
		SPudDestinationAddress	Second plane addresses to be programmed.	
		SPBuffer	Source buffers containing the data to be read.	
		SPnumOfChunks	Number of addresses contained in the second plane.	
		SPChunkSize	Length of the data pieces to be programmed.	
NAND_Ret	**NAND_BlockErase** (NMX_uint32 udAddress);	udAddress	Address of the block to be erased.	Performs a BLOCK ERASE operation.
NAND_Ret	**NAND_CacheProgram** (udword udPageAddresses[ ], NMX_uint8 piecesNumber, dataWidth *Buffer, NMX_uint32 udLength[ ], ubyte*errorPage);[1]	udPageAddresses	Addresses of the pages to be programmed.	Performs a CACHE PROGRAM operation.
		PiecesNumber	Number of data pieces to write.	
		Buffers	Buffer of data to write.	
		udLength	Size of each piece of data to write.	
		errorPage	Number of the page where (if applicable) the cache program failed.	

Table 5:   Command Functions (Continued)

Return Value	Function Name	Parameter	Parameter Description	Function Description
NAND_Ret	**NAND_CopyBack** (NMX_uint32 udSourceAddr, NMX_uint32 udDestinationAddr, NMX_uint16 *offsetInPage, NMX_uint16 *chunkSizes, NMX_uint16 numOfChunks, dataWidth *Buffer);(1)	udSourceAddr	Address of the source page to copy.	Performs a COPY BACK operation.
		udDestinationAddr	Address of the destination page.	
		OffsetInPage	Starting offsets of the pieces of data to rewrite.	
		ChunkSizes	Size of each piece of data to rewrite.	
		numOfChunks	Number of data pieces to rewrite.	
		Buffers	Buffers of data pieces to write.	
NAND_Ret	**NAND_PageRead** (NMX_uint32 *udAddresses, dataWidth *Buffer, NMX_uint16 numOfChunks, NMX_uint32 *udLength);(1)	udAddresses	Addresses to read from in the page.	Performs a PAGE READ operation.
		Buffer	Destination buffer to store the read data.	
		numOfChunks	Number of addresses contained in udAddresses.	
		udLength	Lengths of the data pieces to be read.	
NAND_Ret	**NAND_PageProgram** (NMX_uint32 *udAddresses, dataWidth *Buffer, NMX_uint8 numOfChunks, NMX_uint32 *udLength);(1)	udAddresses	Addresses to be programmed in the page.	Performs a PAGE PROGRAM operation.
		Buffer	Source buffers containing the data to be programmed.	
		numOfChunks	Number of addresses contained in udAddresses.	
		udLength	Length of the data pieces to be programmed.	
void	**NAND_ReadElectronicSignature** (dataWidth * Buffer);(1)	Buffer	Buffer to store the read data.	Reads the electronic signature.
void	NAND_Reset (void);	–	–	Resets the NAND device.
NAND_Ret	NAND_CacheRead (NMX_uint32 udAddress, dataWidth *Buffer, NMX_uint32 length);(1)	udAddress	Addresses of the page where the cache read starts.	Performs a CACHE READ operation.
		Buffer	Destination buffer to store the data read in the first page.	
		length	Length of the data to be read.	
void	NAND_Terminate_CacheRead void)	–	–	Exits from cache read mode.

Table 5:   Command Functions (Continued)

Return Value	Function Name	Parameter	Parameter Description	Function Description
NAND_Ret	NAND_CacheReadDataOutput (dataWidth *Buffer, NMX_uint32 length)[1]	Buffer	Destination buffer to store the data.	Reads the following pages from buffer.
		length	Length of data to be read.	
ubyte	NAND_ReadStatusRegister (void);	–	–	Read the status register.
NAND_Ret	**NAND_MultiplanePageProgram** (NMX_uint32 FPudAddresses[ ], dataWidth *FPBuffer, NMX_uint8 FPnumOfChunks, NMX_uint16 *FPudlength, NMX_uint32 *SPudAddresses, dataWidth *SPBuffer, NMX_uint8 SPnumOfChunks, NMX_uint16 *SPudlength);	FPudAddresses	Addresses of the first plane to be programmed.	Performs multi-plane page programming with random data input.
		FPBuffer	Source buffer with the first plane data to be programmed.	
		FPnumOfChunks	Numbers of chunks to program in the first plane.	
		FPudlength	Length of data to be programmed to the first plane.	
		SPudAddresses	Addresses of the second plane to be programmed.	
		SPBuffer	Source buffer with the second plane data to be programmed.	
		SPnumOfChunks	Numbers of chunks to program in the second plane.	
		SPudlength	Length of data to be programmed to the first plane.	

Table 5: Command Functions (Continued)

Return Value	Function Name	Parameter	Parameter Description	Function Description
NAND_Ret	**NAND_CopyBack** (NMX_uint32 FPudSourceAddr, NMX_uint32 FPudDestinationAddr, NMX_uint16 *FPoffsetInPage, NMX_uint16 *FPchunkSizes, NMX_uint16 FPnumOfChunks, dataWidth *FPBuffer, NMX_uint32 SPudSourceAddr, NMX_uint32 SPudDestinationAddr, NMX_uint16 *SPoffsetInPage, NMX_uint16 *SPchunkSizes, NMX_uint16 SPnumOfChunks, dataWidth *SPBuffer)	FPudSourceAddr	Address of the source plane to be programmed.	Performs the MULTI-PLANE COPY BACK function.
		FPudDestinationAddr	Address of the source page to be programmed.	
		FPoffsetInPage	The starting offset of the first page.	
		FPchunkSizes	Length of source data to be programmed.	
		FPnumOfChunks	Numbers of chunks of the source plane to be programmed.	
		FPBuffers	Source buffer with the second plane data to be programmed.	
		SPudSourceAddr	Address of the source plane to be programmed.	
		SPudDestinationAddr	Address of the destination page to be programmed.	
		SPoffsetInPage	The starting offset of the second plane.	
		SPchunkSizes	Length of source data of second plane to be programmed.	
		SPnumOfChunks	Numbers of chunks of the source plane of second plane to be programmed.	
		SPBuffers	Source buffer with the second plane data to be programmed.	
NAND_Ret	**NAND_MultiplaneBlockErase** NMX_uint32 FPudAddress, NMX_uint32 SPudAddress);	FPudAddresses	Addresses of the first plane to be erased.	Performs a MULTI-PLANE ERASE.
		SPudAddress	Addresses of the second plane to be erased.	

Notes: 1. The dataWidth is either NMX_uint8 for x8 bus widths or NMX_uint16 for x16 bus widths.

Using the Driver

Choosing the Device

Before using the software on the target device, set the device specific #define section located in the file.

- The software drivers support all 4224 byte page NAND Flash memory devices (see "References" on page 18). All very large page NAND Flash memory devices have the same number of pages per block. It is specified by:

    ```
    #define NUM_PAGE_BLOCK
    ```

- The NAND Flash memory device is defined using the appropriate #define statement (see Table 6 for a definition of the constants):

    ```
    #define NAND083F32A
    ```

The target device is automatically defined by a set of constants. Device constants are listed in Table 6 and the values of the constants according to the selected device are listed in Table 7.

Table 6: Device Constants Definitions

Constant Name	Description
FLASH_WIDTH	FLASH_WIDTH
FLASH_SIZE	FLASH_SIZE
PAGE_SIZE	PAGE_SIZE
PAGE_DATA_SIZE	PAGE_DATA_SIZE
PAGE_SPARE_SIZE	PAGE_SPARE_SIZE
NUM_BLOCKS	NUM_BLOCKS
SHIFT_A8	SHIFT_A8
MULTIPLANE	MULTIPLANE

Table 7: Device Constants Values

Device	FLASH_ WIDTH	FLASH_ SIZE	PAGE_ SIZE	PAGE_DATA_ SIZE	PAGE_SPARE_ SIZE	NUM_ BLOCKS
8Gb	8 bits	8Gb	4224 bytes	4096 bytes	224	4095

Enabling DMA

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

DMA moves a block of adjacent data from one memory to the other.

The following registers must be set before using DMA:
- **SRC DMA**: This register contains the start address of the source block.
- **DST DMA**: This register contains the start address of the destination block.
- **CNT DMA**: This register contains the number of memory units of a block.
- **CON DMA**: This is the configuration register.

The DMA engine is enabled by setting the following #define statement:

#define DMA_ENABLE

According to the target hardware application, the code to manage the DMA engine must
be inserted at every occurrence of the #define. The following example is for a platform
based on an ARM7TDMI core:

```
#ifdef DMA_ENABLE

do

i=*(volatile udword*) (GDMACON0);

while ( (i&0x2) != 0);

*((volatile unsigned int*) (GDMASRC0)) = Base_Address;

*((volatile unsigned int*) (GDMADST0)) = (udword)Buffer;

*((volatile unsigned int*) (GDMACNT0)) = udLength[0];

*(volatile unsigned int*) (GDMACON0) = 0x0081;

udIndex+=udLength[0];

#endif

/*read data*/

#ifndef DMA_ENABLE

for(i=0;i<udLength[0];i++)

{

Buffer[udIndex++] = NAND_DataOutput();

}

#endif
```

Getting Started

To test the source code in a target system, start by reading from the NAND08GW3F2A device. If the device is erased, only 0xFFh data should be read. Then, read the manufacturer and device codes by issuing a NAND_ReadElectronicSignature 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. All very large page Flash memory functions can be called and executed within the main() function. The following example checks the device identifiers (device code, manufacturer code) and performs a simple BLOCK ERASE command:

```
void main(void) {
    dataWidth buffer[4];
    udword address;
    udword n_block, n_page = 0x0, column_addr = 0x0;

    memset(buffer,0xFF,4);
    printf("Read Electronic Signature \n");
    NAND_ReadElectronicSignature(buffer);
    printf("Manufacturer Code: %x\n",buffer[0]);
    printf("Device Code: %x\n",buffer[1]);
    printf("Reserved Byte 3: %x\n",buffer[2]);
    printf("Byte 4: %x\n",buffer[3]);

    n_block = 10;    // block 10 will be erased
    address = ((n_block<<19)|(n_page<<13)|column_addr);
    NAND_BlockErase(address);
} /* EndFunction Main */
```

Porting the Software Driver

Porting the software driver on a particular platform requires only that the hardware dependent layer functions be rewritten to correctly manage the signals of the NAND Flash memory device connected to the platform.

Connecting with GPIO

A NAND Flash memory device can be connected to a ROM/SRAM/Flash bank and at least two GPIO ports and the following signals:

- **Not output enable (nOE)**: If a memory access occurs, the nOE output controls the output enable port of the specific memory.
- **Not write byte enable (nWBE)**: If a memory access occurs, the nWBE outputs control the write enable port of the specific memory.
- **Not ROM/SRAM/Flash chip select (nRCS)**: KS32C50100 can access up to six external Flash banks. By controlling the nRCS signals, the CPU addresses can be mapped into the physical memory banks.

Any bus operation that accesses the NAND device consists of two steps:

1. Set the GPIO pins that drive the NAND control signals (CL, AL).
2. Perform a WRITE or READ operation to the bank where the NAND Flash memory device is mapped.

The control nOE, nWBE, and nRCS signals of the microcontroller are directly connected to the NAND memory device. This hardware solution is possible only for NAND Flash memory devices with the Chip Enable Don't Care option, which allows the NAND device to share the ROM/SRAM/Flash controller.

The NAND device's write protect (WP) signal can be managed via hardware using a jumper or via software with a GPI/O. It is possible to monitor the ready/busy state of the NAND device by reading the status register or by connecting a GPI/O to the ready/busy (RB) pin.

Hardware Interface

The Evaluator7T board used for this example is built around a KS32C50100 microcontroller and based on a 16/32-bit ARM7TDMI RISC processor (refer to the KS32C50100 data sheet).

The KS32C50100 can operate at a bus frequency of 50MHz. It has a system memory controller (SMC) with configurable banks for ROM/SRAM/Flash and DRAM and for external devices.

Configuring the GPIO – Example

The following example shows how to configure the GPIO connection by using the hardware-dependent NAND_CommandInput function. Three steps are required:

1. **Set GPIO signals:** Set GPI/O 0 LOW and GPI/O 1 HIGH to drive the NAND control signals latch enable (AL) LOW and command latch enable (CL) HIGH.
2. **Issue a command to the NAND Flash memory:** Perform a WRITE or READ operation to the bank where the memory is mapped to set the chip enable ($\overline{E}$) and write enable ($\overline{W}$) signals.
3. **UnSet signals:** Reset GPI/O 0 and GPI/O 1 LOW to drive the NAND control signals latch enable (AL) LOW and command latch enable (CL) LOW.

```
void NAND_CommandInput(ubyte ubCommand) {

    ubyte * udAddress =  (ubyte*) Base_Address;

    /* Set Op mode Command (AL=0,CL=1) *

     *(GPIODATA)=BASE_IO_DATA|ASSERT_CL;

    /*transfer to NAND ubCommand*/

     *(udAddress) = ubCommand;

    /* UnSet Op mode (AL=0,CL=0) */

     *(GPIODATA)=BASE_IO_DATA;

}
```

Connecting With a Field Programmable Gate Array (FPGA)

The driver was tested on a Mainstone II board equipped with a Rodan card. The Rodan card consists of all components required to support a configurable Flash interface based on FPG.s interposer/socket design. With the Flash interposer removed, an FPGA will isolate the Flash interposer data and address the bus from the Intel Bulverde processor and SDRAM system bus.

With the Rodan card, all board transceivers are bypassed and the signals are passed through the field programmable gate array (FPGA) to interface with the Flash side of the bus. The interposer data and addresses bus comes from the FPGA and directly connects to the high-density BGA for the Flash interposer site. The control signals are mostly dedicated single point nets between the FPGA and respective interposer site.

The memory management register bank (MMRB) provides a memory mapped register set to control the base address, size, and type of each chip select driven out of the FPGA.

A function can be created that automatically writes to the MMRB configuration register at boot up. The function is put into the universal bootloader for ease of configuration.

Since NAND devices are not able to be directly memory mapped, a memory mapped I/O method must be used to communicate with the device. For the Rodan card, it has been set so that the NAND memory will be on chip select 5 of the XScale processor.

Table 8:   NAND Controller Register Addresses

Transfer Type	Address	Read/Write
Chip Base (CB)	0x14000000	–
Command	CB Address + 0x20000	Write
Address	CB Address + 0x10000	Write
Data Write	CB Address + 0x0	Write
Data Read	CB Address + 0x0	Read

To configure the Mainstone II registers:

```
/*MAINSTONEII BOARD*/
#define BASE_ADDR 0x94000000


NMX_uint16*     p_command;  /*CLE address*/
NMX_uint16*     p_address_common_area; /*ALE address*/
NMX_uint16*     p_read;      /*Common memeory read address*/
NMX_uint16*     p_write;     /*Common memeory write address*/


p_command = (NMX_uint16*)(BASE_ADDR + 0x20000);
p_address_common_area = (NMX_uint16*)(BASE_ADDR+ 0x10000);
p_read  = (NMX_uint16*)BASE_ADDR;
p_write = (NMX_uint16*)BASE_ADDR;


void NAND_CommandInput(NMX_uint8 ubCommand) {
```

```
*p_command = ubCommand;
}
```

References

- NAND08GW3F2A 8Gb, 2442 byte page, 1.8 or 3V, SLC Flash memory data sheet