NAND

Driver Design Document

Document #	Author(s)	Approval(s)

Template Version 0.1

Information in this document is subject to change without notice. Texas Instruments may have pending patent applications, trademarks, copyrights, or other intellectual property rights covering matter in this document. The furnishing of this documents is given for usage with Texas Instruments products only and does not give you any license to the intellectual property that might be contained within this document. Texas Instruments makes no implied or expressed warranties in this document and is not responsible for the products based from this document.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

This document discusses the TI device driver design and architecture for NAND Device Driver for DSP/BIOS platforms. The target audience includes device driver developers from TI as well as consumers of the driver.

The NAND storage devices are used for applications such as mass storage applications. This design is based on the PSP driver framework architecture.

1.1 Terms and Abbreviations

Term	Description
API	Application Programmer's Interface
CSL	TI Chip Support Library – primitive h/w abstraction
IP	Intellectual Property
Bad Block	A unusable physical block in the NAND device
OS	Operating System
NAND	NAND storage devices
OneNAND	OneNAND storage devices
FTL	Flash Translation Layer
WLA	Wear Leveling Algorithm
DDC	Device Driver Core
DDA	Device Driver Adaptation
LLC	Low Level Controller

1.2 Related Documents

1.	spru403o.pdf	DSP/BIOS Driver Developer's Guide
2.	xxxx_BIOSPSP_Userguide.doc	NAND user guide
3.	SPRUFL6_EMIFA.pdf	EMIFA H/W Controller

2 System Purpose

EMIFA memory controller is complaint with the JESD21-C SDR SDRAM memories utilizing 16-bit data bus of EMIFA memory controller. The purpose of this EMIFA is to provide a means for the CPU to connect to a variety of external devices including:

- Single data rate (SDR) SDRAM
- Asynchronous devices including NOR Flash, NAND Flash, and SRAM

Nand driver uses EMIFA to interface with both a NAND device.

NAND driver lies below the Block Media module. The Block Media Driver transfers calls from application/file system to the NAND driver which is registered to block media. The driver would use the DSP BIOS™ APIs for OS services. The Driver is supposed to be synchronous driver. The NAND driver actually read/write the data to the nand device.

2.1 Context

The block diagram below shows the overall system architecture:

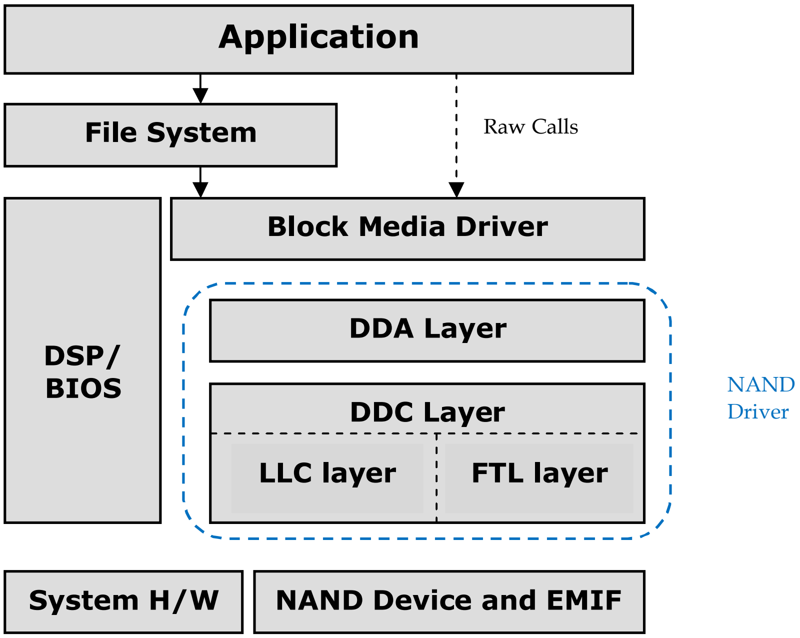

Figure 1: System Architecture

The central portion shown constitutes the mainline NAND driver component; the surrounding modules constitute the supporting system components. The supporting system modules, do not specifically deal with NAND, but assist the driver by providing utility services.

2.2 System Interface

The driver supports following key features:

- Supports 512-byte page and 2048-byte page NAND devices
- Supports 8-bit and 16-bit NAND devices
- Error correction using 4-bit ECC mechanism
- Supports wear-leveling and bad-block management functionalities
- Supports protecting a portion of the NAND flash from application access

2.3 Non-functional Requirements

The NAND driver does not know anything about the file system being used and in no way dependent on it for its functioning. So formatting, recognizing the partition and understanding the FAT does not come under the purview of the driver.

3 Structure

This section deals with the overall architecture of NAND device driver, including the device driver partitioning as well as deployment considerations. We'll first examine the system decomposition into functional units and the interfaces presented by these units. Following this, we'll discuss the deployed driver or the dynamic view of the driver where the driver operational scenarios are presented.

3.1 Overview

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities. The NAND device driver functionality can be enacted by three key roles defined here under:

- Device Driver Adaptation (DDA), takes care of interfacing with the generic block media layer
- Device Driver Core (DDC) along with FTL implements the protocol part of the driver
- Low level Controller (LLC), provides services to perform primitive access necessary to control/configure/examine status, of the underlying h/w device.

3.2 Components

The components of the NAND device driver and interdependent modules are briefly described below:

3.2.1 Device Driver Core (DDC)

The DDC layer implements the NAND device driver functionality. The DDC layer handles functionalities such as initialization, open, close, page write, page read, block erase and device configuration. The functionalities of the DDC layer can be invoked by the device driver abstraction layer. The DDC layer is composed of two sub-modules.

1. Low Level Controller (LLC): The LLC module provides the NAND module hardware abstraction to the DDC and FTL modules. The LLC implements the functionalities to configure the NAND hardware module for various to interface with various NAND device types and perform basic NAND device operations such as physical page write, physical page read block erase and initialization.
2. Flash Translation Layer (FTL): The FTL module provides a linear logical sector abstraction of the NAND device. It abstracts the device characteristics such as bad block management from the applications. This allows the applications to view the NAND device media as consecutively arranged sectors.

3.2.2 Device Driver Abstraction (DDA)

The DDA layer implements abstractions to interface the DDC layer of the NAND driver with a block media driver. The DDA module implements functionalities to hookup to a block media driver and receive read and write requests from the block media driver. The functionalities of the DDC layer can be invoked applications layer (file system or RAW mode).

3.2.3 Block Media Driver

Block media driver provides abstraction to register the media driver with the file system. The NAND driver registers itself with block media driver, which in turn registers the driver with file system. All file operations from file system would be redirected to NAND driver.

3.2.4 Operating System and Applications

The applications layer invokes the various functionalities provided by the NAND device driver. The DSP/BIOS layer provides operating system related functionality such as locking and critical region protection for the NAND DDC driver layer.

3.3 Interfaces

In the following subsections, the interfaces implemented by each of the sub-component are specified. Please refer to BIOSPSP_nand.chm for complete details on APIs

3.3.1 LLC Module

The NAND LLC module abstraction defines the interfaces that should be supported by all implementations of the LLC modules in the NAND driver.

3.3.1.1 Data types

This section describes the data types that are defined by the LLC module abstraction. The implementations of the LLC module are create instances of the data types defined by the LLC module abstraction.

1. LLC_nandInitConfig - The LLC_nandInitConfig data structure defines the initialization parameters that are accepted by the implementations of the LLC module. Table 1 lists the elements contained in the LLC_nandInitConfig data structure.

#	Element Type	Element Name	Description
1	Uint32	instanceId	Instance number of the chip select number to be used
2	Uint32	clkFreq	Input clock frequency for the EMIF module
3	PSP_nandDeviceInfo*	deviceInfo	Pointer to data structure of type PSP_nandDeviceInfo
4	PSP_nandDeviceTiming *	deviceTiming	Pointer to data structure of type PSP_nandDeviceTiming
5	Ptr	hEdma	EDMA driver handle
6	Uint32	edmaEvtQ	EDMA event queue to be used for transfer requests
7	Uint32	edmaChanne l	EDMA channel number

Table 1: Elements in the LLC_nandInitConfig data structure

2. LLC_nandSpareArea - The LLC_nandSpareArea data structure defines the representation of the information in the spare area in the page of a NAND device. Table 1lists the elements contained in the LLC_nandSpareArea data structure.

#	Element Type	Element Name	Description
1	Uint32	logicalBlock	Logical block number to which the physical page belongs
2	Bool	badBlockMark	Indicates whether a block is good or bad
3	Uint8	logicalPage	Logical page number to which the physical page is mapped
4	Uint8*	eccData	ECC data, if any, applicable to the page of the NAND page

Table 2: Elements in the LLC_nandSpareArea data structure

3.3.1.2 Interfaces

This NAND LLC module abstraction layer specifies a list of functions that should be implemented by the NAND LLC modules. These functions are abstract, that is, the LLC abstraction layer defines the type of the functions that LLC modules should implement. This section describes the abstract interfaces that LLC abstraction layer specifies and general guidelines about the functionality to be implemented in these functions by the LLC module implementations.

Table 3 lists the abstract functions that nand drivers should implement.

Abstract Function Name	Description
LLC_nandInitFxn	The implementation of this function should initialize the hardware controller, detect presence of NAND device and report the organization of the NAND device.
LLC_nandWritePageFxn	The implementation of this function should write data to a physical page specified.
LLC_nandReadPageFxn	The implementation of this function should read data from a physical page specified.
LLC_nandEraseBlockFxn	The implementation of this function should erase a physical block specified.
LLC_nandMarkBlockAsBadFxn	The implementation of this function should delete an instance of the audio codec driver.
LLC_nandReadSpareAreaFxn	The implementation of this function should read the spare area of the page.
LLC_nandIoctlFxn	The implementation of this function should execute IOCTL commands specified by the LLC abstraction.
LLC_nandDeinitFxn	The implementation of the function should de-initialization of the LLC module.
LLC_nandIsWriteProtectFxn	The implementation of the function should determine write protection status
LLC_nandIsBlockBadFxn	The implementation of the function should determine if a given block is BAD

Table 3: Abstract interfaces defined by the nand abstraction

1. LLC_nandInitFxn - The implementation of the LLC_nandInitFxn interface should initialize the LLC module and prepare the device for the data transfer. The declaration of this interface is listed below.

```
typedef Ptr (*LLC_nandInitFxn)(LLC_nandInitConfig const *initCfg,
                               PSP_nandDeviceInfo      **devInfo);
```

Table 4 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
LLC_nandInitConfig	*initCfg	This parameter specifies the initialization parameters for the LLC module.
PSP_nandDeviceInfo	**devInfo	This parameter provides the address of a PSP_nandDeviceInfo structure.

Table 4: Parameters for LLC_nandInitFxn interface

The return value of LLC_nandInitFxn interface is of type Ptr. Table 5 lists the value of the return type and description of the return value.

Return value	Description
Non-Null Ptr	The LLC module has been initialized successfully. The handle should be used for subsequent invocation of the LLC module interfaces.
Null Ptr	The LLC module has not been initialized. The following could be the reasons for the failure in opening the channel A. Invalid parameter has been specified. B. There was no NAND device detected or an unrecognized NAND device was detected. C. The system resources such as EDMA channels and semaphores could not be acquired. D. NAND device dose not respond to commands.

Table 5: Return values of LLC_nandInitFxn Interface.

The implementation of the LLC_nandInitFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return a null handle.
B. Initialize the EMIF for NAND flash interface.
C. Reset NAND device
D. Read the device ID of the NAND device and configure the timing parameters in the EMIF.
E. Acquire resources such as EDMA channel and semaphores as required.

2. LLC_nandWritePageFxn - The implementation of the LLC_nandWritePageFxn interface should write data into a page of the NAND device. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandWritePageFxn)(Ptr  const  handle,
                                      Uint32             phyBlock,
                                      Uint8              phyPage,
                                      Uint8              *data,
                                      LLC_nandSpareArea  *spareArea,
                                      Bool               computeEcc);
```

Table 4 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	phyBlock	This parameter specifies the physical block in which the page has to be written.
Uint8	phyPage	This parameter specifies the physical page in the physical block number that should be written.
Uint8	*data	This parameter points to the buffer of data that should be written into the specified page.
LLC_nandSpareArea	*spareArea	This parameter points to structure that holds logical block number and logical page number. Pointer eccData of this structure will not be NULL and will hold ECC values that have to be written, when computeEcc function parameter is FALSE.
Bool	computeEcc	Flag to indicate if ECC has to be computed for the given data. If this flag is set to FALSE, member eccData of LLC_nandSpareArea will not be NULL and will hold the ECC for data to be written. If flag is set to TRUE, this function will have to compute and store ECC for the given data

Table 6: Parameters for LLC_nandWritePageFxn interface

The return value of LLC_nandWritePageFxn interface is of type Int32. **Error! Reference source not found.** lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	The page write has been completed successfully.
PSP_NAND_E_WRITE_FAIL	The page write failed.
PSP_NAND_E_WRITE_PROTECTED	The physical block is write-protected and page cannot be written.
PSP_E_TIMEOUT	The page write operation timed-out since the device did not respond to the write command.

Table 7: Return values of LLC_nandWritePageFxn Interface.

The implementation of the LLC_nandWritePageFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.

B. Perform the device specific page write operation. Ensure that the logical block number and logical page number are copied into the spare area of the page.

C. Determine the status of the page write operation and report any error in the page write operation. Note: If there is any error in writing the page, this function should not mark the block as bad. The marking of bad block would be initiated by the consumer of the LLC module.

3. LLC_nandReadPageFxn - The implementation of the LLC_nandReadPageFxn interface should read data from a page of the NAND device. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandReadPageFxn)(Ptr const   handle,
                                     Uint32             phyBlock,
                                     Uint8              phyPage,
                                     Uint8              *data,
                                     LLC_nandSpareArea  *spareData,
                                     Bool               useEcc);
```

Table 4 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	phyBlock	This parameter specifies the physical block from which the page has to be read.
Uint8	phyPage	This parameter specifies the physical page in the physical block number that should be read.
Uint8	*data	This parameter points to the buffer of data into which the data read from the page will be copied.
LLC_nandSpareAres	*spareData	Used to return the logical block number, logical page number, bad block indicator and ecc (optionally)
Bool	useEcc	Flag to indicate if ECC module should be used or not.

Table 8: Parameters for LLC_nandReadPageFxn interface

The return value of LLC_nandReadPageFxn interface is of type Int32. **Error! Reference source not found.** lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	The page read has been completed successfully.

PSP_NAND_E_ECC_FAIL	Uncorrectable ECC error has been detected in the data read from the specified page.
PSP_E_TIMEOUT	The page read operation timed-out since the device did not respond to the read command.

Table 9: Return values of LLC_nandReadPageFxn Interface.

The implementation of the LLC_nandReadPageFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Perform the device specific page read operation.
C. Perform any error correction operations as applicable to the device.
D. Check the status of the page read operation and report any error in the page read operation.

4. LLC_nandEraseBlockFxn - The implementation of the LLC_nandEraseBlockFxn interface should erase a physical block in the NAND device. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandEraseBlockFxn)(Ptr const handle,
                                       Uint32           blockNum);
```

Table 4 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	blockNum	This parameter specifies the physical block that should be erased.

Table 10: Parameters for LLC_nandEraseBlockFxn interface

The return value of LLC_nandEraseBlockFxn interface is of type Int32. Table 11 lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	The erase has been completed successfully.
PSP_NAND_E_BLOCK_BAD	The block is a bad block and cannot be erased.
PSP_NAND_E_WRITE_PROTECTED	The block is write-protected and cannot be erased.
PSP_NAND_E_ERASE_FAIL	The block erase operation failed and the block should be treated as a bad block

PSP_E_TIMEOUT	The block erase operation timed-out since the device did not respond to the read command.

Table 11: Return values of LLC_nandEraseBlockFxn Interface.

The implementation of the LLC_nandEraseBlockFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Perform the device specific block erase operation.
C. Determine the status of the block erase operation and report any error in the block erase operation. Note: If there is any error in erasing the block, this function should not mark the block as bad. The marking of bad block would be initiated by the consumer of the LLC module.

5. LLC_nandMarkBlockAsBadFxn - The implementation of the LLC_nandMarkBlockAsBadFxn interface should mark a physical block in the NAND device as bad block. The convention used to mark the block as bad block is specific to the LLC module implementation and not defined by the LLC abstraction. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandMarkBlockAsBadFxn)(Ptr const handle,
                                           Uint32           blockNum);
```

Table 12 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	blockNum	This parameter specifies the physical block that should be marked as bad block.

Table 12: Parameters for LLC_nandMarkBlockAsBadFxn interface

The return value of LLC_nandMarkBlockAsBadFxn interface is of type Int32. Table 13 lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	The block has been marked as bad successfully.
PSP_E_TIMEOUT	The mark bad block operation timed-out.

Table 13: Return values of LLC_nandMarkBlockAsBadFxn Interface.

The implementation of the LLC_nandMarkBlockAsBadFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Perform the device specific bad block mark operation.
C. Determine the status of the bad block mark operation and report any error in the marking the block as bad block.

6. LLC_nandReadSpareAreaFxn - The implementation of the LLC_nandReadSpareAreaFxn interface should read the spare area of a page. The spare area information read from the page should be returned back in the LLC_nandSpareArea data structure. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandReadSpareAreaFxn)(Ptr const   handle,
                                          Uint32             phyBlock,
                                          Uint8              phyPage,
                                          LLC_nandSpareArea  *spareData);
```

Table 14 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	phyBlock	This parameter specifies the block number of the physical page from which the spare area information have to be read.
Uint8	phyPage	This parameter specifies the physical page number in the block from which the spare area information has to be read.
LLC_nandSpareArea	*spareData	The parameter specifies the reference to a LLC_nandSpareArea data object into which the spare area information read from the page will be stored.

Table 14: Parameters for LLC_nandReadSpareAreaFxn interface

The return value of LLC_nandReadSpareAreaFxn interface is of type Int32. Table 15 lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	The spare area has been successfully read.

PSP_E_TIMEOUT	The spare area read operation timed-out.

Table 15: Return values of LLC_nandReadSpareAreaFxn Interface.

The implementation of the LLC_nandReadSpareAreaFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Perform the device specific spare area read operation.
C. Determine the status of the read spare area operation and report any error in the reading the spare area information.

7. LLC_nandIoctlFxn - The implementation of the LLC_nandIoctlFxn interface should execute the IOCTL commands specified by the LLC module abstraction definition. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandIoctlFxn)(Ptr const  handle,
                                  LLC_nandIoctlCmd  ioctlCmd,
                                  Ptr               *param);
```

Table 16 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
LLC_nandIoctlCmd	ioctlCmd	This parameter specifies the NAND IOCTL command specified that should be executed by the LLC module.
Ptr	*param	This is an optional parameter that could be used to supply arguments for executing the IOCTL command.

Table 16: Parameters for LLC_nandIoctlFxn interface

The return value of LLC_nandIoctlFxn interface is of type Int32. Table 17 lists the value of the return type and description of the return value. Note: All the error values that could be returned by this interface is not listed since the error values are implementation specific.

Return value	Description
IOM_COMPLETED	The IOCTL command has been successfully executed
PSP_NAND_E_ERROR	Invalid parameters have been specified for the IOCTL command.

Table 17: Return values of LLC_nandIoctlFxn Interface.

The implementation of the LLC_nandIoctlFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Execute the specified IOCTL command.
C. Determine the status of executing the IOCTL command and return appropriate status information.

8. LLC_nandDeinitFxn - The implementation of the LLC_nandDeinitFxn interface should de-initialize the LLC module. The declaration of this interface is listed below.

```
typedef Int32 (*LLC_nandDeinitFxn)(Ptr const handle);
```

Table 18 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.

Table 18: Parameters for LLC_nandDeinitFxn interface

The return value of LLC_nandDeinitFxn interface is of type Int32. Table 19 lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	The LLC module has been successfully de-initialized
PSP_NAND_E_ERROR	Invalid parameters have been specified for the de-initialize operation.

Table 19: Return values of LLC_nandDeinitFxn Interface.

The implementation of the LLC_nandDeinitFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Reset the device as required and reset the EMIF register to reset values.
C. Release any system resources such as EDMA channels and semaphores that were acquired during initialization.

9. LLC_nandIsWriteProtectFxn - The implementation of the LLC_nandIsWriteProtectFxn interface should determine if the media is write protected. The granularity of this operation shall be limited to whole media (i.e. no block level protection shall be checked).

```
typedef Int32 (*LLC_nandIsWriteProtectFxn)(Ptr const handle,
                                            Uint32           phyBlkNum);
```

Table 18 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	phyBlkNum	Specifies the physical block number, this parameter is reserved for future use and should have value of FTL_NAND_INVAL_BLOCK

Table 20: Parameters for LLC_nandIsWriteProtectFxn interface

The return value of LLC_nandIsWriteProtectFxn interface is of type Int32. Table 19 lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	If the media is NOT write protected
PSP_NAND_E_WRITE_PROTECTED	If the media is WRITE protected
PSP_NAND_E_ERROR	If the physical block number is in-valid or handle is NULL

Table 21: Return values of LLC_nandIsWriteProtectFxn Interface.

The implementation of the LLC_nandIsWriteProtectFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the PSP_NAND_E_ERROR error.
B. Read the status of the media.
C. Check if media is write protected and return appropriate value.

10. LLC_nandIsBlockBadFxn - The implementation of the LLC_nandIsBlockBadFxn interface should determine if the given physical block is BAD or not. Its expected that function would look for BAD BLOCK marker in first, second and last page of the device.

```
typedef Int32 (*LLC_nandIsBlockBadFxn)(Ptr const  hNandObj,
                                        Uint32           blockNum);
```

Table 18 lists the parameters expected by this interface.

Parameter Type	Parameter Name	Description
Ptr	handle	This parameter specifies the handle of the LLC module that was returned by the LLC_nandInitFxn function.
Uint32	blockNum	Specifies the physical block number.

Table 22: Parameters for LLC_nandIsBlockBadFxn interface

The return value of LLC_nandIsBlockBadFxn interface is of type Int32. Table 19 lists the value of the return type and description of the return value.

Return value	Description
IOM_COMPLETED	If the block is not marked as BAD
LLC_NAND_E_BLOCK_BAD	If the block is marked as BAD
IOM_EBADARGS	If the physical block number is in-valid or handle is NULL
PSP_NAND_E_READ_FAIL	If function is unable to read the spare area.

Table 23: Return values of LLC_nandIsBlockBadFxn Interface.

The implementation of the LLC_nandIsBlockBadFxn could include the following functionalities.

A. Validate the input parameters. If the parameters are invalid, return the IOM_EBADARGS error.
B. Read first, second and last page of the device.
C. Determine if any of the pages is marked as BAD and return appropriate value.

3.3.2 FTL Interfaces

The NAND FTL provides interfaces for logical page read and write operations. This section describes the design of the FTL interfaces.

3.3.2.1 *FTL Initialization*

The NAND FTL initialization interface initializes the NAND FTL module. The NAND FTL module is composed of sub-modules such as the WLA/BBM module, cache module and block mapping/translation module. All the sub-modules are initialized during the NAND FTL initialization procedure. The following is the procedure of initializing the NAND FTL module:

1. Initialize the logical to physical block mapping table, WLA/BBM module and the cache module.
2. Initialize all the sub-modules of the FTL based in the eraseAtInit option provide with the FTL initialization. If the eraseAtInit option is true, then erase

all the physical blocks and update the sub modules of the FTL, care should be taken to NOT erase blocks marked as BAD. If the eraseAtInit option is false, then reads the spare area of the first/second page of all the physical blocks in the device and update the sub-modules of the NAND FTL.

3. Determine the usable number of logical sectors in the NAND device and return this information to the caller.

3.3.2.2 FTL Logical Page Write Interface

The NAND FTL logical page write interface implement the functionality required to write the contents of the logical page to a physical page and maintain the mapping between the logical and the physical page. Figure 2 illustrates the NAND FTL page write procedure.

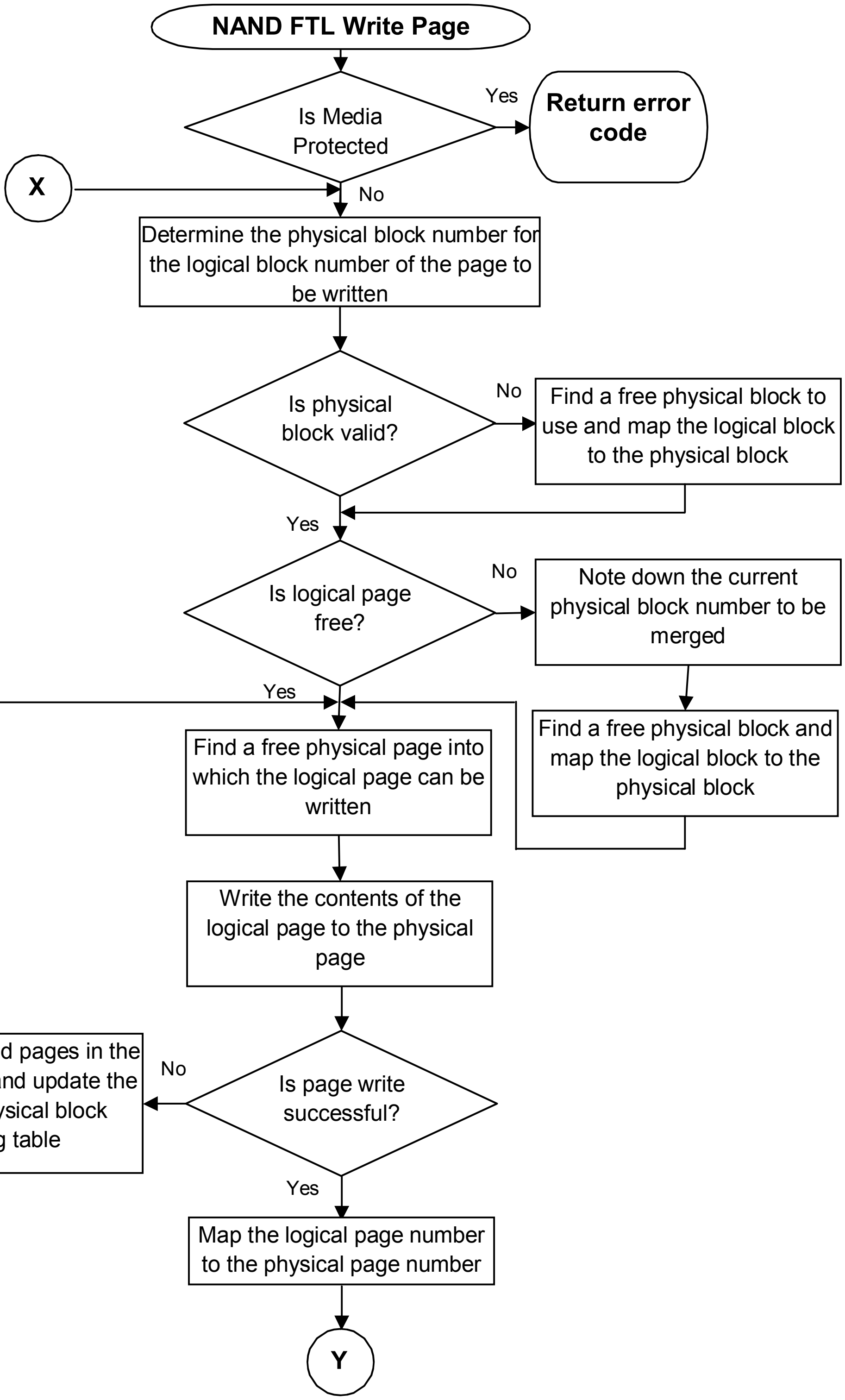
NAND FTL Write Page
Is Media Protected
Yes
Return error code
X
No
Determine the physical block number for the logical block number of the page to be written
Is physical block valid?
No
Find a free physical block to use and map the logical block to the physical block
Yes
Is logical page free?
No
Note down the current physical block number to be merged
Yes
Find a free physical block and map the logical block to the physical block
Find a free physical page into which the logical page can be written
Write the contents of the logical page to the physical page
Move all the valid pages in the physical block and update the logical to physical block mapping table
No
Is page write successful?
Yes
Map the logical page number to the physical page number
Y

Figure 2: NAND FTL Write Page procedure

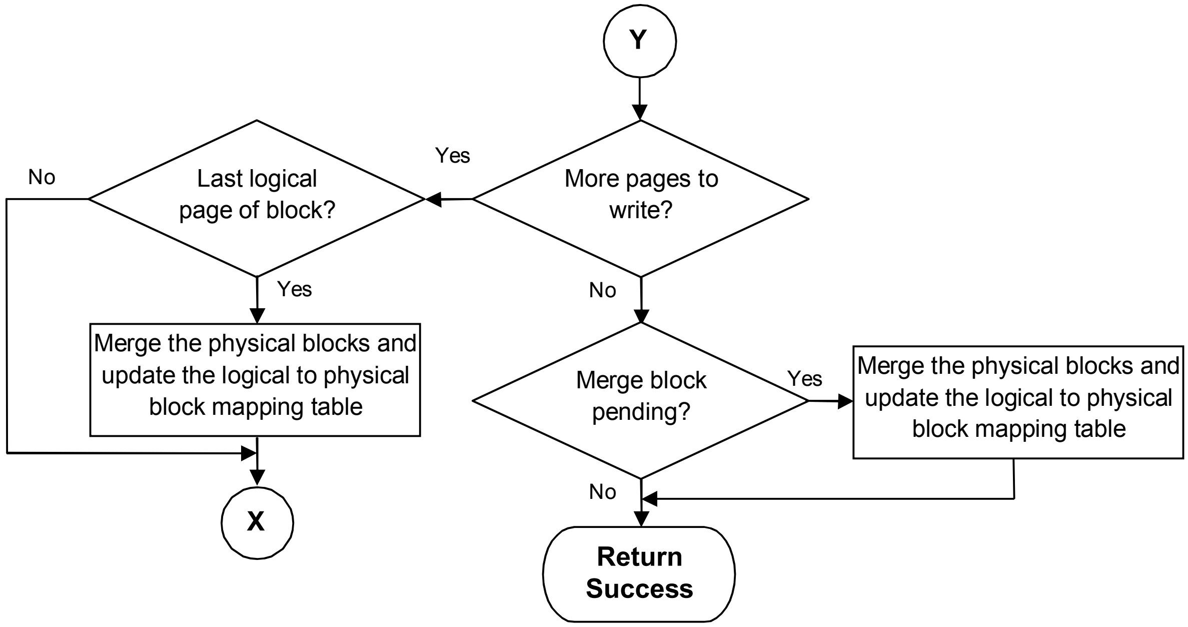
Y
More pages to write?
Yes
Last logical page of block?
No
Yes
Merge the physical blocks and update the logical to physical block mapping table
X
No
Merge block pending?
Yes
Merge the physical blocks and update the logical to physical block mapping table
No
Return Success

Figure 3: NAND FTL Write Page procedure (continued)

The NAND FTL logical page write procedure is described below:

1. Determine if the media is write protected if media is write protected return appropriate error code.
2. Determine the logical block number ‘LB’ and the logical page number ‘LP’ of the logical page to be written to the NAND device.
3. Determine the physical block number ‘PB1’of logical block number ‘LB’. If the logical block number ‘LB’ is not mapped to any physical block number, then allocate a new physical block ‘PB1’ for the logical block number and update the logical to physical mapping table.
4. Determine if the logical page ‘LP’ is already written in the physical block ‘PB1’. If the logical page is already written in the NAND device, allocate a new physical block ‘PB2’ for the logical block ‘LB’ and update the logical to physical block mapping table.
5. Allocate a physical page ‘PP’ for the logical page ‘LP’ in the physical block mapped to the logical block ‘LB’.
6. Write the data from the buffer to the physical page number ‘PP’. If the page write is complete, then proceed to step 7.
7. If the page write results in a page write error, then move the contents of the physical block to a new physical block and assign the new physical block number to ‘PB’. If the block move operation failed, then terminate the page write procedure and return the error code to the caller. Repeat again from step 4.
8. Map the logical page number ‘LP’ to the physical page number ‘PP’ in the cache.
9. Determine if there are additional pages to write to NAND flash. If there are no more pages to write to the NAND device, then proceed to step 12.

10. Determine if the current logical page 'LP' is the last logical page of the logical block 'LB'. If it is not the last logical page of the logical block, then proceed to step 2.
11. The logical page written was the last logical page of the logical block written. Merge the physical blocks if the block merge operation is pending.
12. Repeat from step 2.
13. All the request number of pages has been written. Merge the physical blocks if the block merge operation is pending and update the logical to physical block mapping table.

3.3.2.3 *FTL Logical Page Read Interface*

The NAND FTL logical page read interface implement the functionality required to read the contents of the logical page that is mapped to a physical page of the NAND device. **Error! Reference source not found.** illustrates the NAND FTL logical page read procedure.

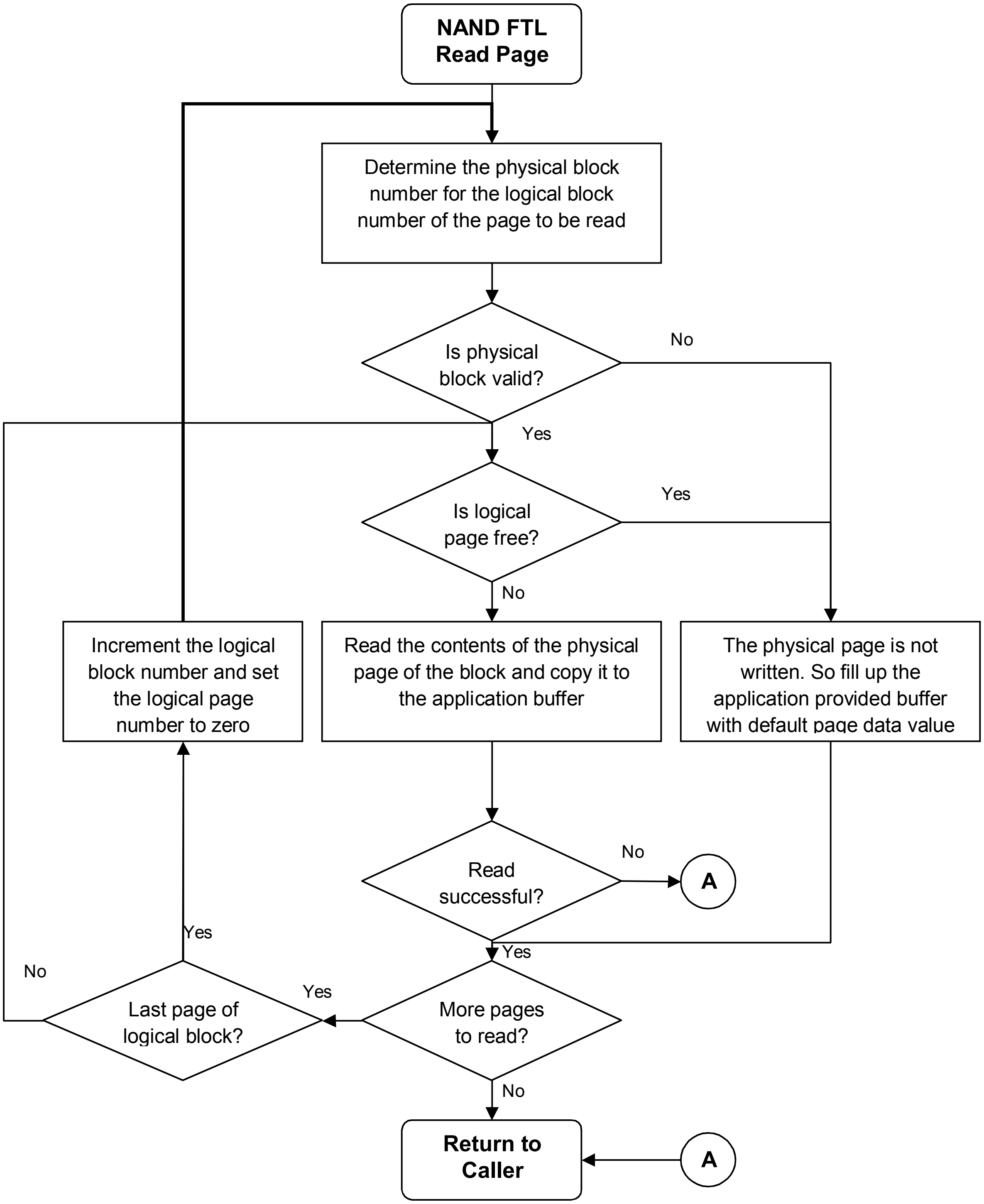
NAND FTL Read Page
Determine the physical block number for the logical block number of the page to be read
Is physical block valid?
No
Yes
Is logical page free?
Yes
No
Increment the logical block number and set the logical page number to zero
Read the contents of the physical page of the block and copy it to the application buffer
The physical page is not written. So fill up the application provided buffer with default page data value
Read successful?
No
A
Yes
Yes
No
Last page of logical block?
Yes
More pages to read?
No
Return to Caller
A

Figure 4: NAND FTL Read Page procedure

The NAND FTL logical page read procedure is described below:

1. Determine the logical block number 'LB' and the logical page number 'LP' of the logical page to be read from the NAND device.

2. Determine the physical block number 'PB1'of logical block number 'LB'. If the logical block number 'LB' is not mapped to any physical block number, then proceed to step 5.
3. Determine the physical page number 'PP' of the logical page number 'LP'. If the logical page number is not mapped to any physical page number, then proceed to step 5.
4. Read the data from the physical page 'PP' and physical block 'PB' and copy the data to the application buffer. Proceed to step 6. If the read operation fails, report appropriate error code.
5. The logical page it not mapped to any physical page in the NAND device. So fill up the application buffer with default NAND page data "0xFF".
6. Determine if there are additional pages to read. If there are no additional pages to read, then return to the caller with status as success.
7. Determine if the last page read was the last page the logical block. If it was the last page of the logical block, then increment the logical block number, set the logical page number to zero and repeat from step 1.

3.3.3 FTL Utility Interfaces

The NAND FTL module uses two utility functions for performing the block merge and block move operations. This section describes these utility functions in detail.

3.3.3.1 FTL Merge Block Interface

During the page write operations, if the page write has to be performed on a physical page in block 'X' that is already written, then the page to be written is written to a new physical block 'Y'. All the subsequent writes to the physical block 'X' are written to the newly allocated physical block. Since the pages of a logical block are split into two physical blocks, the merging of the two physical blocks have to be performed.
The merging operation is performed under the following two scenarios.

A. The current multiple page write request have been completed.
B. The last logical page of the logical block write is complete.

Figure 5 illustrates the procedure of merging two blocks.

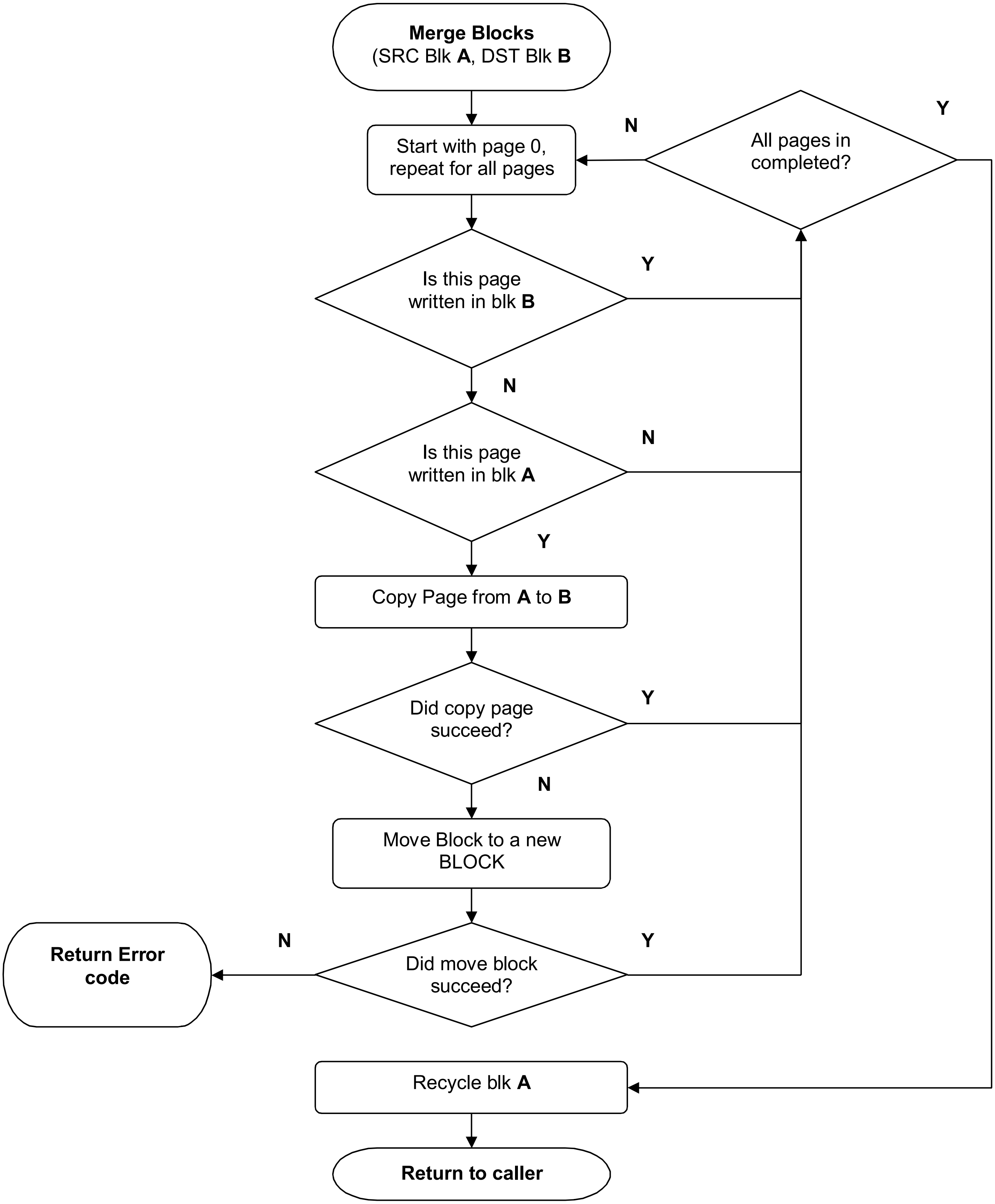

Figure 5: NAND FTL merge block procedure

The following is the procedure for merging two physical page in two physical blocks ‘B1’ and ‘B2’.

1. Let the logical page number ‘LP’ be equal to zero.
2. Determine if the logical page number ‘LP’ is already written in the physical block **B**. If the logical page ‘LP’ is already written in block **B**, then proceed to step 8, as the latest data is available in **B**.
3. Determine if the logical page number ‘LP’ is written in physical block **A**. If the logical page ‘LP’ is not written in physical block **A**, then proceed to step 8.
4. The logical page ‘LP’ has to be moved from block **A** to block **B**. Allocate a new physical free page ‘P2’ from the physical block **A**.
5. Copy the physical page ‘P1’ to physical page ‘P2’.
6. Determine if the page write failed during the page copy of ‘P1’ to ‘P2’. If the page write failed, then move all the available pages of block **B** to a new block **B’**. Set the block number **B** = **B’**’ and repeat the page copy from step 4. If no free blocks are available, then return to caller with the out of free blocks error.
7. Map the logical page number ‘LP’ to physical page number ‘P2’ for block **B** in the FTL cache module.
8. Determine if all the logical pages of a block have been merged. If there are more logical blocks to be merged, then repeat from step 2.
9. All the logical pages in block **A** and block **B** have been merged. Recycle the physical block **A** and return to caller.

3.3.3.2 Moving the contents of valid pages from one block to another

During a page write operation, if the write to page fails, the physical block to which the page belongs is considered to have become a bad block. In such a case, the contents of the physical block have to be moved to a new physical block and the write should be performed on the new physical block.

Figure 6 illustrates the procedure of moving a physical block ‘B1’ to a new physical block ‘B2’.

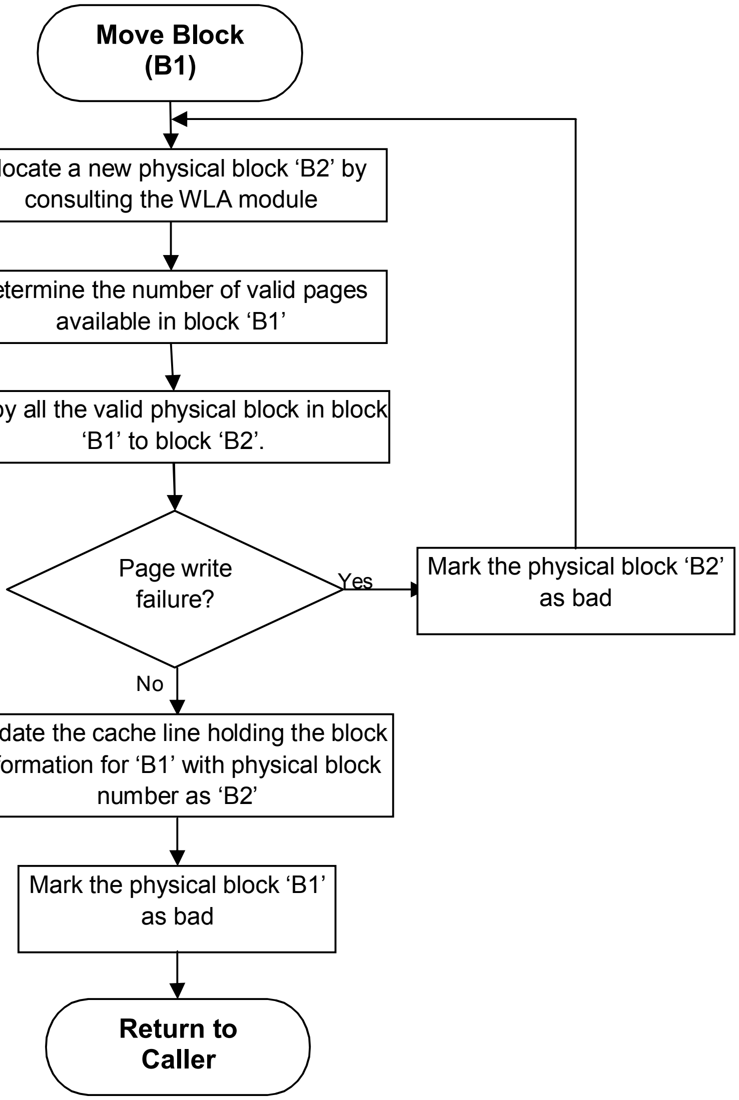
Move Block (B1)
Allocate a new physical block ‘B2’ by consulting the WLA module
Determine the number of valid pages available in block ‘B1’
Copy all the valid physical block in block ‘B1’ to block ‘B2’.
Page write failure?
Yes
Mark the physical block ‘B2’ as bad
No
Update the cache line holding the block information for ‘B1’ with physical block number as ‘B2’
Mark the physical block ‘B1’ as bad
Return to Caller

Figure 6: NAND FTL move block procedure

The following is the procedure for moving the contents of a physical block ‘B1’ to a new block:

1. Allocate a new physical block ‘B2’ by consulting the WLA table. If there are no free blocks available then return to the caller with out of free blocks error code.
2. Determine the number of valid pages available in the physical block ‘B1’. All the valid pages in block ‘B1’ have to be moved to physical block ‘B2’.
3. Copy all the valid physical pages from the physical block ‘B1’ to physical block ‘B2’.
4. Determine if there was page write failure when moving the block from physical block ‘B1’ to physical block ‘B2’. If a page write failure occurred, then mark the physical block ‘B2’ as bad and repeat from step 1.
5. Update the physical block number in the cache line holding the page mapping information of block ‘B1’.
6. Mark the physical block ‘B1’ as bad block and return the caller.

4 Dynamic Behavior

The nand device driver implement synchronous interface to the user. All synchronous IO occur in the application thread of control, the calling thread may suspend for the requested transaction to complete. This 'wait on completion' occurs in the DDA layer.

4.1 Driver Creation/Initialization

The sequence below depicts the creation/initialization phase of the NAND driver. Once this phase is complete, the basic driver data structures and setups are complete and ready for formally opening device to perform IO.

User is expected to invoke PSP_nandDrvInit(), way up in the application startup phase, which in turn invokes PSP_nandInit(). The PSP_nandInit() performs book-keep functions on the driver and allocates memory for instance data structures. It attaches the DDC create functions for use later during actual initialization of each device instance. This function is not reentrant function. Once the init is done it calls PSP_nandOpen() to open the nand driver. It then registers itself with the block media driver and tells its availability.

4.2 Driver Open

When the application calls the PSP_nandOpen() driver entry point, the DDC open function is invoked enabling the hardware instance of NAND. This function is not reentrant function.

4.3 IO Control

The NAND Driver provides an ioctl interface to set/get common configuration parameters on the driver at run time. This function is reentrant.

It should be observed that the user's IOCTL request completes in the context of calling thread i.e., application thread of control.

4.4 IO Access

The application invokes the PSP_nandRead() and PSP_nandWrite() IO interfaces for data transfer using the NAND. Both of these functions are reentrant.

4.5 Driver Close

The application can invoke the PSP_nandDrvDeInit() to de-init nand driver. This function calls the PSP_nandClose() function to close the instance of the NAND device. Any DMA channels opened are also released via appropriate calls. This function is not reentrant.

5 Design of NAND DDC module

The DDC module of the NAND driver provides an abstracted interface to the LLC and FTL modules of the driver. The DDC module is an OS independent module. The following section describes the interfaces provided by the NAND DDC module

5.1 Initializing the NAND DDC module

The initialization interface of the NAND DDC module performs the following operations:

1. Validates the state of the DDC module. The DDC module should be in the deleted state for the initialization to proceed. If the state of the NAND module is incorrect, the DDC NAND module initialization is terminated.
2. Validate the input parameters supplied for the initialization. If any of the parameters are invalid, the DDC NAND module initialization is terminated.
3. Initializes the NAND LLC module (either NAND or OneNAND LLC module as selected by the initialization parameter). If the LLC module initialization fails, the DDC NAND module initialization is terminated.
4. Initializes the NAND FTL module. If the FTL module initialization fails, the DDC NAND module initialization is terminated.
5. If all the initialization operations are complete, the state of the NAND driver is set as created and the result of the initialization is returned to the caller.

5.2 Opening the NAND DDC module

The open interface of the NAND DDC module performs the following operations:

1. Validates the state of the DDC module. The DDC module should be in the created state. If the state of the NAND module is incorrect, the DDC NAND module open functionality is terminated.
2. Validate the input parameters supplied for the initialization. If any of the parameters are invalid, the DDC NAND module open is terminated.
3. Sets the state of the NAND DDC module as opened and returns a handle of the NAND DDC module instance.

5.3 Write page interface of the NAND DDC module

The page write interface of the NAND DDC module performs the following operations:

1. Validates the state of the DDC module. The DDC module should be in the opened state. If the state of the NAND module is incorrect, the DDC NAND module page write functionality is terminated.
2. Validates the input parameters supplied for the initialization. If any of the parameters are invalid, the DDC NAND module page write operation is terminated.
3. Performs a range check on the logical sectors requested to be written to the NAND device. If the range check fails, then the page write operation is terminated.
4. Invokes the NAND FTL module page write functionality to perform the writes of the requested logical pages and returns the status of the write operation to the caller.

5.4 Read page interface of the NAND DDC module

The page read interface of the NAND DDC module performs the following operations:

1. Validates the state of the DDC module. The DDC module should be in the opened state. If the state of the NAND module is incorrect, the DDC NAND module page read functionality is terminated.
2. Validates the input parameters supplied for the initialization. If any of the parameters are invalid, the DDC NAND module page read operation is terminated.

3. Performs a range check on the logical sectors requested to be read from the NAND device. If the range check fails, then the page write operation is terminated.
4. Invokes the NAND FTL module page read functionality to perform the reads of the requested logical pages and returns the status of the read operation to the caller.

5.5 IOCTL interface of the NAND DDC module

The IOCTL interface of the NAND DDC module performs the following operations:

1. Validates the state of the DDC module. The DDC module should be in the opened state. If the state of the NAND module is incorrect, the DDC NAND module IOCTL operation is terminated.
2. Validates the input parameters supplied for the initialization. If any of the parameters are invalid, the DDC NAND IOCTL operation is terminated.
3. Executes the requested IOCTL command the returns the status of executing the IOCTL command.

5.6 Close interface of the NAND DDC module

The close interface of the NAND DDC module performs the following operations:

1. Validates the state of the DDC module. The DDC module should be in the opened state. If the state of the NAND module is incorrect, the DDC NAND module close operation is terminated.
2. Validates the input parameters supplied for the initialization. If any of the parameters are invalid, the DDC NAND IOCTL operation is terminated.
3. Executes the requested IOCTL command the returns the status of executing the IOCTL command.

6 Design of NAND FTL module

The NAND media is organized as set of pages and these pages have to written sequentially in order to enhance lifetime of the media. Further it may not be possible to erase single page, a set of pages (called block) could be erased at a time. To re-write a page/block, the block will have to be erased before a re-write operation can be performed.

Typically file system might require re-write operations frequently (such as FAT update), frequent erase and re-write would reduce the lifetime of the media. In order to enhance the lifetime of the media, sectors/blocks specified by file system would be treated as logical sector/block number which could be mapped to different physical sector/block number in the media. The following sections describe these in detail.

6.1 Design of the NAND FTL Logical to Physical Page mapping table

The FTL module utilizes a temporary table of logical to physical page mapping for the pages in a physical block. This module is referred to as the cache. Since the write to the NAND pages in a block are sequential, the mapping between the logical page number and the physical page number are maintained in the cache for faster translation from logical to physical page number. The cache module supports multiple cache lines which in turn enables maintaining the logical to physical page mapping information for multiple blocks.

The cache line consists of meta data and logical to physical page translation table. The meta data in the cache consists of the physical block number being cached by the cache line and the next free physical page available in the block for write operations. Figure 7 illustrates the structure of a cache line. Multiple cache lines can be supported by the cache module.

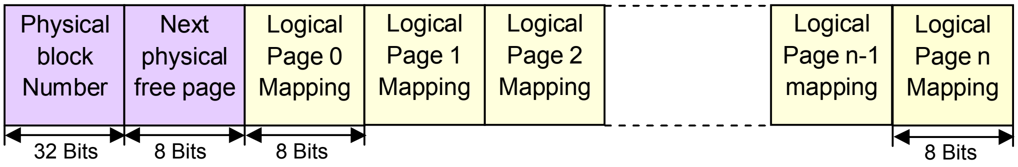

Figure 7: Organization of a cache line in NAND FTL cache module

The cache module supports multiple functions to operate on the data in the cache lines. The following section describes the functions supported by the NAND FTL cache module.

6.1.1 Initializing the cache module

The cache module is initialized before it is used. The initialization is performed during the FTL initialization. During the initialization, all the cache lines in the cache module are reset to default values, that is, the meta data in the cache lines are marked with invalid physical block number.

6.1.2 Determining the cache line index of a cached physical block

The cache module includes a function to determine the cache line index in which the physical page is cached. Given a physical block number, this function searches the meta data in the cache line to determine the cache line index that holds the data.

6.1.3 Invalidate a cache line

The cache module includes a function to invalidate a cache line. Given a physical block number, the cache line invalidate function searches the cache lines to determine the cache line that is caching the logical to physical mapping for a physical block. If a cache line is found, the meta data of that cache line is updated with a invalid physical block number.

6.1.4 Find a free cache line

The cache module includes a function to allocate a free cache line. It searches the meta data of the cache lines to determine if there is any free cache line. If a free cache line is found, that cache line is allocated. If there are no free cache lines, then a round robin algorithm is used to evict one of the cache lines.

6.1.5 Prefetch a cache line

The cache module includes a function to prefetch the logical to physical page mapping of a block into a cache line. Given a physical block, this function allocates a free cache line, reads the logical page number from the spare area of all the valid pages in a block and populates the cache line with that information.

6.1.6 Determine whether a logical page is available for write

The cache module includes a function to determine whether a logical page is available for write in a physical block number. This function searches for the cache line index that is caching the physical block and determines the status of the logical page. If there are no cache lines that currently hold the specified physical block, then the logical to physical mapping information is first prefetched into the cache line.

6.1.7 Map a logical page number to a physical page number

The cache module includes a function to map a logical page number to a physical page number for a specified physical block. This function searches for the cache line index that is caching the physical block. If there are no cache lines that currently hold the specified physical block, then the logical to physical mapping information is first prefetched into the cache line. The physical page number is then mapped to the logical page number in that cache line.

6.1.8 Get a free physical page in a physical block

The cache module includes a function to determine the next usable/writable page in a physical block. This function searches for the cache line index that is caching the physical block. If there are no cache lines that currently hold the specified physical block, then the logical to physical mapping information is first prefetched into the cache line. This function then returns the next usable/writable page number in the physical block. Note: This function does not map logical page number to a physical page number.

6.1.9 Update the physical block number in the meta data of the cache line

The cache module includes a function to update the physical block number of the cache line. The new physical block number specified is updated in the meta data of the cache line.

6.2 Design of NAND FTL Wear Leveling and Bad Block Management

The FTL module of the NAND driver supports wear leveling and bad block management operations. The wear leveling ensures that all the blocks equally undergo erase-write operations. The bad block management ensures that bad blocks are not involved the data storage operations.

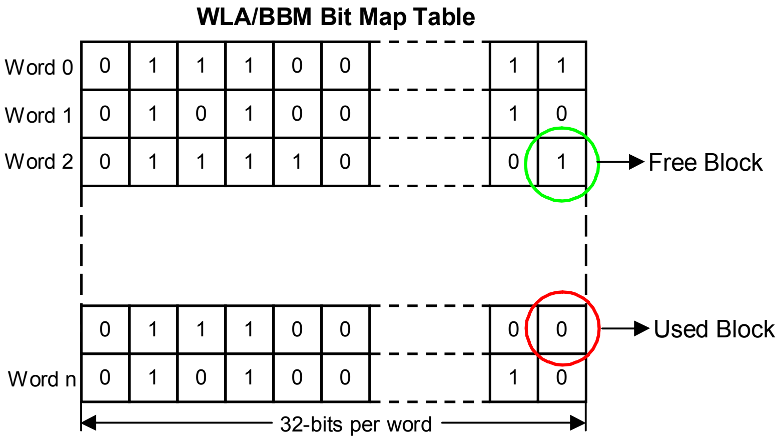

Figure 8: Organization of the WLA/BBM Bit Map table

Figure 8 illustrates the organization of the WLA/BBM bitmap table. The bit map table is organized into multiple words of 32-bits each. Each bit of the bitmap table represents one physical block managed by the WLA/BBM module. The following are the general rules applicable to the information stored in the WLA/BBM table:

1. The first LSB bit represents 0th logical block, second LSB bit represents 1st physical block and so on.
2. A bit with value 1 means that the physical block is available for use and it is in erased state.
3. A bit with value 0 means that the physical block is either in the used state or a bad block.
4. A free block is searched with a round-robin algorithm in the WLA/BBM bitmap table. Starting with the last allocated block.
5. When a physical block is allocated for usage, the bit representing that block is mark as zero.
6. The WLA/BBM bit map table additionally stores meta data to track the last physical block allocated and the number of physical blocks available.

7 Design of NAND LLC module

The NAND driver supports NAND device types of various organizations such as small page, big page, SLC and MLC devices. Interfacing with all the types of NAND devices is performed by the NAND LLC module. The NAND LLC module implements the interfaces defined by the LLC abstraction. This section describes the design of each of the interfaces supported by the NAND LLC module.

7.1 Initialization of the NAND LLC module

Figure 9 illustrates the initialization procedure of the NAND LLC module. The initialization procedure is described below:

1. Initialize the EMIF register for interfacing with NAND devices and set the wait timing to maximum wait states.

2. Determine whether application has provided the NAND device organization information. If application has provided this information, then proceed to step 6.
3. Issue the NAND Read Identification command to determine the NAND device ID. If no NAND devices are found, then the initialization procedure should be terminated with appropriate error code.
4. Determine if the device ID read from the NAND device is a known NAND device. If it is not a known NAND device, then the initialization procedure should terminate with appropriate error code.
5. The details about the NAND device organization found should be copied into a data structure provided by the caller.
6. Determine if the device timing information is provided by the application. If the device timing information is provided by the application, then proceed to step 8. If the application has provided the device organization information but not provided the device timing information, then proceed to step 10.
7. Determine if the device timing information is available in the internal lookup table. If the device timing information is not available in the internal lookup table, then proceed to step 10.
8. Configure the EMIF timing registers based on the input EMIF clock frequency information and timing values of the NAND device.
9. Acquire the required system resources such as EDMA channels and semaphores as required.
10. Return from the LLC NAND module initialization interface with appropriate status.

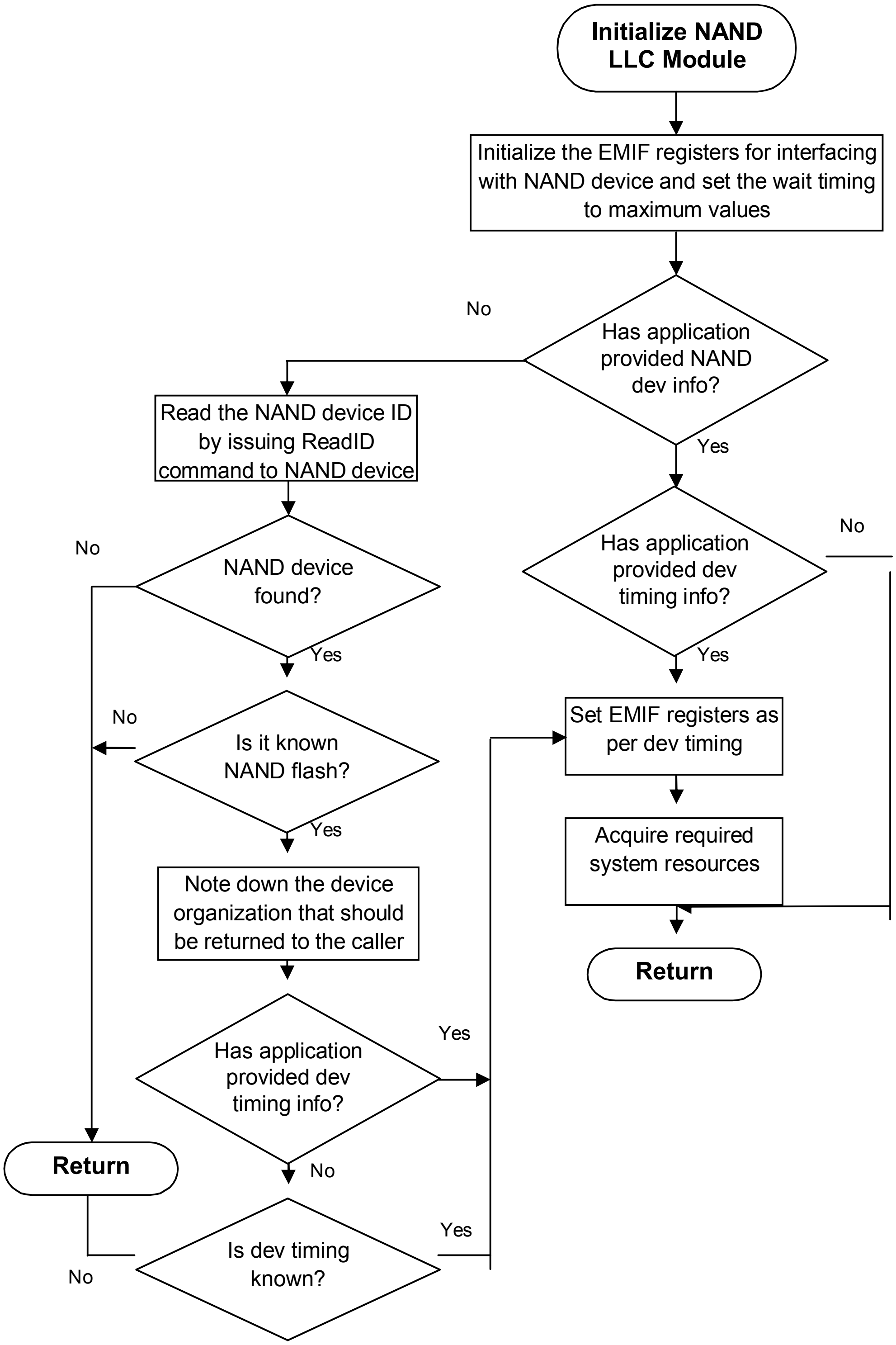
Initialize NAND LLC Module
Initialize the EMIF registers for interfacing with NAND device and set the wait timing to maximum values
Has application provided NAND dev info?
No
Yes
Read the NAND device ID by issuing ReadID command to NAND device
NAND device found?
No
Yes
Is it known NAND flash?
No
Yes
Note down the device organization that should be returned to the caller
Has application provided dev timing info?
Yes
No
Is dev timing known?
Yes
No
Return
Has application provided dev timing info?
No
Yes
Set EMIF registers as per dev timing
Acquire required system resources
Return

Figure 9: NAND LLC module initialization procedure

7.2 Page Write procedure of NAND LLC module

Figure 10 illustrates the page write procedure used by the LLC NAND module.

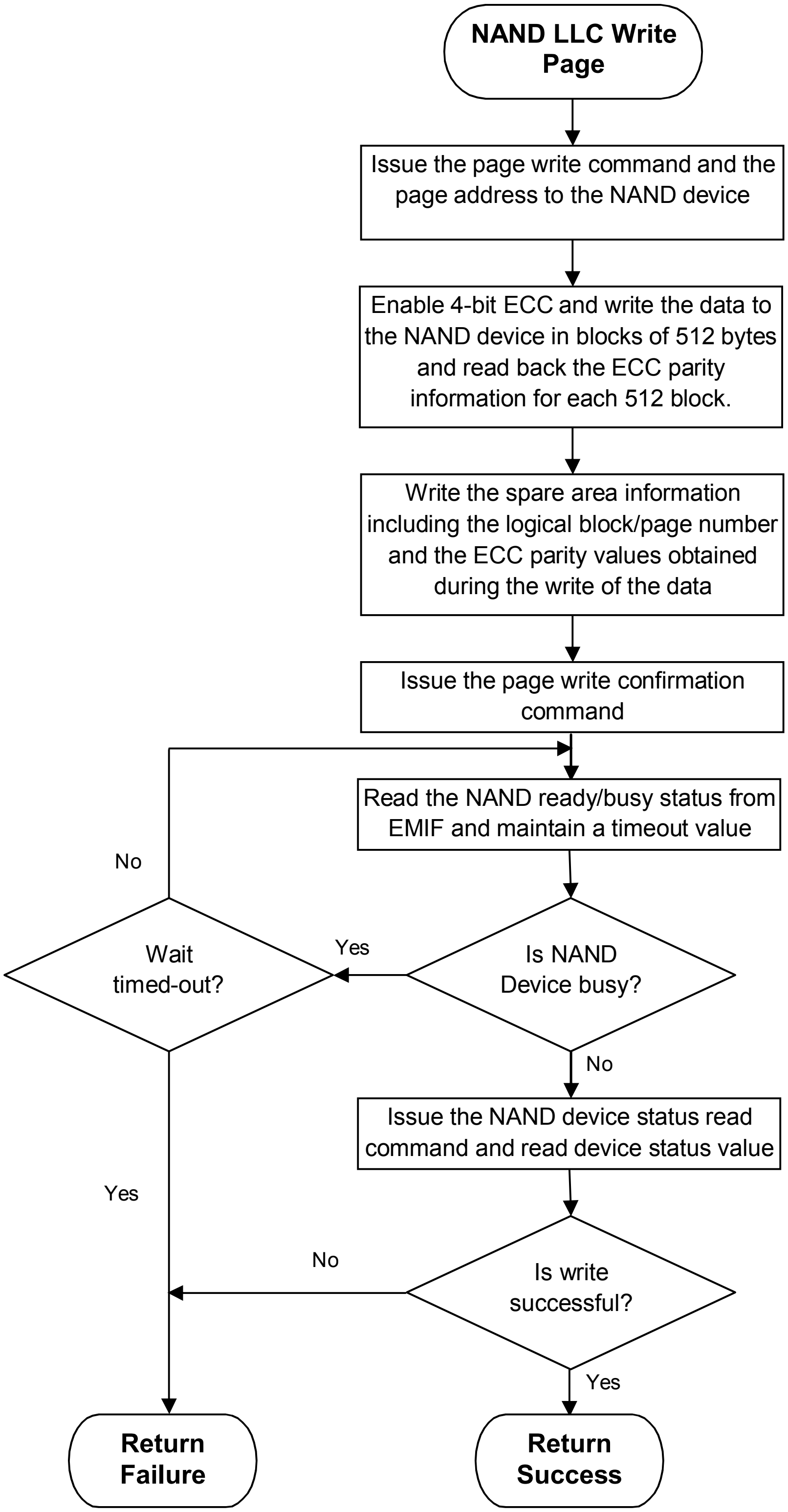
NAND LLC Write Page
Issue the page write command and the page address to the NAND device
Enable 4-bit ECC and write the data to the NAND device in blocks of 512 bytes and read back the ECC parity information for each 512 block.
Write the spare area information including the logical block/page number and the ECC parity values obtained during the write of the data
Issue the page write confirmation command
Read the NAND ready/busy status from EMIF and maintain a timeout value
No
Wait timed-out?
Yes
Is NAND Device busy?
No
Issue the NAND device status read command and read device status value
Yes
No
Is write successful?
Yes
Return Failure
Return Success

Figure 10: NAND LLC module page write procedure

The LLC NAND module page write procedure is described as below:

1. Issue the NAND device page write command followed by the address of the page to be written.
2. Write the contents of the data buffer to the NAND device. The data should be written to the NAND device in blocks of 512 bytes in case the size of the page is bigger that 512 bytes. Before each write of 512 blocks, enable the 4-bit computation by the EMIF module. After every write of 512 bytes, reads the 10-byte ECC parity value generated by the EMIF module and keep a copy of it.
3. Write the contents of the spare area to the NAND device. The spare area information should include the logical block number, logical page number, bad block indicator and ECC parity values that was obtained when writing the data area of the page.
4. Issue the page write confirmation command.
5. From the EMIF registers, determine if the NAND device is in ready or busy state. A timeout count should also be maintained to terminate in case NAND does not change to ready state. If the NAND device is ready, proceed to step 7.
6. If the device is not ready and if the wait has timed-out, then terminate the wait and return from the page write procedure with a error value.
7. After the device turns to ready state, issue the NAND device status read command and determine if the page write succeeded. If the page write did not succeed, then return from the page write procedure with a error value.
8. The page write is complete. Return from the page write procedure with status as success.

7.3 Page Read procedure of NAND LLC module

Figure 11 illustrates the page read procedure used by the LLC NAND module. Note that on there may scenarios where read of data area could fail, in such scenarios the function should attempt retry further this retry should be configurable at compile time. This re-operation is not mentioned in the flow chart.

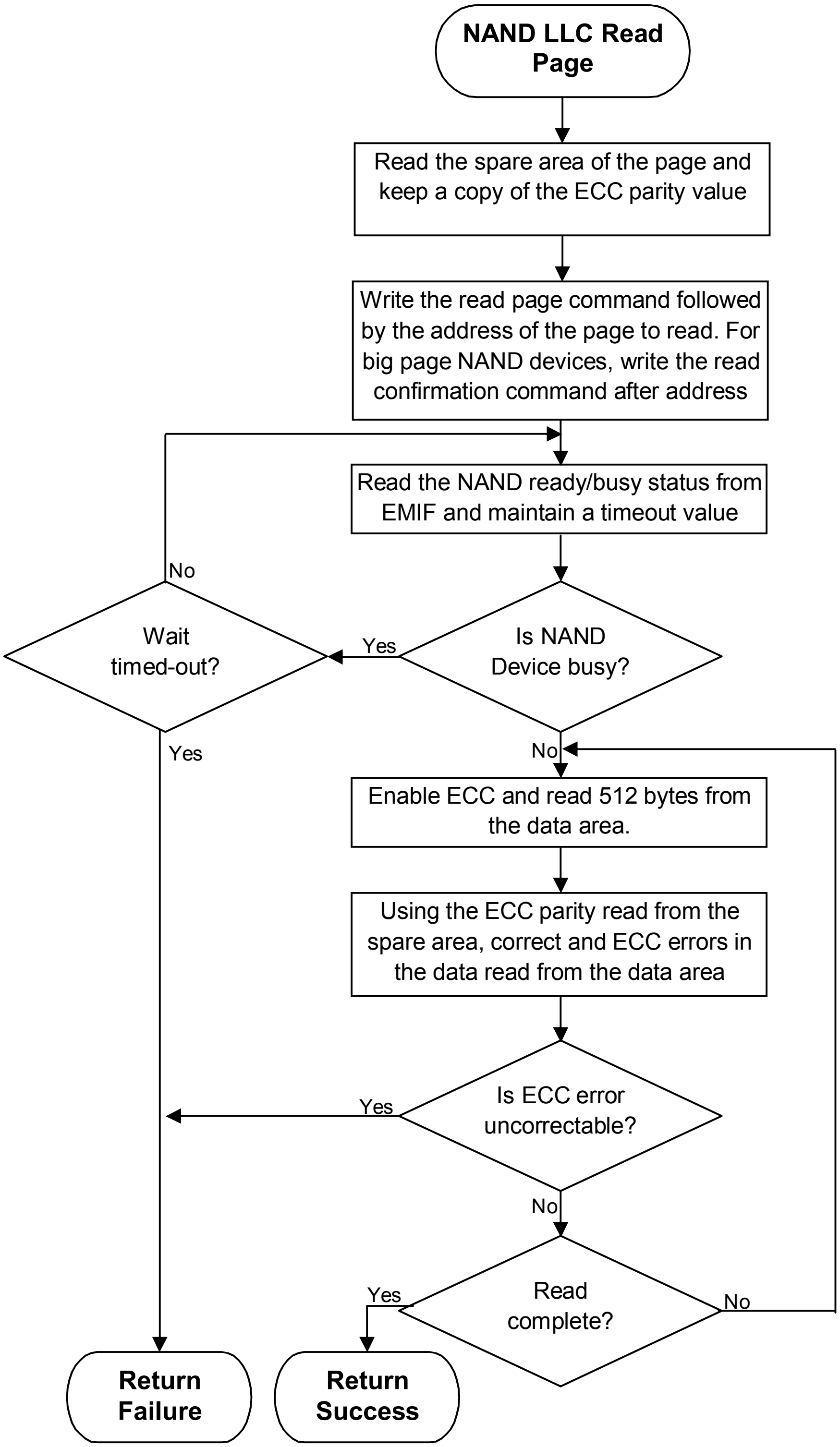
NAND LLC Read Page
Read the spare area of the page and keep a copy of the ECC parity value
Write the read page command followed by the address of the page to read. For big page NAND devices, write the read confirmation command after address
Read the NAND ready/busy status from EMIF and maintain a timeout value
Is NAND Device busy?
Yes
Wait timed-out?
No
Yes
No
Enable ECC and read 512 bytes from the data area.
Using the ECC parity read from the spare area, correct and ECC errors in the data read from the data area
Is ECC error uncorrectable?
Yes
No
Read complete?
Yes
No
Return Failure
Return Success

Figure 11: NAND LLC module page read procedure

The LLC NAND module page read procedure is described as below:

1. Read the 4-bit ECC parity value stored in the spare area of the page and keep a copy of it. The 4-bit parity values will be used to correct any ECC errors in the data from the data area of the page.

2. Issue the NAND device page read command followed by the address of the page to be written. For big page devices, issue the read-confirm command after writing the page address.
3. From the EMIF registers, determine if the NAND device is in ready or busy state. A timeout count should also be maintained to terminate in case NAND does not change to ready state. If the NAND device is ready, proceed to step 7.
4. If the device is not ready and if the wait has timed-out, then terminate the wait and return from the page read procedure with a error value.
5. Enable 4-bit ECC in the EMIF and read 512 bytes of data from the data area of the page. Using the parity information read in step 1, initiate the ECC error correction operation in the EMIF. If there are any correctable ECC errors detected, then correct the ECC errors. If any uncorrectable ECC errors are detected, then terminate the page read procedure and return the failure status.
6. Repeat step 5 until all the complete page area of the page is read from the NAND device.
7. Return from the NAND page read procedure with success as the status.

7.4 Block Erase procedure of NAND LLC module

Figure 12 illustrates the block erase procedure used by the LLC NAND module.

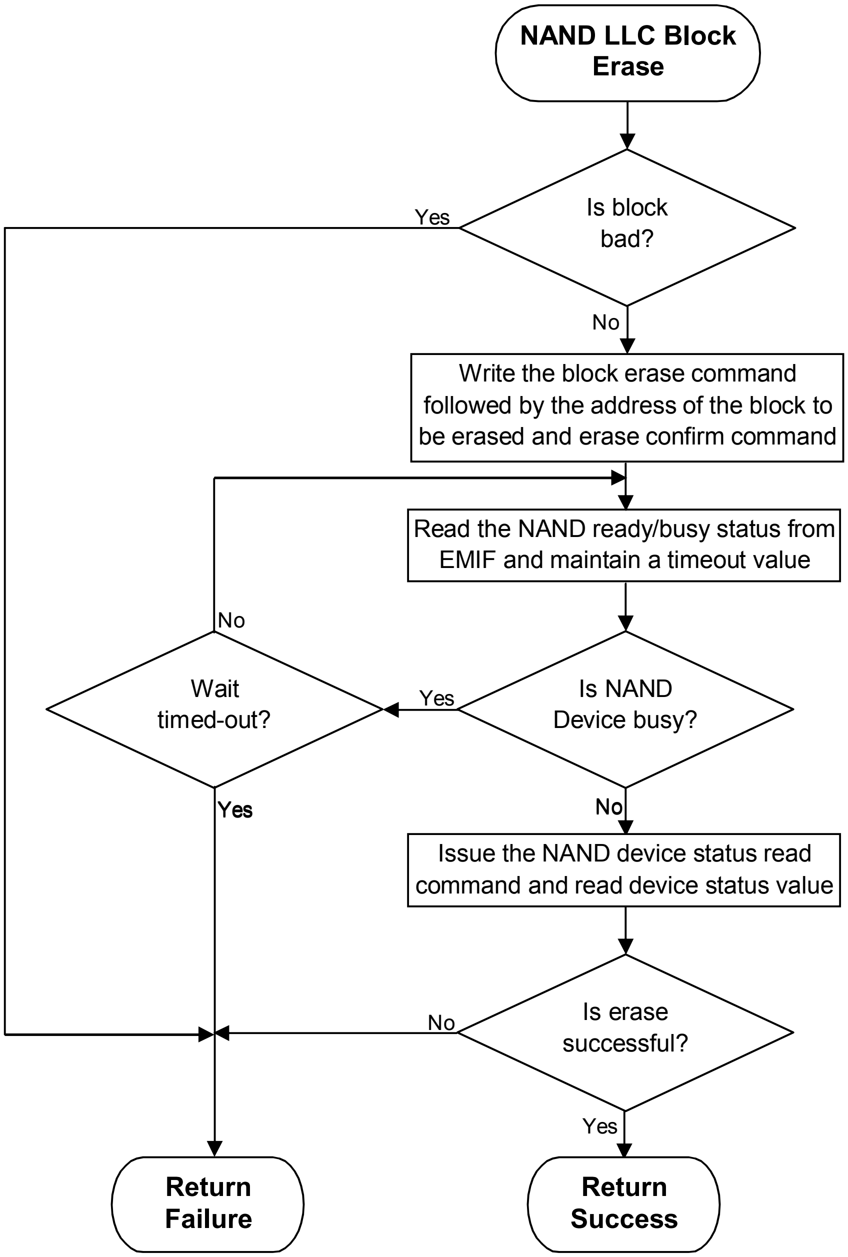
NAND LLC Block Erase
Is block bad?
Yes
No
Write the block erase command followed by the address of the block to be erased and erase confirm command
Read the NAND ready/busy status from EMIF and maintain a timeout value
No
Wait timed-out?
Yes
Is NAND Device busy?
Yes
No
Issue the NAND device status read command and read device status value
No
Is erase successful?
Yes
Return Failure
Return Success

Figure 12: NAND LLC module block erase procedure

The LLC NAND module block erase procedure is described as below:

1. Determine if the block to be erased is a bad block. If the block is a bad block, then return to the caller with failure status.
2. Issue the NAND device block erase command followed by the address of the block to be erased.
3. Issue the block erase confirmation command.
4. From the EMIF registers, determine if the NAND device is in ready or busy state. A timeout count should also be maintained to terminate in case NAND

does not change to ready state. If the NAND device is ready, proceed to step 4.

5. If the device is not ready and if the wait has timed-out, then terminate the wait and return from the block erase procedure with a error value.
6. After the device turns to ready state, issue the NAND device status read command and determine if the block erase succeeded. If the block erase did not succeed, then return from the block erase procedure with an error value.
7. The block erase operation is complete. Return from the block erase procedure with status as success.

7.5 Mark Bad Block procedure of NAND LLC module

Figure 13 illustrates the procedure used by the NAND LLC module to mark a block as bad.

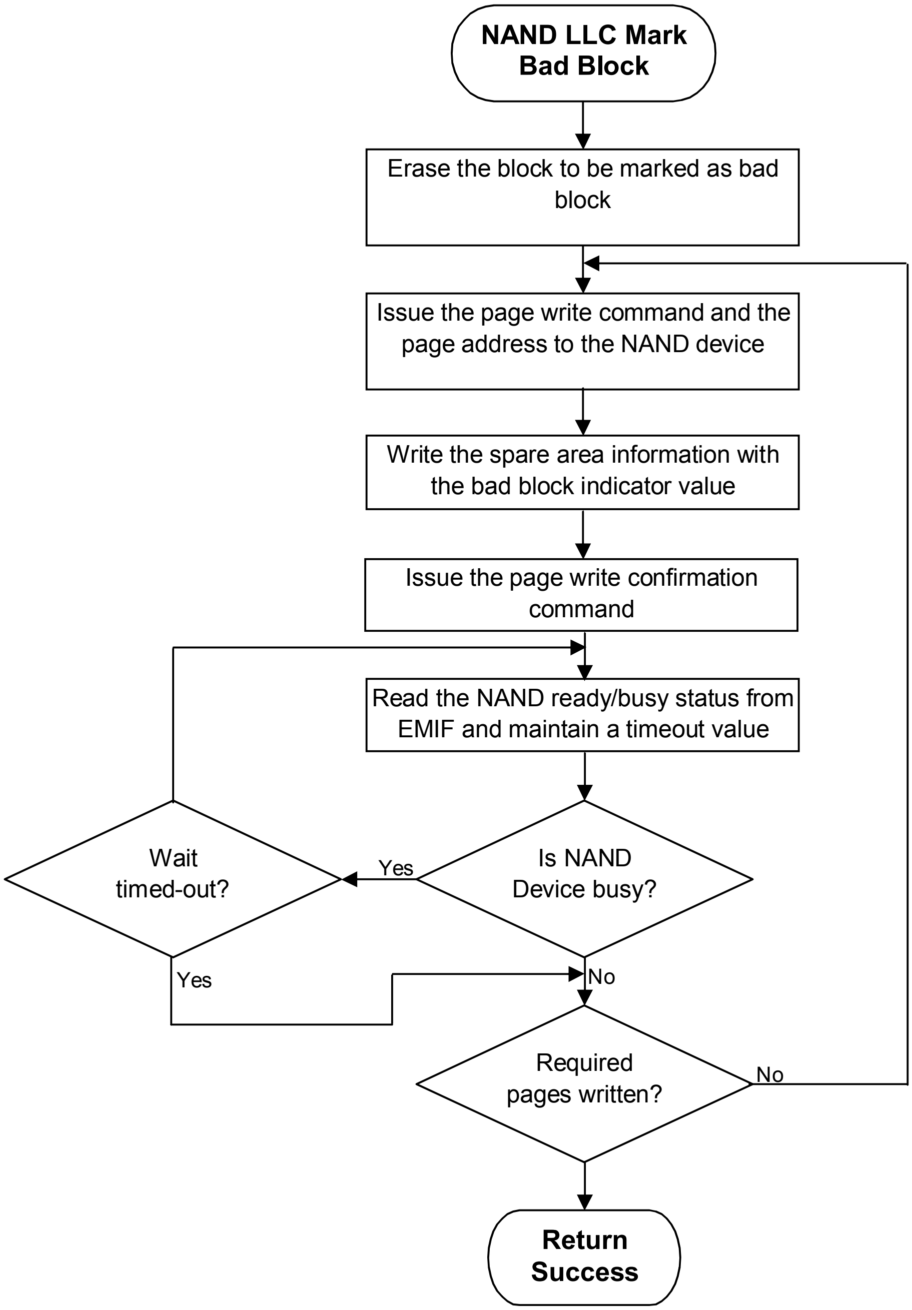
NAND LLC Mark Bad Block
Erase the block to be marked as bad block
Issue the page write command and the page address to the NAND device
Write the spare area information with the bad block indicator value
Issue the page write confirmation command
Read the NAND ready/busy status from EMIF and maintain a timeout value
Is NAND Device busy?
Yes
Wait timed-out?
Yes
No
Required pages written?
No
Return Success

Figure 13: NAND LLC module mark bad block procedure

The LLC NAND module mark bad block procedure is described as below:

1. Erase the block that should be marked as bad block. Since the block is being marked as bad, the status of the erase operation is ignored.
2. The bad block mark is placed in the first, second and the last page of the block. Steps X to Y are repeated to mark all the three pages with the bad block information.
3. Issue the NAND device page write command followed by the address of the page to be written.

4. Write the contents of the spare area to the NAND device with the bad block indicator value. Issue the page write confirmation command.
5. From the EMIF registers, determine if the NAND device is in ready or busy state. A timeout count should also be maintained to terminate in case NAND does not change to ready state. If the NAND device is ready, proceed to step 6.
6. If the device is not ready and if the wait has timed-out, then terminate the wait and proceed to step 7.
7. Have all the first, second and the last page of the block been written with bad block indicator value? If not, repeat from step 3.

7.6 Spare Area Read procedure of NAND LLC module

Figure 14 illustrates the spare area read procedure used by the LLC NAND module.

`

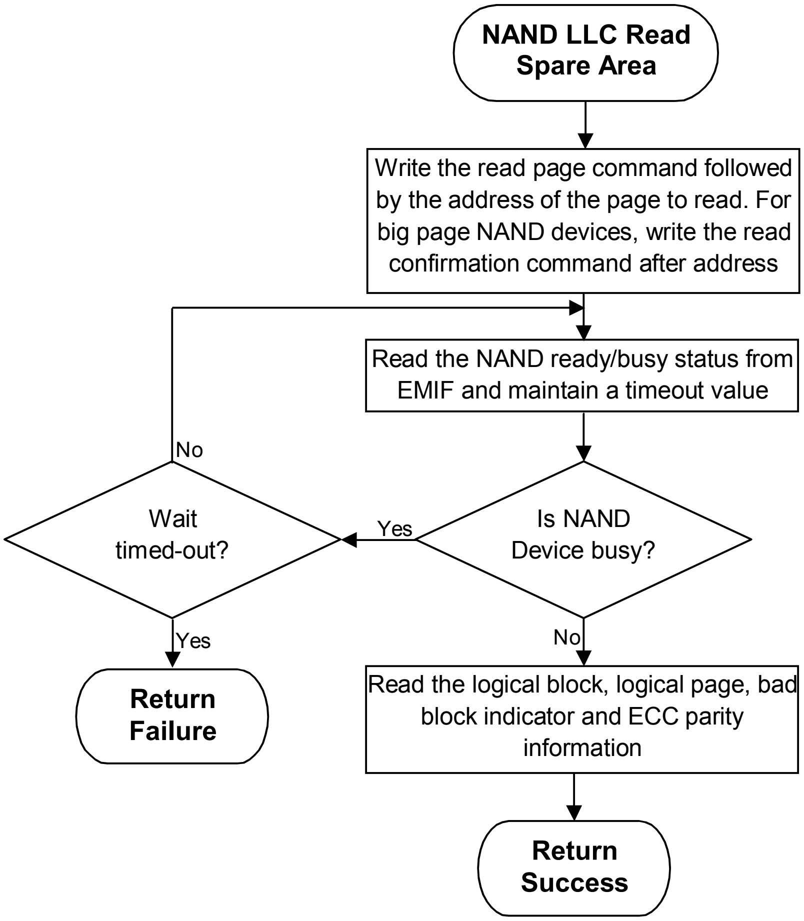
NAND LLC Read Spare Area
Write the read page command followed by the address of the page to read. For big page NAND devices, write the read confirmation command after address
Read the NAND ready/busy status from EMIF and maintain a timeout value
Is NAND Device busy?
Yes
Wait timed-out?
No
Yes
Return Failure
No
Read the logical block, logical page, bad block indicator and ECC parity information
Return Success

Figure 14: NAND LLC module spare area read procedure

The LLC NAND module spare area read procedure is described as below.

1. Issue the NAND device page read command followed by the address of the page to be written. For big page devices, issue the read-confirm command after writing the page address.
2. From the EMIF registers, determine if the NAND device is in ready or busy state. A timeout count should also be maintained to terminate in case NAND

does not change to ready state. If the NAND device is ready, proceed to step 7.

3. If the device is not ready and if the wait has timed-out, then terminate the wait and return from the page read procedure with a error value.
4. Read the logical block number, logical page number, bad block indicator value and the ECC parity values from the spare area of the page.
5. Return from the NAND spare area read procedure with success as the status.

7.7 IOCTL command execution procedure of NAND LLC module

The LLC NAND module IOCTL command execution procedure is described as below:

1. Validate the IOCTL command specified. If the IOCTL command is not supported, return to caller with appropriate error code.
2. Execute the specified IOCTL command.
3. Return to caller with the status of executing the IOCTL command.

7.8 De-initialization procedure of NAND LLC module

The LLC NAND module de-initialization procedure is described as below:

1. Release any system resources such as EDMA channels and semaphores acquired during the LLC NAND module initialization.
2. Return to caller with the status of releasing the system resources.

7.9 Spare Area Assignment in NAND devices

The usage of the spare area of the small page 8-bit NAND page devices is illustrated in Table 24.

Data	Position
Logical block address	Bytes 0 to 2
Logical page address	Byte 3
Bad Block Mark	Byte 4 and 5
ECC parity data	Starting from byte 6

Table 24: Spare Area data assignment for 8-bit small page devices.

The usage of the spare area of the other organizations of NAND devices is illustrated in Table 25.

Data	Position
Bad Block Mark	Bytes 0 and 1
Logical block address	Bytes 2 to 4
Logical page address	Byte 5
ECC parity data	Starting from byte 6

Table 25: Spare Area data assignment for 16-bit small/big block and 8 bit big block devices.

8 Revision History

Version #	Date	Author Name	Revision History
0.1	07/01/2009	Vipin Bhandari	Document Created