Texas Instruments

DSP/BIOS NAND Device Driver

User's Manual

May 08 *Platform Software Group*

Document Version 1.11

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:
Texas Instruments
Post Office Box 655303, Dallas, Texas 75265

Read This First

About This Manual

This User's Manual serves as a software programmer's handbook for working with the ***DSP/BIOS NAND Device Driver.*** This manual provides necessary information regarding how to effectively install, build and use ***DSP/BIOS NAND Device Driver*** in user systems and applications.

This manual provides details regarding how the ***DSP/BIOS NAND Device Driver*** is architected, its composition, its functionality, the requirements it places on the hardware and software environment where it can be deployed, how to customize/ configure it to specific requirements, how to leverage the supported run-time interfaces in user's own application etc.,

This manual also provides supplementary information regarding steps to be followed for proper installation/ un-installation of the ***DSP/BIOS NAND Device Driver***. Also included are appendix sections on related Glossary, Web sites and Pointers for gathering further information on the ***DSP/BIOS NAND Device Driver***.

Terms and Abbreviations

Term	Description
IP	Intellectual Property
OS	Operating System
API	Application Programmer's Interface
CSL	TI Chip Support Library – primitive h/w abstraction
DDC	Device Driver Core - TI terminology for portion of device driver that is abstracted of any given OS
DDA	Device Driver Adaptation - TI terminology for portion of device driver that is specific to target OS. This constitutes "adaptation" of the generic DDC to identified target OS.
ISR	Interrupt Service Routine
LLC	Low Level Controller

Related Documentation

Internal

This is a list of documents that are TI Proprietary and Strictly Private. Exposure to audience outside TI will need due considerations and approvals from TI Legal authorities.

- Nand Hardware Module Specifications
- SOC Technical Reference Manual
- PSP Framework Architecture 1.0

External

- None

Trademarks

Revision History

Date	Author	Revision History	Version
8th August 2006	Kiran Sutariya	Created the Document	1.0
8th September,2006	Kiran Sutariya	Added protected region details.	1.1
5th October,2006	Kiran Sutariya	Updated for Presilicon 0.2.0 release	1.2
1st December, 2006	Kiran Sutariya	Modified for the release 0.3.0	1.3
2nd January, 2007	Rinkal Shah	Review comments closed	1.4
16th January, 2007	Rinkal Shah	Bios version changed	1.5

27th January, 2007	Rinkal Shah	CCS version changed	1.6
20th April, 2007	Rinkal Shah	Updated the for release 0.7.0	1.7
7th May, 2007	Rinkal Shah	Functions and IOCTL details added	1.8
22 June 2007	Anuj Aggarwal	Updated the for GA patch release 1.00.01	1.9
29 June 2007	Amit Chatterjee	Modified Release Version	1.10
18 July 2007	Rinkal Shah	Modified Release Version	1.11
20 May 2008	Chandan Nath	Updated for adding compiler switches in build options	1.12

Contents

Tables

Chapter 1

Introduction

This chapter introduces the ***DSP/BIOS NAND Device Driver*** to the user by providing a brief overview of the purpose and construction of the ***DSP/BIOS NAND Device Driver*** along with hardware and software environment specifics in the context of ***DSP/BIOS NAND Device Driver*** deployment.

1.1 Overview

This section describes the functional scope of the ***DSP/BIOS NAND Device Driver*** and its feature set. The section also details the various deployment environments, hardware and software, that the ***DSP/BIOS NAND Device Driver*** is presently supported on. The chapter introduces the system architecture of the ***DSP/BIOS NAND Device Driver*** to the user along with the functional decomposition and run-time specifics regarding deployment of ***DSP/BIOS NAND Device Driver*** in user's application.

1.1.1 Supported Services and features

The ***DSP/BIOS NAND Device Driver*** provides the following functional services and features:

- Provides Sync IO mechanism
- Operates in Polled mode and DMA mode
- Modeled after TI Device Driver Archiectrure for Storage Class Devices that allow for easy porting and customization.

1.1.2 System Requirements

Details about the tools and the BIOS version that the driver is compatible with can be found in the system Release Notes.

Chapter 2

Installation Guide

The NAND Driver shall be available as a package by itself or part of another PSP package. Follow the installation instructions provided along with the package.

2.1 Component Folder

Upon installing the NAND driver the following directory structure is found in the driver's directory.

Figure 1. NAND Driver Directory Structure

This top level nand folder contains nand driver psp header file and XDC package files (package.bld, package.xdc and package.xs)

- **build:** This folder contains nand driver library project file. The generated driver library shall be included in the application where NAND driver have to be used.
- **docs:** This folder contains architecture document, datasheet, release notes and user guide.

  Architecture document contains the driver details which can be helpful for the developers as well as consumers to understand the driver design.

  Datasheet gives the idea about the memory consumption by the driver and description of the top level APIs.

  Release Note gives the details about system requirements, steps to Install/Uninstall the package.This document list the known issues of the driver.

  User Guide provides information about how to use the driver. It contains description of sample applications which guide the end user to make their applications using this driver.

- **Lib:** This folder contains libraries generated in all the configuration modes(debug, idebug, irelease and release)
- **Package:** This folder contains files generated by XDC tool.
- **Src:** This folder contains nand driver source files. It also contains header files that are used by the driver.

2.2 Build

This section describes for each supported target environment, the applicable build options, supported configurations and how selected, the featured capabilities and how enabled, the allowed user customizations for the software to be installed and how the same can be realized.

The component might be delivered to user in different formats:

- Source-less ie., binary executables and object libraries only
- Source-inclusive ie., The entire source code used to implement the driver is included in the delivered product
- Source-selective ie., Only a part of the overall source is included. This delivery mechanism might be required either because; certain parts of the driver require source-level extensions and/or customization at the user's end or because, specific parts of the driver is exposed to user at the source-level to insure user's software development.

When source is included as part of the product delivery, the CCS project file is provided as part of the package. When object format is distributed, the driver header files are part of the "inc" folder and the driver library is provided in /drivers/lib folder.

2.2.1 Build Options

Following compiler switches are used to compile for different options.

- **_DEBUG**

This is used as a flag to compiler whether to include the debug statements inserted in the code into the final image. This flag helps to build DEBUG image of the program. For RELEASE images this is not passed to the compiler.

- **CHIP_DM6437**

This macro is used for select DM6437 specific header files.

- **NAND_INSTRUMENTATION_ENABLED**

This macro is used to enable instrumentation related code.

- **PSP_NAND_DEBUG**

This macro is used for debugging purposes. This macro enables display of general debug messages in the NAND driver code.

- **NAND_DBG_LEVEL_1**

This macro is used for debugging purposes. This macro enables display of debug messages for logical page read and writes operations.

- **DDC_NAND_POWER_SAFE**

This macro enables the software features that safeguard the NAND driver from power failures.

- **DDC_NAND_BBM**

This macro is used to enable the support for bad block management. If this macro is not enabled, bad block management code is not included in the compilation.

- **NAND_DBG_LEVEL_3**

This macro is used for debugging purposes only. It is used to display the physical page number mapped to the logical page number in FTL page read and write functions.

- **DDC_NAND_8_BIT**

This macro statically (compilation time) selects 8-bit data bus support. If this macro is defined, only 8-bit NAND devices are supported. If this macro is not defined, only 16-bit NAND devices are supported.

- **NAND_DBG_LEVEL_2**

This macro is used for debugging purposes only. It is used to display a message on the terminal when a read fails.

- **DDC_NAND_ENABLE_ECC**

This macro is used when software based ECC is required. In DM6437, this macro is disabled since the ECC is always calculated by hardware.

Chapter 3

DSP/BIOS NAND

This chapter discusses the DSP/BIOS NAND Device Driver functions, data structures, enumerations and macros for the List module.

3.1 Functions

This section lists the functions available in the PSP NAND module.

3.1.1 PSP_nandInit

PSP_Result PSP_nandInit (PSP_NandInstanceId *instanceId*
const PSP_NandConfigInt * *config*
)

This function is called by the application to initialize the nand device and the driver. It prepares the background for the driver to launch.

Parameters:

instanceId [IN] NAND instance number
config [IN] driver configuration information

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

PSP_E_INVAL_STATE – if the driver is in an invalid state (any state other than deleted for this function)

Example:

```
PSP_Result result;
PSP_NandConfig   config;

config.opMode       = PSP_OPMODE_DMAINTERRUPT;
config.inputClkFreq = nandClkFreq;
config.eraseAtInit  = FALSE;

/* Initialize the driver */
result = PSP_nandInit(0, &config);
```

3.1.2 PSP_nandTerminate

PSP_Result PSP_nandTerminate (PSP_NandInstanceId *instanceId*)

This function de-initializes and deletes the driver for a particular instance.

Parameters:

instanceId [IN] NAND instance number

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Result result;
result = PSP_nandTerminate(0);
```

3.1.3 PSP_nandOpen

PSP_Handle PSP_nandOpen (PSP_NandInstanceId *instanceId*)

This function opens the device for transaction. The API reads the device id of the nand device available and initializes the configuration structures depending on the device type found.

Parameters:

instanceId [IN] Nand instance number

Returns:

PSP_Handle – if the operation is successful

NULL – if the operation is failed

Example:

```
PSP_Handle hNand;
hNand = PSP_nandOpen(0);
```

3.1.4 PSP_nandClose

PSP_Result PSP_nandClose (PSP_Handle *hNand*)

This API closes the driver that was opened by the open call. Any specific objects that were created/used by the open call are freed here.

Parameters:

hNand [IN] Nand Driver Instance Handle

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result result;

hNand = PSP_nandOpen(0);
……
Result = PSP_nandClose(hNand)
```

3.1.5 PSP_nandReadSync

PSP_Result PSP_nandReadSync (PSP_Handle *hNand*
Uint32 logSector
Uint16 * pReadBuff
Int size
Int * xferActual
)

This function reads number of sectors specified by 'size' starting from the NAND device logical sector specified by 'logSector', to 'buf', depending on the mode configured by the application. 'xferActual' holds the actual number of bytes read.

Parameters:

hNand	*[IN] NAND Driver Instance Handle*
logSector	*[IN] Sector to be read*
pReadBuff	*[OUT] Character Buffer pointer - where read data is to be stored*
size	*[IN] Number of sector to be read*
xferActual	*[OUT] Number of bytes actually transacted*

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

PSP_E_INVAL_STATE – if the driver is in an invalid state

Example:

```
PSP_Handle hNand;
Uint32 logSector = 100;
Uint16 pReadBuff[512];
Int size = 1;
Int xferActual;
PSP_Result Result;
```

```
Result = PSP_nandReadSync( hNand, logSector, pReadBuff, size, &xferActual
);
```

3.1.6 PSP_nandWriteSync

PSP_Result PSP_nandWriteSync (PSP_Handle *hNand*
Uint32 logSector
Uint16 * pWriteBuff
Int size
Int * xferActual
)

This function writes number of sectors specified by ‘size’ starting from the NAND device logical sector specified by ‘logSector’ from 'buf', depending on the mode configured by the application. 'xferActual' holds the actual number of bytes written.

Parameters:

hNand	*[IN] NAND Driver Instance Handle*
logSector	*[IN] Sector to be written*
pWeadBuff	*[OUT] Character Buffer pointer - where data to be written is stored*
size	*[IN] Number of sector to be written*
xferActual	*[OUT] Number of bytes actually transacted*

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

PSP_E_INVAL_STATE – if the driver is in an invalid state

Example:

```
PSP_Handle hNand;
Uint32 logSector = 100;
Uint16 wData[512];
Int size = 1;
Int xferActual;
PSP_Result Result;
Int bufIndex;

for (bufIndex = 0; bufIndex < (512/(2u)); bufIndex++)
{
    WData[bufIndex] = ((0xAAu) << (8u)) | bufIndex;
}


Result = PSP_nandWriteSync ( hNand, logSector, wData, size, &xferActual
);
```

3.1.7 PSP_nandErase

PSP_Result PSP_nandErase (PSP_Handle *hNand*)

This API erases all the blocks in the NAND and the action cannot be un-done. From a File system perspective, this function formats the NAND device.

Parameters:

hNand *[IN] NAND Driver Instance Handle*

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result Result;
result =  PSP_nandErase(hNand);
```

3.1.8 PSP_nandAddDevice

PSP_Result PSP_nandAddDevice (const PSP_NandFlashDev * *cfgParams*)

This API sets the following configuration parameters: Bytes per page, Pages per block, Size of NAND, Bus Width: 8/16 bit. These parameters shall be used when the device ID read from the device doesn't match the one in the known 'list'.

Parameters:

cfgParams *[IN] Configuration parameter structure*

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_NandFlashDev  cfgParams;
PSP_Result Result;
cfgParams.name         = "NAND 128MB 8BIT"; /* user given name*/
cfgParams.id           = (0xFEu);          /* device ID*/
cfgParams.bytesPerPage = (1024u);          /* bytes per page in bytes*/
cfgParams.chipSize     = (256u);           /* chip size in MB*/
cfgParams.nand8Bit     = 0;                /* 0- 8bit; 1- 16 bit*/
cfgParams.eraseSize    = (16384u);         /* Block/erase size in bytes*/

Result = PSP_nandAddDevice (&cfgParams);
```

3.1.9 PSP_nandIoctl

PSP_Result PSP_nandIoctl	**(**	**PSP_Handle**	***hNand***
		PSP_NandIoctlCmd	**cmd**
		Void *	**cmdArg**
		Void *	**param**
	)		

This function supports various IOCTLs for the NAND controller. All the supported IOCTLs are listed in section 3.2

Parameters:

hNand	*[IN] NAND Driver Instance Handle*
cmd	*[IN] IOCTL command*
cmdArg	*[IN/OUT] Arguments, if any, for the command*
param	*[IN] IOCTL specific parameter*

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result result;
Uint32 sectorSize;

/* Get the sector size */
result = PSP_nandIoctl( hNand,
                        PSP_NAND_IOCTL_GET_SECTOR_SIZE,
                        (void*)&sectorSize,
                        0);
```

3.2 Nand module IOCTLs

This section illustrates various IOCTL commands for the NAND controller.

3.2.1 PSP_NAND_IOCTL_GET_OPMODE

This IOCTL returns the operation mode for the session.

Synopsis:

```
PSP_Result  PSP_nandIoctl(
                PSP_Handle          hNand,
                PSP_NandIoctlCmd    cmd,
                void*               cmdArg,
                void*               param);
```

Arguments:

PSP_Handle – Nand driver instance handle

cmd - IOCTL command (PSP_NAND_IOCTL_GET_OPMODE)

cmdArg - OPMODE returned by the driver. This field holds any value of the enum PSP_OpMode:

```
typedef enum
{
   PSP_OPMODE_POLLED      = (0U),
   PSP_OPMODE_INTERRUPT   = (1U),
   PSP_OPMODE_DMA         = (2U),
   PSP_OPMODE_DMAINTERRUPT = (3U)
}PSP_OpMode;
```

param – NULL.

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result result;
PSP_OpMode opMode;

/* Get the OPMODE */
result = PSP_nandIoctl( hNand,
                        PSP_NAND_IOCTL_GET_OPMODE,
                        (void*)&opMode,
                        0);
```

3.2.2 PSP_NAND_IOCTL_GET_NAND_SIZE

This IOCTL returns nand size in sectors.

Synopsis:

```
PSP_Result  PSP_nandIoctl(
                PSP_Handle          hNand,
                PSP_NandIoctlCmd    cmd,
                void*               cmdArg,
                void*               param);
```

Arguments:

PSP_Handle – Nand driver instance handle

cmd - IOCTL command (PSP_NAND_IOCTL_GET_NAND_SIZE)

cmdArg – nand size returned by the driver. This should point to a Uint32 variable allocated by the application.

param – NULL.

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result result;
Uint32 nandSize;

/* Get the nand size */
result = PSP_nandIoctl( hNand,
                        PSP_NAND_IOCTL_GET_NAND_SIZE,
                        (void*)& nandSize,
                        0);
```

3.2.3 PSP_NAND_IOCTL_GET_SECTOR_SIZE

This IOCTL returns nand sector size in bytes.

Synopsis:

```
PSP_Result  PSP_nandIoctl(
                PSP_Handle          hNand,
                PSP_NandIoctlCmd    cmd,
                void*               cmdArg,
                void*               param);
```

Arguments:

PSP_Handle – Nand driver instance handle

cmd - IOCTL command (PSP_NAND_IOCTL_GET_SECTOR_SIZE)

cmdArg – nand sector size returned by the driver. This should point to a Uint32 variable allocated by the application.

param – NULL.

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result result;
Uint32 sectorSize;

/* Get the sector size */
result = PSP_nandIoctl( hNand,
                        PSP_NAND_IOCTL_GET_SECTOR_SIZE,
                        (void*)&sectorSize,
                        0);
```

3.2.4 PSP_NAND_IOCTL_CHECK_NAND

This IOCTL returns detected nand device type.

Synopsis:

```
PSP_Result  PSP_nandIoctl(
                PSP_Handle          hNand,
                PSP_NandIoctlCmd    cmd,
                void*               cmdArg,
                void*               param);
```

Arguments:

PSP_Handle – Nand driver instance handle

cmd - IOCTL command (PSP_NAND_IOCTL_CHECK_NAND)

cmdArg – nand type defined by the following enum:

```
typedef enum
{
   LLC_NAND_NONE = 0u,
   /**< No NAND detected              */
   LLC_NAND_BIG_BLOCK,
   /**< Big Block NAND detected       */
   LLC_NAND_SMALL_BLOCK,
   /**< Small Block NAND detected     */
   LLC_NAND_INVALID
   /**< Unknown/Invalid NAND          */
} LLC_NandType;
```

param – NULL.

Returns:

PSP_SOK – if the operation is successful

PSP_E_INVAL_PARAM – if any parameter is invalid

Example:

```
PSP_Handle hNand;
PSP_Result result;
Uint32 nandType;

/* Get the sector size */
result = PSP_nandIoctl( hNand,
                        PSP_NAND_IOCTL_CHECK_NAND,
                        (void*)&nandType,
                        0);
```

3.3 Symbolic Constants and Enumerated Data types

The following configuration defines are provided:

Table 1. Configuration Defines

Defines	Description
PSP_NAND_NUM_INSTANCES	Number of Instances of NAND hardware modules supported by the driver – the number of instances of the NAND will depend upon the SOC for which this driver is provided. Currently for DM6437/C6424 it is 1.
PSP_NAND_NUM_CHANNELS	DO NOT CHANGE THIS VALUE in the code – Number of channels supported.
PSP_NAND_NUM_DEVICES_PER_CONTROLLER	Max no. of slaves that can be connected on each NAND device instance. Currently for DM6437/C6424 it is 1.
PSP_NAND_DEVICE_POLL_INTERVAL	Internal Poll time in milliseconds – used when driver operates in polled mode – the driver uses this time period as a parameter to the delay function during polling.

Table 2. OpMode Enumerations

Defines	Description
IOM_OPMODE_POLLED (0)	POLLED Mode
IOM_OPMODE_INTERRUPT(1)	Interrupt Mode (Not supported)
IOM_OPMODE_DMAINTERRUPT(2)	DMA Mode
IOM_OPMODE_DMA(3)	DMA Mode (Not supported)

- The file dda_nandCfg.c contains the initial configuration parameters used by the driver. Configuration Parameters for each instance are set in this data structure. The configuration data variable "gDDA_NandHwConfig" is populated with the initial configuration data.
- Driver supports protected region functionality, there is a #define DDC_NAND_PROTECTED_AREA in file ddc_nand.h which describes size of protected region in terms of page size. Currently it is configured to 0. The protected region should be in the multiple of 32 sectors. The protected region reserves area only at the start of the NAND device.

3.4 Run-time Configuration Data Structure

This section summarizes all users visible data structure elements pertaining to the ***DSP/BIOS NAND Device Driver*** configuration interfaces.

The file psp_nand.h has the PSP_NandConfig data structure that is passed to the initialization function of the driver. The params are explained below:

Table 3. Configuration Data Structure

Parameter	Description
opMode	Operational mode of the driver – Polled / DMA based.
inputClkFreq	Clock Frequency of NAND.
eraseAtInit	Erase all NAND devices on board during initialization if value is 1.

The file psp_nand.h also includes the PSP_NandFlashDev data structure that is used to initialize and configure new device at run-time. This is passed to the PSP_nandAddDevice function to set the new device configuration. This needs to be done every time the target is run. Alternatively, the same information can be added in the dda_nandCfg.c file (and re-compiled), which is searched at the NAND initialization phase for known device IDs.

The params are explained below:

Table 4. NAND Flash Data Structure for run-time configuration

Parameter	Description
Name	Name of the device
Id	Device ID of the chip (typically the second byte)
bytesPerPage	Number of bytes per page for this new chip, excluding the spare area.
chipSize	Total size of the chip in MBs; excluding the spare area. **Note:** 2GB is 2048 MB
eraseSize	Typically erase is block-wide. Hence this parameter is the block size in bytes, excluding the spare area.
nand8Bit	Organization of NAND; 8 or 16bit wide.

Table 5. OpMode Enumerations

Defines	Description
IOM_OPMODE_POLLED (0)	POLLED Mode
IOM_OPMODE_INTERRUPT(1)	Interrupt Mode (Not supported)
IOM_OPMODE_DMAINTERRUPT(2)	DMA Mode
IOM_OPMODE_DMA(3)	DMA Mode (Not supported)

- The file dda_nandCfg.c contains the initial configuration parameters used by the driver. Configuration Parameters for each instance are set in this data structure. The configuration data variable “gDDA_NandHwConfig” is populated with the initial configuration data.

- Driver supports protected region functionality, there is a #define DDC_NAND_PROTECTED_AREA in file ddc_nand.h which describes size of protected region in terms of page size. Currently it is configured to 0. The protected region should be in the multiple of 32 sectors. The protected region reserves area only at the start of the NAND device.

Chapter 4

Porting

This chapter discusses how to port ***DSP/BIOS NAND Device Driver*** to other supported target platforms.

4.1 Getting Started

The ***DSP/BIOS NAND Device Driver*** is based upon PSP Framework architecture making portability and re-usability as prime requirements. Based upon the architecture, the driver sources are divided into Low-level controller/ CSL, Device Driver Core (DDC) and Driver Adaptation (DDA) to a given OS.

The low level controller provides the abstraction needed for the driver to be re-usable even if the underlying Nand hardware is changed. For porting to any Nand controller, the low-level controller module should be ported.

The driver adaptation layer needs to be ported when moving the driver from one OS to another (with the same Hardware Nand module).

4.2 Sources needing re-targeting

The following source files are required for driver porting:

Device Driver Adaptation (DDA):

dda_nand.h, dda_nand.c – Contains the “common” functions required for any adaptation – normally these files should remain common so that any os specifics are taken care of in another file.

dda_nandBios.h, dda_nandBios.c – Contains DSP/BIOS specific functions that are required to complete the OS port – for porting the driver to another OS, in most cases, implementing the functions provided in this file for the given OS is sufficient to get the driver running.

dda_nandCfg.c – Contains the configuration information required for the driver – this file does not need to change during OS ports

Low Level Controller (LLC/CSL):

llc_nandIf.h, llc_nandTypes.h – Contains the definitions that are used to abstract the low level controller so that the DDC can be agnostic of the underlying controller. This file needs to be modified only when porting the driver for another Nand controller.

llc_nand.c – Contains the initialization and hardware setup functions for the controller – these functions need to be ported and implemented when porting the driver for another Nand controller.

4.3 Dependency of Sample application:

Following Components needs to be linked for successful build and functionality of the application.

- NAND
- PAL_OS
- SoC specific PAL_SYS
- EDMA3

Pragma directives used in the sample application:

The RData and WData are used by the application and need to be cache aligned at 128 bytes.