Texas Instruments

NAND Device Driver

Architecture/Design Document

December 2007

Platform Software Group

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

About This Document

This document discusses the TI device driver architecture for NAND Device Driver for DM6437/C6424 SOC. The target audience includes device driver developers from TI as well as consumers of the driver.

Trademarks

The TI logo design is a trademark of Texas Instruments Incorporated. All other brand and product names may be trademarks of their respective companies.

This document contains proprietary information of Texas Instruments. The information contained here in is not to be used by or disclosed to third parties without the express written permission of an officer of Texas Instruments Incorporated.

Related Documents

- NAND Driver Documentation
- PSP Framework Architecture 1.0

Notations

None

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	TI terminology for portion of device driver that is abstracted of any given OS
DDA	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
EMIF	External Memory Interface
DMA	Direct Memory Access

Revision History

Date	Author	Comments	Version
8th August 2006	Kiran Sutariya	Created the document	1.0
27th September 2006	Kiran Sutariya	Changed after removing streaming and storage layer.	1.1
5th October, 2006	Kiran Sutariya	Changed after removing streaming and storage layer.	1.2
30th November, 2006	Kiran Sutariya	Changed for release version 0.3.0	1.3
16th January, 2007	Rinkal Shah	Bios version changed	1.4
27th January, 2007	Rinkal Shah	CCS version changed	1.5
June 22, 2007	Anuj Aggarwal	Bios version changed	1.6
June 29,2007	Amit Chatterjee	Modified Release Version	1.7
July 18, 2007	Rinkal Shah	Modified Release Version	1.8

Table of Contents

1 System Context 1

1.1 Hardware 1

1.2 Software 1

1.2.1 Operating Environment and dependencies 1

1.3 Design Philosophy 1

2 NAND Driver Software Architecture 2

2.1 Static View 2

2.1.1 Functional Decomposition 2

2.1.2 H/W Device Specific Layer (LLC) 3

2.1.3 Device Driver Core functionality (DDC) 3

2.1.4 OS Specific Device Driver Adaptation (DDA) 5

2.1.5 Platform Abstraction Layer for OS services (PALOS) 5

2.1.6 Component Interfaces 6

2.2 Dynamic View 7

2.2.1 The Execution Threads 7

2.2.2 Driver Creation/Initialization 7

2.2.3 Driver Open 8

2.2.4 IO Control 8

2.2.5 IO Access 8

2.2.6 Driver Close 8

1 System Context

The NAND device driver architecture presented in this document is situated in the context of DM6437/C6424 SOC.

1.1 Hardware

The NAND module used in DM6437/C6424 SOC is external and can be interfaced through EMIF.

1.2 Software

The document provides an overall understanding of the TI NAND device driver architecture.

1.2.1 Operating Environment and dependencies

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

1.3 Design Philosophy

Please refer section 1.2 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2 of C6424_BIOS_PSP_User_Guide.doc for C6424 for Design Philosophy.

2 NAND Driver Software Architecture

This chapter deals with the overall architecture of TI 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.

2.1 Static View

2.1.1 Functional Decomposition

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities already discussed in section 1.3. In the following sub-sections, each of these functional sub-components of the device driver is further elaborated.

Please refer to section 1.2.1 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.1 of C6424_BIOS_PSP_User_Guide.doc for C6424 for diagrammatic representation.

The central portion shown constitutes the mainline NAND driver component; the surrounding modules contained in the dotted area constitute the supporting system components. The modules in the later part, do not specifically deal with NAND, but assist the driver by providing other abstracted (OS and H/W) and utility services as shall be discussed in following sub-sections.

2.1.2 H/W Device Specific Layer (LLC)

The LLC forms the lower most, h/w specific under-pinning of the TI device driver. It consists of two parts:

- **CSL Register Layer:** Please refer section 1.2.2 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.2 of C6424_BIOS_PSP_User_Guide.doc for C6424 for CSLr description.
- **LLC Layer**: Please refer section 1.2.2 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.2 of C6424_BIOS_PSP_User_Guide.doc for C6424 for Hardware Layer explanation.

To improve componentization of NAND device driver, the LLC module for NAND device is clubbed with the other driver modules (DDA and DDC).

2.1.3 Device Driver Core functionality (DDC)

Please refer section 1.2.3 of DM6437_BIOS_PSP_User_Guide.doc for DM6437 and section 1.2.3 of C6424_BIOS_PSP_User_Guide.doc for C6424 for DDC Layer description.

The DDA/DDC inter-operation is further discussed later in section on dynamic view.

The following table outlines the basic interfaces published by DDC.

DDC Implemented I/F for DDA	DESCRIPTION
PAL_Result nandCreateInstance(Int instId, DDA_Handle hDDA, DDC_Handle *hDDC, Ptr param);	Driver Instance Creation: Tasks performed here include - allocation of memory/OS Resources for the DDC instance, container object. Its possible that the desired memory is declared statically, but specific reservation for the identified device is done here. This function should not touch the h/w.
PAL_Result nandDelInst(DDC_Handle hDDC, Ptr param);	Driver Instance Deletion: This service frees all memory originally claimed by the nandCreateInstance. It also relinquishes any OS resources acquired during create operation. After this call, the driver instance ceases to exist in the system altogether
PAL_Result nandInit(DDC_Handle hDDC, Ptr param);	Driver Instance Initialization: This service initializes the device driver instance to a reasonable startup condition. It is possible that the connected h/w device is discovered during this call and appropriate initializations done. In case power management is effected, this call places the driver in a stand by, low power mode.

PAL_Result nandDeinit(DDC_Handle hDDC, Ptr param);	Driver Instance DeInitialization: This service is counter to the previously described nandInit service. It basically undoes the operations performed during Initialization to extent possible so as to return the driver instance to its original state, right after creation. In most cases, this function is likely to be a trivial one. In case power management is effected, this call returns the driver to its stand by or low power mode corresponding to state during initialization.
PAL_Result nandOpen(DDC_Handle hDDC, Ptr param);	Open device driver controller: This service marks the start of use of the device driver instance. Operations performed include completing the h/w configuration to desired operating mode and finalization of internal driver book-keep. After this call, the device driver is declared to be in active state, interrupts are enabled and data transfers can proceed. Often times, plugging of ISRs are deffered to this stage to minimize pre-mature reservation of CPU vector slots. When there are far too many interrupts in the system, but only few are relevant for any given scenario, grabbing CPU vector slots at nandInit time itself can possibly lead to scarcity of vectors.

DDC Implemented I/F for DDA	DESCRIPTION
PAL_Result nandClose(DDC_Handle hDDC, Ptr param);	Close device driver controller: This service is counter to the previously described nandOpen It releases any resources claimed by driver during the open operation. After this call, the driver remains dormant and no data transfer is possible unless it is re-opened.
PAL_Result nandControl(DDC_Handle hDDC, Int cmd, Ptr cmdArg, Ptr param);	Driver IO Control: This service provides classic *IOCTL* kind of capability to control the device h/w and/or the driver internal state/behaviour

DDC Implemented I/F for DDA	DDC Implemented I/F for DDA
PAL_Result nandReadSync(DDC_NandObj *hDDC, Uint16 *buf, Const Int32 xferRequest, Uint32 timeout, Ptr param, Int32 *xferActual);	Synchronous Read: Interface that reads data from a slave in a synchnonous manner, when the controller is operating in slave mode.
PAL_Result nandWriteSync(DDC_NandObj *hDDC, Uint16 *buf, const Int32 xferRequest, Uint32 timeout, Ptr param, Int32 *xferActual);	Synchronous Write: Interface that writes data to a slave in a synchnonous manner, when the controller is running in transmit mode.

The above generic DDC interface is extended and realized in a device specific fashion by the driver for the particular device. In our case, we'll see how the NAND implements a concrete interface from base functionality prescribed above.

2.1.4 OS Specific Device Driver Adaptation (DDA)

As discussed above, the DDC is not complete unless it is supplemented by the DDA. The DDA "adapts" the driver core to the specific OS. DDA implements aspects such as – threading model for transaction processing, Interrupts registration and de-registration and handshaking with OS etc., The DDA has full visibility to underlying OS services and is custom-built for a given OS.

While the DDA is primarily intended for presenting an OS manifest to the underlying DDC, it is also possible that the DDA upper-edge interface (user level) imbibes the semantics of any pre-specified Framework, if one exists. This is necessary to prevent undue overheads in system integration.

For further details on DDA/DDC interactions, refer to section on device driver dynamic views later in this document.

2.1.5 Platform Abstraction Layer for OS services (PALOS)

Please refer section 1.2.5 of DM6437_BIOS_PSP_User_Guide.doc for PALOS Layer description for DM6437 and section 1.2.5 of C6424_BIOS_PSP_User_Guide.doc for PALOS Layer description for C6424.

2.1.6 Component Interfaces

In the following subsections, the interfaces implemented by each of the sub-component are specified.

1.1.1.1 DDA Interface

The DDA constitutes the Device Driver Manifest to Application. This adapts the Driver Core (DDC) to DSP/BIOS.

The user of device driver will only need bother about the DDA interface, especially the upper-edge services exposed to the Application/OS. All other interfaces discussed later in this document are more of interest to people developing/maintaining the device driver.

The DDA can be modified to re-target Driver and/or customize to specific Applications framework by doctoring the upper-edge services.

The PSP_nandInit () populates static settings in driver object creates the necessary interrupt handler, attaches the Driver Core interfaces. All these operations in effect, constitute the “loading” of NAND Driver implementation.

1.1.1.2 DDC Interface

DDC forms the heart of Device driver. It models driver state machine and implements the data movement (read/write). The Device DDC is inherently Asynchronous .

The DDA takes buffer from application and passes them down to DDC. Device IOCTL can be performed on an Open’d Controller (hence the hDDC argument in APIs)

The NAND DDC implements a method to create itself wherein it exchanges the interface contract with the DDA above. Once the driver instance is created, using its handle one can initialize the driver thru’ nandInit () passing optional setup parameters. Following this, the device controller and IO Channel are opened via nandOpen(). Once the device has been opened, transactions can be performed using nandReadSync(), nandWriteSync () and nandControl() methods.

Upon completion of use, driver can be gracefully closed and removed from system by following sequence of calls –nandClose (), nandDeInit (), nandDelete ().

The lower device DDC only implements a fully asynchronous data-mover. The job of implementing synch-IO wherein user’s read/write calls block, awaiting completion of transaction is relegated to the DDA.

1.1.1.3 LLC Interface

LLC Services will have usual call sequence of: Init (), HwSetup () etc. LLC maintains its own instance specific data and executes in caller's context and interfaces non-blocking in nature. They do NOT allocate memory dynamically and do NOT use OS services.

It should be noted that LLC never calls DDC functions; it's always the other way round.

1.1.1.4 PALOS Interface

PALOS Abstracts DSP/BIOS services to the Driver. PALOS Functions are state-less on their own and run in context of calling thread. However, when underlying OS function is called, OS may switch context as appropriate, to realize the requested function.

NAND driver uses following service abstractions – SEM, Bi-directional Linked List, PROTECT, Time. The DDC is always the caller & PALOS the callee, the PALOS will never call a DDC function.

The NAND driver release will include an implementation of necessary OS abstraction services for DSP/BIOS environment, if driver is ported to a different platform or integrated with a custom framework, these functions would need to be ported so as to reuse the DDC as-is.

2.2 Dynamic View

2.2.1 The Execution Threads

The device drivers 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.

2.2.2 Driver Creation/Initialization

The sequence below depicts the creation/initialization phase of the DSP/BIOS NAND driver. While at the DDC level, create and init phases of driver instance are clearly demarcated, the same is not the case in DDA and above. Regardless, 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_nandInit(), way up in the application startup phase, perhaps in a central driver initialization function.

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.

The driver also has an interface to format the device: PSP_nandErase (). This function is not reentrant function.

2.2.3 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.

2.2.4 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 ie. application thread of control.

2.2.5 IO Access

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

2.2.6 Driver Close

The application invokes the PSP_nandClose () function to close the instance of the NAND device. Any DMA channels opened are also released via appropriate LLC calls. In addition, the NAND driver's ISRs are unregistered and associated device interrupts disabled to ensure quiescence. This function is not reentrant.