Freescale Semiconductor
Application Note

Document Number: AN4139
Rev. 0, 06/2010

Multi-NAND Disks Implementation Guide

by *Multimedia Applications Division*
Freescale Semiconductor, Inc.
Austin, TX

1 Introduction

Both the Flash Abstraction Layer (FAL) and Flash Media Driver (FMD)-based Flash driver and the Model Device Driver (MDD) and Platform-Dependent Driver (PDD)-based Flash driver, manages Flash ROM as a single disk by default. When the WinCE system is booted up, the file system on the NAND Flash disk is scanned. This scan covers all the partitions on the disk, including the un-mounted partitions. If there are many files on the Flash disk, the time taken to scan the files is too long, and this in turn makes the WinCE boot time longer.

When WinCE boots up, the device.dll file loads the NAND Flash disk driver. Other system services such as gwes.dll, explorer.exe, and servicesStart.exe depend on the device.dll file, and these services start only after the device.dll file completes its tasks. But, the file system scan on the NAND Flash disk, blocks the device.dll file to complete its tasks and due to this, the start of the system services gets delayed, and the booting time also becomes too long.

One of the ways to avoid this problem is to stop loading the NAND Flash disk driver at boot time. After the system is booted up, allow the device.dll file to load the disk driver by

Contents

freescale semiconductor

calling the ActivateDevice function (see Section Appendix A, “Improving the Boot Up Speed on Old NAND Flash Driver,”). After the system boots up, all the threads run in parallel, and the file system scan does not block other services. But, if the user enables the Hive-based registry or the ROM-Only file system, then the NAND Flash disk driver must be loaded at boot time. However, this solution is not feasible.

Based on the above analysis, the multi-NAND Disks solution is imported. The NAND Flash is divided into two disks—when the system boots up, only the small disk (used for Hive-based registry or ROM-Only file system) is loaded with the device.dll file, and the big disk (used for user storage) is loaded after booting up an auto-run application.

This application note describes the ways to implement the multi-NAND Disks solution on a WinCE 6.0 Board Support Package (BSP), and introduces the multi-NAND Disks supported by the Flash driver architecture.

The reference codes given in this application note are based on the Single-Level Cell (SLC) NAND Flash chip.

2 Design References

This section includes the framework description and a porting example.

2.1 Framework Description

The multi-NAND Disks Flash driver is based on the Microsoft Windows CE R2 MDD- and PDD-based Flash driver. To support the R2 MDD- and PDD based-Flash driver, the `SYSGEN_FLASHMDD` must be set to "1", and it includes the following items:

- Block device driver—implements the NAND Flash PDD driver.
- NANDFMD LIB—implements the Flash hardware access code, and this LIB is shared by the OEM Adaption Layer (OAL) and Eboot.
- OAL kernel IOControl code—manages all the Flash access requests between the Flash PDD driver and the Flash hardware access LIB.
- Registry setting—identifies the multi-NAND Disks.
- Auto-run application—used to load the storage NAND disk after the WinCE is booted up.

The earlier version of the NAND Flash driver had a single driver that managed the whole NAND chip, and the blocks used for the XLDR, EBOOT, IPL, BOOT CONFIG, and NK regions were marked as Reserved. Also, this earlier version of the NAND Flash driver do not use these blocks for storage.

Figure 1 shows the layout of the single NAND flash driver.

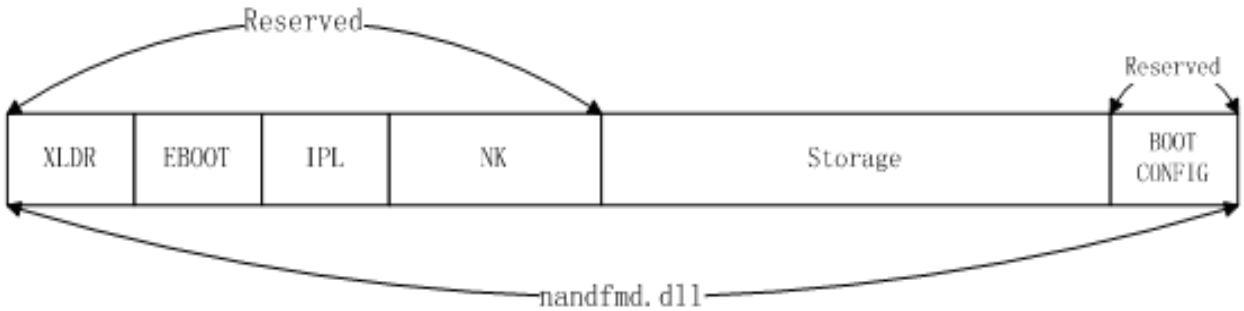

Figure 1. Layout of Single NAND Flash Driver

In the multi-NAND Disks Flash driver, the driver for each disk manages the blocks that are assigned only to it. These drivers do not have access to other NAND Flash blocks. All storage disks use the same drivers—`flashmdd.dll` and `flashpdd_nand.dll`—and these storage disks are identified from each other by the difference in context of the `FmdWrapperPdd` structure.

Figure 2 shows the layout of the multi-NAND Disks Flash driver.

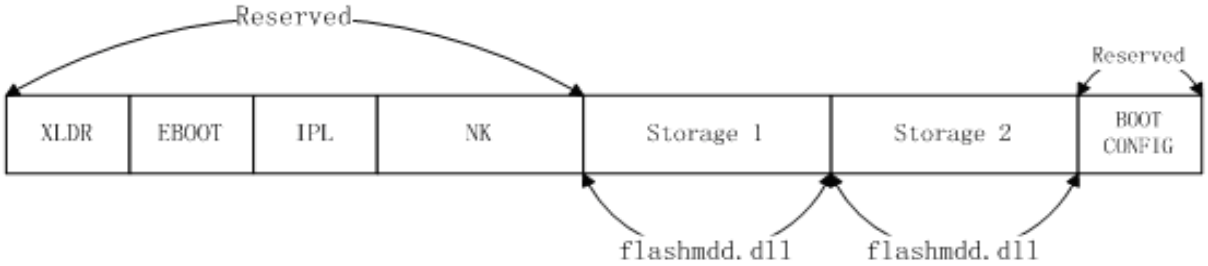

Figure 2. Layout of Multi-NAND Disks Flash Driver

Figure 3 shows the system architecture of the new multi-NAND Disks based Flash driver.

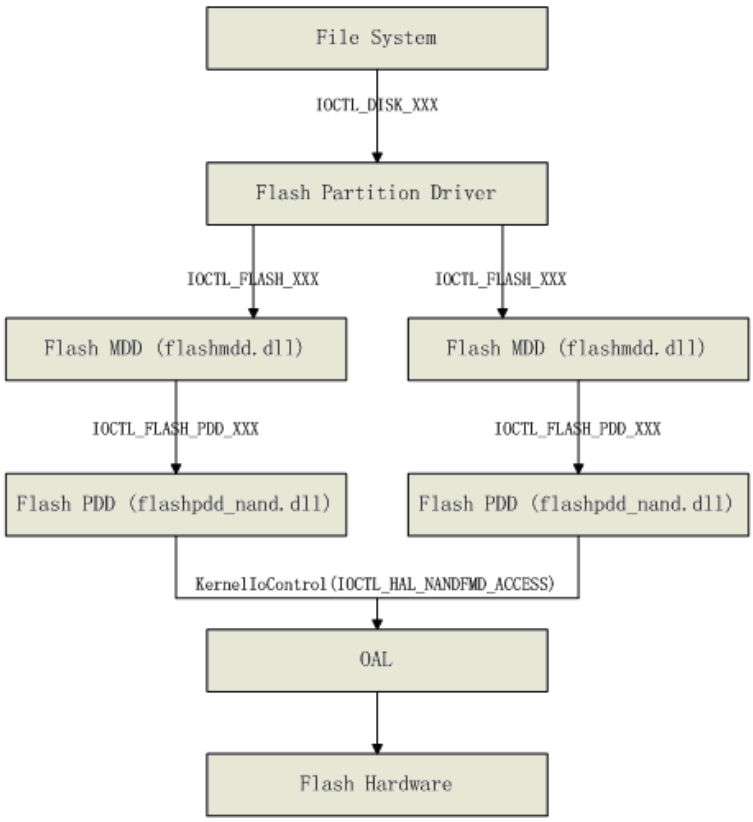

Figure 3. System Architecture of the Multi-NAND Disks Flash Driver

2.2 Porting Example

This application note uses the 2 Kbytes page size SLC NAND Flash (MT29F4G08ABC) and 512 bytes page size SLC NAND Flash as the target Flash ROM, based on the i.MX27ADS WinCE 6.0 F15 BSP. This solution can also be used for other WinCE 6.0 BSP systems with some modifications done to the NAND Flash driver.

The following is the list of all changes that need to be done in the i.MX27ADS F15 WinCE 6.0 BSP to support the multi-NAND Disks, and all these changes should be in the BSP folder:

- `WINCE600\PLATFORM\i.MX27ADS\i.MX27ADS.bat` file:
  - Rename `MX27ADS.bat` to `i.MX27ADS.bat`.
  - Add `set IMGNAND=1`.
  - Add `set SYSGEN_FLASHMDD=1` to support the WinCE 6.0 R2 MDD- and PDD-based Flash driver.
  - Add `set BSP_NAND_PDD=1` to use the Flash PDD driver and include all the related codes.
  - Change `set BSP_NAND_FMD=1` to avoid the earlier version of the NAND FMD driver:

```
set IMGNAND=1
set BSP_NAND_FMD=1
set BSP_NAND_PDD=1
set SYSGEN_FLASHMDD=1
```

- `WINCE600\PLATFORM\i.MX27ADS\sources.cmn` file:
  - Add the macro defined for `BSP_NAND_PDD`. This macro is used in the source codes to support the multi-NAND Disks:

```
!IF "$(BSP_NAND_PDD)"=="1"
CDEFINES=$(CDEFINES) -DBSP_NAND_PDD
!ENDIF
```

- `WINCE600\PLATFORM\i.MX27ADS\FILES\Platform.bib` file:
  - Add the Flash PDD driver `flashpdd_nand.dll` and the auto-run application `InstallNand.exe` in the MODULES section as follows:

```
IF BSP_NAND_PDD
    flashpdd_nand.dll $(_FLATRELEASEDIR)\flashpdd_nand.dll NKSHK
    InstallNand.exe $(_FLATRELEASEDIR)\InstallNand.exe NK SH
ENDIF
```

- `WINCE600\PLATFORM\i.MX27ADS\FILES\Platform.reg` file:
  - Add the registry settings for the multi-NAND Disks support:

```
IF BSP_NAND_PDD
; HIVE BOOT SECTION
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\NAND_Flash]
    "Dll"="flashmdd.dll"
    "FlashPddDll"="flashpdd_nand.dll"
    "Order"=dword:1
    "Prefix"="DSK"
    "Ioctl"=dword:4
    "Profile"="NSFlash"
    "IClass"=multi_sz:"{A4E7EDDA-E575-4252-9D6B-4195D48BB865}"
    "FriendlyName"="NAND FLASH Driver"
    "RegionNumber"=dword:1
IF SYSGEN_FSREGHIVE
    "Flags"=dword:1000
ENDIF
; Override names in default profile
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\NSFlash]
    "PartitionDriver"="flashpart.dll"
    "Name"="NANDFLASH"
    "Folder"="NANDFlash"
    "AutoMount"=dword:1
    "AutoPart"=dword:1
    "AutoFormat"=dword:1
    "MountFlags"=dword:0
    "Ioctl"=dword:4
IF SYSGEN_FSREGHIVE
    "MountAsBootable"=dword:1
    "MountPermanent"=dword:1
;   "MountHidden"=dword:1
ENDIF
IF SYSGEN_FSROMONLY
    "MountAsRoot"=dword:1
ENDIF
[HKEY_LOCAL_MACHINE\Drivers\BlockDevice\NAND_Flash2]
    "Dll"="flashmdd.dll"
    "FlashPddDll"="flashpdd_nand.dll"
    "Order"=dword:1
    "Prefix"="DSK"
```

```
        "Ioctl"=dword:4
        "Profile"="NSFlash2"
        "IClass"=multi_sz:"{A4E7EDDA-E575-4252-9D6B-4195D48BB865}"
        "FriendlyName"="NAND FLASH Driver2"
        "RegionNumber"=dword:2
    ; Override names in default profile
    [HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\NSFlash2]
        "PartitionDriver"="flashpart.dll"
        "Name"="NANDFLASH"
        "Folder"="NANDFlash"
        "AutoMount"=dword:1
        "AutoPart"=dword:1
        "AutoFormat"=dword:1
        "MountFlags"=dword:0
        "Ioctl"=dword:4
    ; END HIVE BOOT SECTION
    [HKEY_LOCAL_MACHINE\init]
        "Launch129"="InstallNand.exe"
        "Depend129"=hex:14,00
    ENDIF
```

- `WINCE600\PLATFORM\i.MX27ADS\src\inc\Ioctl_cfg.h` file:
  - Add the definition for `IOCTL_HAL_NANDFMD_ACCESS`:

```
#ifdef BSP_NAND_PDD
//OEM IOCTL CODE
#define IOCTL_HAL_NANDFMD_ACCESSCTL_CODE(FILE_DEVICE_HAL, 4000, METHOD_BUFFERED,
FILE_ANY_ACCESS)
#endif
```

- `WINCE600\PLATFORM\i.MX27ADS\src\inc\Ioctl_tab.h` file:
  - Add a link between the macro `IOCTL_HAL_NANDFMD_ACCESS` and the function `OALIoCtlHalNandfmdAccess()`:

```
#ifdef BSP_NAND_PDD
{ IOCTL_HAL_NANDFMD_ACCESS, 0, OALIoCtlHalNandfmdAccess },
#endif
```

- `WINCE600\PLATFORM\i.MX27ADS\src\common\nandfmd` folder:
  - Update the `nandfmd_lib` for multi-NAND Disks support, and it can support both the 2 Kbytes page size and 512 bytes page size SLC NAND Flash. Refer to `NANDFMD.zip` file located in the *`AN4139SW.zip`* file for more details.
- `WINCE600\PLATFORM\i.MX27ADS\src\drivers\block\nandpdd` folder:
  - Add this folder to support the new PDD Flash driver. Refer to `NANDPDD.zip` file located in the *`AN4139SW.zip`* file for more details.
- `WINCE600\PLATFORM\i.MX27ADS\src\drivers\block\dir` file:
  - Add `NANDPDD` to `DIRS`.
- `WINCE600\PLATFORM\i.MX27\SRC\OAL\OALLIB\nfc.cpp` file:
  - Add this new file to support the `NANDFMD_LIB` in OAL. Refer to `nfc.cpp` file located in the *`AN4139SW.zip`* file for more details.
- `WINCE600\PLATFORM\i.MX27\SRC\OAL\OALLIB\sources` file:
  - Add `nfc.cpp` to `SOURCES` for compilation.

- `WINCE600\PLATFORM\i.MX27\SRC\OAL\OALLIB\ioctl.c` file:
  - Implement the `OALIoCtlHalNandfmdAccess` and `CriticalSection` functions for the NAND Flash operation:

```
#ifdef BSP_NAND_PDD
#include <partdrv.h>
#include "..\..\common\nandfmd\nandfmd.h"
#endif
… …
#ifdef BSP_NAND_PDD
CRITICAL_SECTION g_oalNfcMutex;
#endif
… …
BOOL OALIoCtlHalPostInit(
UINT32 code, VOID *pInpBuffer, UINT32 inpSize, VOID *pOutBuffer,
UINT32 outSize, UINT32 *pOutSize)
{
// Note that WinCE 6.00 only allows the use of critical sections whereas
// WinCE 5.00 also allowed the use of named mutexes. Therefore, we must
// now create and use a single critical section instead of a named mutex
// to provide mutual exclusion between the OAL and all PMIC drivers for
// accessing the CSPI bus.
InitializeCriticalSection(&g_oalPmicMutex);
#ifdef BSP_NAND_PDD
InitializeCriticalSection(&g_oalNfcMutex);
#endif
// Set flag to indicate it is okay to call EnterCriticalSection() and
// LeaveCriticalSection() within the OAL.
g_oalPostInit = TRUE;
return(TRUE);
}
… …
#ifdef BSP_NAND_PDD
BOOL OALIoCtlHalNandfmdAccess(
UINT32 code, VOID* pInpBuffer, UINT32 inpSize, VOID* pOutBuffer,
UINT32 outSize, UINT32 *pOutSize)
{
     BOOL bResult;
     EnterCriticalSection(&g_oalNfcMutex);
     bResult = OALFMD_Access(pInpBuffer, inpSize);
     LeaveCriticalSection(&g_oalNfcMutex);
     return bResult;
}
#endif
```

- `WINCE600\PLATFORM\i.MX27\SRC\OAL\OALEXE\sources` file:
  - Add support for using `nandfmd_lib.lib` in OAL:

```
!IF "$(BSP_NAND_PDD)" == "1"
TARGETLIBS=\
     $(TARGETLIBS) \
     $(_TARGETPLATROOT)\lib\$(_CPUINDPATH)\nandfmd_lib.lib
!ENDIF
```

- `WINCE600\PLATFORM\i.MX27ADS\SRC\TOOLS\InstallNand` folder:

— This is the auto-run application that is used to load the Flash storage disk driver after the WinCE is booted up. Refer to `Tools.zip` file located in the *`AN4139SW.zip`* file for more details.

- `WINCE600\PLATFORM\i.MX27ADS\SRC\DIRS` file:
  — Add the *Tools* folder to compile.
- `WINCE600\PLATFORM\i.MX27ADS\SRC\BOOTLOADER\XLDR\NAND` folder:
  — This folder is updated to support `MT29F4G08AB` and other 5-cycle address mode NAND Flash. Refer to `nandchip.inc`, `xldr.s` and the `sources` file located in the *`AN4139SW.zip`* file for more details.
- `WINCE600\PLATFORM\i.MX27ADS\SRC\OAL\OALLIB\oal_startup.c` file:
  — Add the following lines of code to support both the 2 Kbytes page size and 512 Kbytes page size NAND Flash. Line 80:

```
#ifdef NAND_LARGE_PAGE
pSYSCTRL->FMCR = 0xFCFFFFE9;
#else
pSYSCTRL->FMCR = 0xFCFFFFC9;
#endif
```

3 NAND Flash PDD Driver Reference

The Flash PDD driver does not access the Flash hardware directly and currently all the Flash hardware access requests are sent to OAL by the `KernelIoControl(IOCTL_HAL_NANDFMD_ACCESS)` function.

The main functions of the PDD driver are as follows:

- Create the `FmdWrapperPdd` context to identify each of the NAND disks.
- Get the `RegionNumber` from the registry, and decide the `m_dwStartBlock` and `m_dwBlockCounts` for each of the NAND disks.
- Wrap all the NAND Flash access requests into a structure `FmdAccessInfo`, and send this to OAL by using the `KernelIoControl(IOCTL_HAL_NANDFMD_ACCESS)` function.

3.1 NAND Flash PDD Driver Source Files

The NAND Flash PDD driver source files are contained in the following locations:

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\flashpdd.def`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\fmdwrappermain.cpp`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\fmdwrappermain.h`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\fmdwrapperpdd.cpp`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\fmdwrapperpdd.h`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\makefile`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\nandfmd.cpp`

`WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDPDD\sources`

3.2 Flashpdd.def

This is the Flash PDD stream driver define file, and it is the same as the Microsoft reference file which is contained in the following location:

`WINCE600\PUBLIC\COMMON\OAK\DRIVERS\BLOCK\MSFLASH\FMDWRAPPERPDD\flashpdd.def`

3.3 Fmdwrappermain.cpp

This file implements the stream interface functions for the Flash PDD driver, and the update from Microsoft reference driver is `FlashPdd_Init()`.

Read the `RegionNumber` from registry and save it to `FmdWrapperPdd` context. This `RegionNumber` is used to identify different sets of NAND disks.

3.4 Fmdwrappermain.h

Based on the Microsoft reference driver file, the definitions for REG_NAND_DRIVER_REGION_NUMBER and SMALL_PARTITION_BLOCKS are given below:

- REG_NAND_DRIVER_REGION_NUMBER—defines the registry key name for `RegionNumber`.
- SMALL_PARTITION_BLOCKS—defines the small disk size. In this example, the small disk is used for Hive-based registry, and the default setting is 100 blocks.

3.5 Fmdwrapperpdd.cpp

Based on the Microsoft reference driver file, replace all the `FMD_xxx` function calls with `OemFMD_xxx`. In this example, `FMDInterface` is re-defined to support the multi-NAND Disks functions.

3.6 Fmdwrapperpdd.h

Based on the Microsoft reference driver file, replace `FMDInterface` with `OemFMDInterface`. The sub-functions in the `OemFMDInterface` function can support more parameters, that are required by the multi-NAND Disks.

The structure `FmdWrapperPdd` has the following additional variable members:

- `m_dwRegionNumber`—is used to identify the disk number.
- `m_dwStartBlock`—records the start NAND block for the disk.
- `m_dwBlockCounts`—records the size of the NAND disk in block.
- `m_bInitialized`—flags for driver initialization.

3.7 Nandfmd.cpp

This file defines the `OEMFMD_xxx` functions for `OemFMDInterface` and it includes the following functions:

```
OEMFMD_Init(),
OEMFMD_Deinit(),
OEMFMD_ReadSector(),
OEMFMD_WriteSector(),
```

```
OEMFMD_EraseBlock(),
OEMFMD_PowerUp(),
OEMFMD_PowerDown(),
OEMFMD_OemIoControl(),
OEMFMD_GetInfo(),
OEMFMD_GetBlockStatus(),
OEMFMD_SetBlockStatus().
```

3.7.1 OEMFMD_Init

`OemFMDInterface > PFN_OemINIT`

This function initializes the NAND Flash hardware with `KernelIoControl (IOCTL_HAL_NANDFMD_ACCESS)`, and `FMD_ACCESS_CODE_HWINIT` is the `dwAccessCode`.

The `OEMFMD_Init` function along with its parameters, is displayed below:

```
PVOID OEMFMD_Init(LPCTSTR lpActiveReg, PPCI_REG_INFO pRegIn, PPCI_REG_INFO pRegOut);
```

3.7.1.1 Parameters

The functions of the three parameters are described below:

- `lpActiveReg`: [in]—pointer to the active registry string which is used to find the device information from the registry. This parameter is set to NULL, if not required.
- `pRegIn`: [in]—pointer to the `PCI_REG_INFO` structure which is used to find the Flash hardware on a PCI hardware. This parameter is set to NULL, if not required.
- `pRegOut`: [in/out]—pointer to the `PCI_REG_INFO` structure which is used to return the flash information. This parameter is set to NULL, if not required.

3.7.1.2 Returns

A handle that can be used in a call to `OEMFMD_Deinit` function. It is the responsibility of the specific FMD implementation to determine what this value represents. A value of zero represents failure.

3.7.2 OEMFMD_Deinit

`OemFMDInterface > PFN_OemDEINIT`

This function de-initializes the flash chip and does nothing else.

The `OEMFMD_Deinit` function along with its parameter, is displayed below:

```
BOOL OEMFMD_Deinit(PVOID pContext);
```

3.7.2.1 Parameter

The function of the parameter is described below:

- `pContext`: [in]—is a handle that is returned from the `OEMFMD_Init` function.

3.7.2.2 Returns

This function returns TRUE on success and FALSE on failure.

3.7.3 OEMFMD_ReadSector

`OemFMDInterface > PFN_OemREADSECTOR`

This function reads the requested sector data and metadata from the Flash media, and calls `KernelIoControl (IOCTL_HAL_NANDFMD_ACCESS)` with `dwAccessCode FMD_ACCESS_CODE_READSECTOR`.

The `OEMFMD_ReadSector` function along with its parameters, is displayed below:

```
BOOL OEMFMD_ReadSector(PVOID pContext, SECTOR_ADDR startSectorAddr, LPBYTE pSectorBuff,
PSectorInfo pSectorInfoBuff, DWORD dwNumSectors, BOOL bWithStartBlock);
```

3.7.3.1 Parameters

The functions of the six parameters are described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.
- `startSectorAddr`: [in]—is the starting physical sector address to read.
- `pSectorBuff`: [out]—pointer to the buffer that contains the sector data to be read from the flash memory. This parameter is set to NULL, if not required.
- `pSectorInfoBuff`: [out]—is a buffer for an array of sector information structures. There is only one sector information entry for every sector that is to be read. This parameter is set to NULL, if not required.
- `dwNumSectors`: [in]—is the number of sectors to read.
- `bWithStartBlock`: [in]—if this value is TRUE, the startSectorAddr must be added with BLOCK_TO_SECTOR (pFlashWrapper > m_dwStartBlock) to form the target sector. All function calls from the top layer must set this flag to TRUE.

3.7.3.2 Returns

This function returns TRUE on success and FALSE on failure.

3.7.4 OEMFMD_WriteSector

`OemFMDInterface > PFN_OemWRITESECTOR`

This function writes the requested sector data and metadata to the flash media, and calls `KernelIoControl (IOCTL_HAL_NANDFMD_ACCESS)` with `dwAccessCode FMD_ACCESS_CODE_WRITESECTOR`.

The `OEMFMD_WriteSector` function along with its parameters, is displayed below:

```
BOOL OEMFMD_WriteSector(PVOID pContext, SECTOR_ADDR startSectorAddr, LPBYTE pSectorBuff,
PSectorInfo pSectorInfoBuff, DWORD dwNumSectors, BOOL bWithStartBlock);
```

3.7.4.1 Parameters

The functions of the six parameters are described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.
- `startSectorAddr`: [in]—is the starting physical sector address to write.
- `pSectorBuff`: [in]—pointer to the buffer that contains the sector data to be written. This parameter is set to NULL, if no data is to be written.
- `pSectorInfoBuff`: [in]—is a buffer for an array of sector information structures. There is only one sector information entry for every sector that is to be written. This parameter is set to NULL, if no data is to be written.
- `dwNumSectors`: [in]—is the number of sectors to write.
- `bWithStartBlock`: [in]—if this value is TRUE, the startSectorAddr must be added with BLOCK_TO_SECTOR (pFlashWrapper > m_dwStartBlock) to form the target sector. All function calls from the top layer must set this flag to TRUE.

3.7.4.2 Returns

This function returns TRUE on success and FALSE on failure.

3.7.5 OEMFMD_EraseBlock

`OemFMDInterface > PFN_OemERASEBLOCK`

This function erases the specified flash block, and calls `KernelIoControl (IOCTL_HAL_NANDFMD_ACCESS)` with `dwAccessCode FMD_ACCESS_CODE_ERASEBLOCK`.

The `OEMFMD_EraseBlock` function along with its parameters, is displayed below:

```
BOOL OEMFMD_EraseBlock(PVOID pContext, BLOCK_ID blockID, BOOL bWithStartBlock);
```

3.7.5.1 Parameters

The functions of the three parameters are described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.
- `blockID`: [in]—is the block number to be erased.
- `bWithStartBlock`: [in]—if this value is TRUE, the SectorAddr must be added with BLOCK_TO_SECTOR (pFlashWrapper > m_dwStartBlock) to form the target sector. All function calls from the top layer must set this flag to TRUE.

3.7.5.2 Returns

This function returns TRUE on success and FALSE on failure.

3.7.6 OEMFMD_PowerUp

`OemFMDInterface > PFN_OemPOWERUP`

This function restores power to the flash memory device if required, but this function is not in use currently. The `OEMFMD_PowerUp` function along with its parameter, is displayed below:

```
VOID OEMFMD_PowerUp(PVOID pContext);
```

3.7.6.1 Parameter

The function of the parameter is described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.

3.7.6.2 Returns

NONE.

3.7.7 OEMFMD_PowerDown

`OemFMDInterface > PFN_OemPOWERDOWN`

This function suspends power to the flash memory device if required, but this function is not in use currently. The `OEMFMD_PowerDown` function `along` with its parameter, is displayed below:

```
VOID OEMFMD_PowerDown(PVOID pContext);
```

3.7.7.1 Parameter

The function of the parameter is described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.

3.7.7.2 Returns

NONE.

3.7.8 OEMFMD_OemIoControl

`OemFMDInterface > PFN_OemIOCONTROL`

This function implements the user-defined commands for the flash memory device, but this function is not in use currently. The `OEMFMD_OemIoControl` function along with its parameters, is displayed below:

```
BOOL OEMFMD_OemIoControl(PVOID pContext, DWORD dwIoControlCode, PBYTE pInBuf, DWORD nInBufSize,
PBYTE pOutBuf, DWORD nOutBufSize, PDWORD pBytesReturned);
```

3.7.8.1 Parameters

The functions of the seven parameters are described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.
- `dwIoControlCode`: [in]—is the control code which specifies the command to execute.
- `pInBuf`: [in]—long pointer to a buffer that contains the data required to perform the operation. This parameter is set to NULL, if the `dwIoControlCode` parameter specifies an operation that does not require any input data.
- `nInBufSize`: [in]—is the size of the buffer pointed to by pInBuf, in bytes.
- `pOutBuf`: [out]—long pointer to a buffer that receives the output data for the operation. This parameter is set to NULL, if the `dwIoControlCode` parameter specifies an operation that does not produce an output data.

- nOutBufSize: [in]—is the size of the buffer pointed to by pOutBuf, in bytes.
- pBytesReturned: [out]—long pointer to a variable that receives the size (in bytes) of the data stored in the buffer pointed to by pOutBuf. The DeviceIoControl function uses the variable pointed to by the pBytesReturned parameter, even when an operation produces no output data and pOutBuf is set to NULL. But, the value of the variable has no meaning after such operations.

3.7.8.2 Returns

This function returns TRUE on success and FALSE on failure.

3.7.9 OEMFMD_GetInfo

```
OemFMDInterface > PFN_OemGETINFO
```

This function determines the size characteristics for the flash memory device. Based on m_dwRegionNumber, this function returns different flash region information for different NAND disks. This function also determines the value of pFlashWrapper > m_dwStartBlock and pFlashWrapper > m_dwBlockCounts for each NAND disk.

The OEMFMD_GetInfo function along with its parameters, is displayed below:

```
BOOL OEMFMD_GetInfo(PVOID pContext, PFlashInfo pFlashInfo);
```

3.7.9.1 Parameters

The functions of the two parameters are described below:

- pContext: [in]—is a handle for FmdWrapperPdd.
- pFlashInfo: [out]—pointer to a structure that contains the size characteristics for the flash memory device.

3.7.9.2 Returns

This function returns TRUE on success and FALSE on failure.

3.7.10 OEMFMD_GetBlockStatus

```
OemFMDInterface > PFN_OemGETBLOCKSTATUS
```

This function returns the status of a block and calls OEMFMD_ReadSector() function to get the block status.

The OEMFMD_GetBlockStatus function along with its parameters, is displayed below:

```
DWORD OEMFMD_GetBlockStatus(PVOID pContext, BLOCK_ID blockID, BOOL bWithStartBlock);
```

3.7.10.1 Parameters

The functions of the three parameters are described below:

- pContext: [in]—is a handle for FmdWrapperPdd.
- blockID: [in]—is the block number used to check the status.

- `bWithStartBlock`: [in]—if this value is TRUE, the sector must be added with BLOCK_TO_SECTOR (pFlashWrapper > m_dwStartBlock) to form the target sector. All function calls from the top layer must set this flag to TRUE.

3.7.10.2 Returns

This function return flags to describe the status of the block.

3.7.11 OEMFMD_SetBlockStatus

`OemFMDInterface > PFN_OemSETBLOCKSTATUS`

This function sets the status of a block and calls `OEMFMD_WriteSector()` function to set the block status.

The `OEMFMD_SetBlockStatus` function along with its parameters, is displayed below:

```
BOOL OEMFMD_SetBlockStatus(PVOID pContext, BLOCK_ID blockID, DWORD dwStatus, BOOL
bWithStartBlock);
```

3.7.11.1 Parameters

The functions of the four parameters are described below:

- `pContext`: [in]—is a handle for `FmdWrapperPdd`.
- `blockID`: [in]—is the block number used to set the status.
- `dwStatus`: [in]—is the status value that is to be set.
- `bWithStartBlock`: [in]—if this value is TRUE, the sector must be added with BLOCK_TO_SECTOR (pFlashWrapper > m_dwStartBlock) to form the target sector. All function calls from the top layer must set this flag to TRUE.

3.7.11.2 Returns

This function returns TRUE on success and FALSE on failure.

4 NANDFMD LIB Reference

The nandfmd_lib.lib file includes the low-level code for NAND Flash operations, and this lib is shared by Eboot and OAL.

4.1 NANDFMD LIB Source Files

The NANDFMD LIB source files are contained in the following locations:

`WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\makefile`

`WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\nandfmd.cpp`

`WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\nandfmd.h`

`WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\k9f1g08u0a.h`

`WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\k9f2g08r0a.h`

```
WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\k9f4g08u0m.h
WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\k9k1g08u0b.h
WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\k9k2g08u0a.h
WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\MT29F4G08ABC.h
WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\nfc.s
WINCE600\PLATFORM\iMX27ADS\SRC\COMMON\NANDFMD\sources
```

4.2 Nandfmd.h

This file defines some NAND Flash common macros such as command, operation macro, NAND Flash type file, FMD access code, and the `FmdAccessInfo` structure. The supported NAND Flash types are k9f1g08u0a, k9f2g08r0a, k9f4g08u0m, k9k1g08u0b, k9k2g08u0a, and MT29F4G08ABC.

The structure `FmdAccessInfo` has the following variable members:

- `dwAccessCode`—is the NAND Flash access code and it currently supports HWINIT, READSECTOR, WRITESECTOR, and ERASEBLOCK.
- `dwStartSector`—defines the start sector for read, write, and erase. While erasing, the start sector must be converted from the BLOCK ID.
- `dwSectorNum`—is used only for read/write, and defines the number of sectors to read/write.
- `pMData`—is a pointer to the main data buffer for read/write operation.
- `pSData`—is a pointer to the spare data buffer for read/write operation.

4.3 Nandfmd.cpp

This file is updated to support both the 2 Kbytes page size and 512 bytes page size SLC NAND Flash for nandfmd_lib. It also defines all the hardware operation functions for the NAND Flash.

4.4 Nfc.s

This file is updated to support both the 2 Kbytes page size and 512 bytes page size SLC NAND Flash for nandfmd_lib. It also defines the ASM code for NFC buffer read and write operation.

5 OAL KernelIoControl Code Reference

To implement the multi-NAND Disks function, the two disk drivers must not be accessed at the same time, in order to access the NFC hardware. Therefore, all the NFC and NAND Flash hardware-related codes are moved to `KernelIoControl()`. The `EnterCriticalSection()` and `LeaveCriticalSection()` functions can be used to avoid multiple access at the same time.

5.1 Kernel Sources Files

The Kernel source files are contained in the following locations:

```
WINCE600\PLATFORM\iMX27ADS\SRC\INC\ioctl_cfg.h
```

```
WINCE600\PLATFORM\iMX27ADS\SRC\INC\ioctl_tab.h

WINCE600\PLATFORM\iMX27ADS\SRC\OAL\OALLIB\ioctl.c

WINCE600\PLATFORM\iMX27ADS\SRC\OAL\OALLIB\nfc.cpp

WINCE600\PLATFORM\iMX27ADS\SRC\OAL\OALLIB\sources

WINCE600\PLATFORM\iMX27ADS\SRC\OAL\OALEXE\sources
```

5.2 Ioctl_cfg.h

A definition is added for IOCTL_HAL_NANDFMD_ACCESS. The following code is the main interface between the NAND Flash PDD driver and the OAL:

```
#ifdef BSP_NAND_PDD
//OEM IOCTL CODE
#define IOCTL_HAL_NANDFMD_ACCESSCTL_CODE(FILE_DEVICE_HAL, 4000, METHOD_BUFFERED,
FILE_ANY_ACCESS)
#endif
```

5.3 Ioctl_tab.h

A link is added for the `IOCTL_HAL_NANDFMD_ACCESS` function.

```
#ifdef BSP_NAND_PDD
{ IOCTL_HAL_NANDFMD_ACCESS, 0, OALIoCtlHalNandfmdAccess },
#endif
```

5.4 Ioctl.c

A definition is added for the `CRITICAL_SECTION g_oalNfcMutex` variable, and this global variable is initialized in the `OALIoCtlHalPostInit()` function. It is also used to avoid multiple access of the NAND Flash driver at the same time.

The definition for the `OALIoCtlHalNandfmdAccess()` function is also added, and this function transfers all the top layer `KernelIoControl` functions to OAL NAND Flash operation. The nandfmd_lib.lib file uses the `OALFMD_Access()` function.

```
#ifdef BSP_NAND_PDD
#include <partdrv.h>
#include "..\..\common\nandfmd\nandfmd.h"
#endif
… …
#ifdef BSP_NAND_PDD
CRITICAL_SECTION g_oalNfcMutex;
#endif
… …
BOOL OALIoCtlHalPostInit(
UINT32 code, VOID *pInpBuffer, UINT32 inpSize, VOID *pOutBuffer,
UINT32 outSize, UINT32 *pOutSize)
{
// Note that WinCE 6.00 only allows the use of critical sections whereas
// WinCE 5.00 also allowed the use of named mutexes. Therefore, we must
// now create and use a single critical section instead of a named mutex
// to provide mutual exclusion between the OAL and all PMIC drivers for
```

```
// accessing the CSPI bus.
InitializeCriticalSection(&g_oalPmicMutex);
#ifdef BSP_NAND_PDD
InitializeCriticalSection(&g_oalNfcMutex);
#endif
// Set flag to indicate it is okay to call EnterCriticalSection() and
// LeaveCriticalSection() within the OAL.
g_oalPostInit = TRUE;
return(TRUE);
}
… …
#ifdef BSP_NAND_PDD
BOOL OALIoCtlHalNandfmdAccess(
UINT32 code, VOID* pInpBuffer, UINT32 inpSize, VOID* pOutBuffer,
UINT32 outSize, UINT32 *pOutSize)
{
        BOOL bResult;
        EnterCriticalSection(&g_oalNfcMutex);
        bResult = OALFMD_Access(pInpBuffer, inpSize);
        LeaveCriticalSection(&g_oalNfcMutex);
        return bResult;
}
#endif
```

5.5 Nfc.cpp

This file defines the `NFCAlloc()`, `NFCWait()`, and `NFCSetClock()` functions for using the nandfmd_lib.lib file. It also defines the `OALFMD_Access()` function, which converts the `OALIoCtlHalNandfmdAccess()` function to `FMD_xxx()` operation function.

5.6 Sources (OALLIB)

The following lines of code are to be added to compile nfc.cpp into OAL, when using the multi-NAND Disks support:

```
!IF "$(BSP_NAND_PDD)" == "1"
SOURCES=$(SOURCES)\
        nfc.cpp
!ENDIF
```

5.7 Sources (OALEXE)

The following lines of code are to be added to link nandfmd_lib.lib into OAL.exe, when using the multi-NAND Disks support:

```
!IF "$(BSP_NAND_PDD)" == "1"
TARGETLIBS=\
        $(TARGETLIBS) \
        $(_TARGETPLATROOT)\lib\$(_CPUINDPATH)\nandfmd_lib.lib
!ENDIF
```

6 Registry and BIB Settings

This section shows the settings to be done in the platform.reg and platform.bib files.

6.1 Platform.reg

When WinCE is booted up, only the RegionNumber=1 disk is loaded for the Hive-based registry, and the storage disk RegionNumber=2 is not loaded:

```
; HIVE BOOT SECTION
IF SYSGEN_FSREGHIVE
[HKEY_LOCAL_MACHINE\init\BootVars]
         "SYSTEMHIVE"="\\NANDFlash\\system.hv"
         "PROFILEDIR"="\\NANDFlash\\usr.hv"
         "Start DevMgr"=dword:1
         "DefaultUser"="default"
;        "RegistryFlags"=dword:1
ENDIF
IF BSP_NAND_PDD
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\NAND_Flash]
"Dll"="flashmdd.dll"
"FlashPddDll"="flashpdd_nand.dll"
"Order"=dword:0
"Prefix"="DSK"
"Ioctl"=dword:4
"Profile"="NSFlash"
"IClass"=multi_sz:"{A4E7EDDA-E575-4252-9D6B-4195D48BB865}"
"FriendlyName"="NAND FLASH Driver"
"RegionNumber"=dword:1
IF SYSGEN_FSREGHIVE
"Flags"=dword:1000
ENDIF
; Override names in default profile
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\NSFlash]
"PartitionDriver"="flashpart.dll"
"Name"="NANDFLASH"
"Folder"="NANDFlash"
"AutoMount"=dword:1
"AutoPart"=dword:1
"AutoFormat"=dword:1
         "MountFlags"=dword:0
         "Ioctl"=dword:4
IF SYSGEN_FSREGHIVE
         "MountAsBootable"=dword:1
         "MountPermanent"=dword:1
;        "MountHidden"=dword:1
ENDIF
IF SYSGEN_FSROMONLY
         "MountAsRoot"=dword:1
ENDIF
IF SYSGEN_FSREGHIVE
[HKEY_LOCAL_MACHINE\System\StorageManager\AutoLoad\NSFlash]
"DriverPath"="Drivers\\BuiltIn\\NAND_Flash"
         "LoadFlags"=dword:1
"BootPhase"=dword:0
"Order"=dword:0
```

```
ENDIF
[HKEY_LOCAL_MACHINE\Drivers\BlockDevice\NAND_Flash2]
"Dll"="flashmdd.dll"
"FlashPddDll"="flashpdd_nand.dll"
"Order"=dword:1
"Prefix"="DSK"
"Ioctl"=dword:4
"Profile"="NSFlash2"
"IClass"=multi_sz:"{A4E7EDDA-E575-4252-9D6B-4195D48BB865}"
"FriendlyName"="NAND FLASH Driver2"
"RegionNumber"=dword:2
; Override names in default profile
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\NSFlash2]
"PartitionDriver"="flashpart.dll"
"Name"="NANDFLASH"
"Folder"="NANDFlash"
"AutoMount"=dword:1
"AutoPart"=dword:1
"AutoFormat"=dword:1
        "MountFlags"=dword:0
        "Ioctl"=dword:4
ENDIF
; END HIVE BOOT SECTION
```

6.2 Platform.bib

In platform.bib, the PDD NAND Flash driver must be added in the MODULES section.

```
flashpdd_nand.dll $(_FLATRELEASEDIR)\flashpdd_nand.dll NK SHK
```

7 NAND Storage Disk Auto Load Application

This application is added to the `WINCE600\PLATFORM\iMX27ADS\SRC\TOOLS \InstallNand` folder, and it calls the `ActivateDevice()` function to load the storage disk driver.

In platform.bib, the following line must be added to the MODULES section to include this application in the NK image:

```
InstallNand.exe $(_FLATRELEASEDIR)\InstallNand.exe NK SH
```

In platform.reg, the following lines of code must be added to make the application auto-run, after WinCE boots up:

```
[HKEY_LOCAL_MACHINE\init]
        "Launch129"="InstallNand.exe"
        "Depend129"=hex:14,00
```

8 Patch for i.MX27ADS WinCE 6.0 F15 BSP

Unzip the patch file and cover to old BSP folder, sysgen, and built. This BSP is tested on 512 bytes page size SLC NAND Flash K9K1G08U0B. Refer to `i.MX27ADS_Multi-NANDDisk.zip` file located in the *`AN4139SW.zip`* file for more details.

9 Patch for i.MX313DS WinCE 6.0 SDK 1.4 BSP

Unzip the patch file and cover to old BSP folder, sysgen, and built. This BSP is tested on 2 Kbytes page size SLC NAND Flash K9F2G08R0A. Refer to `i.MX313DS_Multi-NANDDisk.zip` file located in the *`AN4139SW.zip`* file for more details.

10 Revision History

Table 1 provides the revision history for this application note.

Table 1. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	06/2010	Initial release

Appendix A Improving the Boot Up Speed on Old NAND Flash Driver

As mentioned in Section 1, “Introduction,” there is a simple way to avoid the long time boot up issue, which is not available on the Hive-based registry and ROM-Only file system device. This method can also be used on WinCE 5.0.

NOTE

The multi-NAND Disks solution can work only on WinCE 6.0 systems.

When WinCE is booted up, do not load the NAND Flash disk driver. Change the registry setting for nandfmd.dll, and move it from [HKEY_LOCAL_MACHINE\Drivers\ BuiltIn] to [HKEY_LOCAL_MACHINE\Drivers\BlockDevice] location, as follows:

```
;[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\NAND_Flash]
[HKEY_LOCAL_MACHINE\Drivers\BlockDevice\NAND_Flash]
"Dll"="nandfmd.dll"
"Order"=dword:0
"Prefix"="DSK"
"Ioctl"=dword:4
"Profile"="FlashDisk"
"IClass"=multi_sz:"{A4E7EDDA-E575-4252-9D6B-4195D48BB865}"
"FriendlyName"="NAND FLASH Driver"
```

Use the Tools\InstallNand folder to load the NAND Flash driver after the system boots up, and the sample code is as follows:

```
int WINAPI WinMain(
HINSTANCE hinst,
HINSTANCE hinstPrev,
LPWSTR szCmdLine,
int iCmdShow
)
{
        HANDLE hDevHandle;
        TCHAR pszDriverPath[] = TEXT("\\Drivers\\BlockDevice\\NAND_Flash");
        UNREFERENCED_PARAMETER(hinst);
        UNREFERENCED_PARAMETER(hinstPrev);
        UNREFERENCED_PARAMETER(szCmdLine);
        UNREFERENCED_PARAMETER(iCmdShow);
        Sleep(5000);
        hDevHandle = ActivateDevice(pszDriverPath, 0);
        if(hDevHandle == NULL)
        {
                RETAILMSG(1, (TEXT("ActivateDevice failed, error = %d.\r\n"),
GetLastError()));
        }
        else
        {
                RETAILMSG(1, (TEXT("ActivateDevice success.\r\n")));
        }
        return 0;
}
```

The following lines of code are added to the platform.reg file to make InstallNand.exe auto-run, after the system is booted up:

```
[HKEY_LOCAL_MACHINE\init]
         "Launch129"="InstallNand.exe"
         "Depend129"=hex:14,00
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN4139
Rev. 0
06/2010