Freescale Semiconductor
Application Note

Document Number: AN4137
Rev. 0, 06/2010

BinFS Implementation Guide

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

Contents

1 Introduction

This section gives an introduction to the application note.

1.1 Purpose

In a traditional file system, the Windows CE™ (WinCE) image is generally an NK.NB0/NK.BIN signal file. Therefore, when a NAND Flash is used for storage, the NK.NB0 signal file has to be copied into the RAM (by the EBOOT) before the file is run, as the NAND Flash storage cannot support XIP (Execute In Place).

There are two disadvantages in this process: long boot time and requirement of large RAM memory size. If the WinCE image is large (included with more features), these issues become critical. The Binary ROM Image File System (BinFS) can fix the two issues by using a 32-Mbyte RAM to run a 64-Mbyte WinCE image. In a BinFS file system, the final WinCE image is divided into multi-Binary Image (BIN) files, and only the XIPKERNEL BIN (less than 5 Mbytes) is copied into the RAM by the EBOOT. The files in the other BIN files work with demand paging mode. These files are loaded into the RAM only when they are required to be run.

With the BinFS support, the boot time (from start reading Flash image to show WinCE desktop) is reduced to 6

freescale™ semiconductor

seconds in the i.MX27ADS board, and the free RAM size increases, which can be used for storing and programming. This reduces the cost of the final products.

1.2 Scope

This application note gives an introduction to the BinFS support in the Freescale WinCE 6.0 Board Support Package (BSP) based on the NAND Flash.

NOTE

The reference codes in this application note are based on the Freescale i.MX27ADS F15 BSP.

1.3 Audience Description

This application note is helpful to the people who wants to know about:

- NAND Flash boot time
- RAM that uses NK images

2 Design References

This section gives an introduction to the system architecture of the BinFS files and also gives an example for porting.

2.1 Framework Description

In this application note, the BinFS is based on the multi-NAND disks solution (refer to the *Multi-Nand Disks Implementation Guide* (AN4139)). A read-only disk (NAND disk) is created to manage the BinFS partition.

Some important aspects related to the BinFS, which are described in the following sections, are as follows:

- EBOOT—supports multi-BIN WinCE image and BinFS
- NAND disk—blocks device driver to manage the BinFS partition
- Registry setting and bib files—supports BinFS
- Support scripts—processes the ce.bib file that supports BinFS

In a non-BinFS file system, the WinCE image is the RAMIMAGE that follows the NAND Flash and RAM layout as shown in Figure 1 and Figure 2, respectively.

Figure 1. NAND Flash Layout for non-BinFS File System

Figure 2. RAM Layout for non-BinFS File System

In NAND Flash and RAM layout, when the EBOOT loads the NK.NB0 file into the RAM, the RAM reads the 48-Mbyte fixed size data from the NAND Flash. The read data is then copied into the WinCE NK run RAM base address, IMAGE_BOOT_NKIMAGE_RAM_PA_START (defined in image_cfg.h), based on the i.MX27ADS F15 WinCE 6.0 BSP.

In a BinFS file system, only the XIPKERNEL.NB0 and CHAIN.NB0 files are RAMIMAGE, and the others are NANDIMAGE. The total size of the RAMIMAGE is less than 5 Mbytes. In the earlier version of WinCE, the CHAIN.NB0 file is used by the EBOOT to locate each BIN region.

The NAND Flash and RAM layouts followed by the BinFS file system are shown in Figure 3 and Figure 4, respectively.

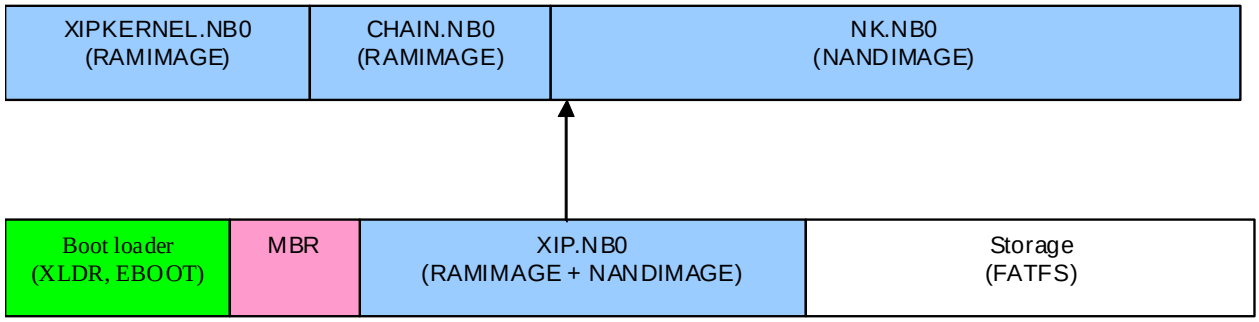

Figure 3. NAND Flash Layout for BinFS File System

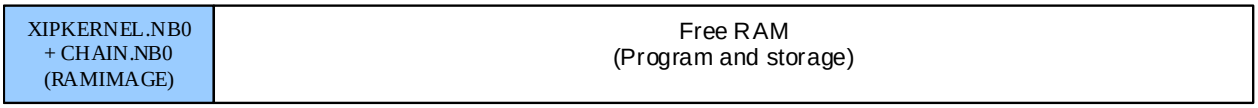

Figure 4. RAM Layout for BinFS File System

In this new layout, when the EBOOT loads a WinCE image, it first checks the Memory Buffer Register (MBR) that includes the RAMIMAGE (partition type 0x23, PART_RAMIMAGE) and NANDIMAGE partitions (partition type 0x21, PART_BINFS). Only the RAMIMAGE partition is copied into the RAM. The EBOOT also gets the logical start sector address and sector size of the RAMIMAGE partition, and the real size of the data that is required to be copied into the WinCE NK run RAM base address, IMAGE_BOOT_NKIMAGE_RAM_PA_START, can be calculated.

NOTE

Both the MBR and XIP.NB0 regions are managed by the NAND Flash block device driver, Nanddisk.dll.

The XIPKERNEL.NB0 contains only the modules necessary for the boot-up. The files, which are loaded prior to implementation of BinFS, are stored in the XIPKERNEL.NB0. As the kernel resides in the RAM and the XIPKERNEL region is the RAMIMAGE, the files loaded into the XIPKERNEL region typically include everything required for the kernel. The FILES\MakeBinfsBib.js script is used to decide the modules that are required to be added into the XIPKERNEL region. The BSP drivers, which are required to be put into the XIPKERNEL region, should be directly specified in platform.bib file.

Microsoft® suggests the following modules to be put into the XIPKERNEL region:

- NK.exe

- Kernel.dll
- Coredll.dll
- K.coredll.dll
- Oalioctl.dll
- Filesystem.dll
- Fsdmgr.dll
- Mspart.dll
- Romfsd.dll
- Binfs.dll
- Default.fdf or boot.hv
- Fpcrt.dll
- Ceddk.dll (if required by the Flash driver)
- Flash driver

If the Flash driver is loaded by the device manager, add device.dll, devmgr.dll, regenum.dll, busenum.dll, and pm.dll files into the XIPKERNEL region. For Kernel Independent Transport Layer (KITL) support, add kitl.dll file into the XIPKERNEL region. For debugging support, add hd.dll, osaxst0.dll, and osaxst1.dll files into the XIPKERNEL region.

2.2 Porting Example

In this application note, based on the i.MX27ADS WinCE 6.0 F15 BSP, the 512-byte page size Single Level Cell (SLC) NAND Flash is used as the target Flash ROM. This solution can also be used in other WinCE 6.0 BSPs with the related NAND Flash driver. In this example, three BINs—XIPKERNEL.BIN, CHAIN.BIN, and NK.BIN—are created for the BinFS support.

In non-BinFS file system, the final WinCE images are NK.BIN and NK.NB0. But in BinFS file system, the final images are XIP.BIN and XIP.NB0. These two files are emerged from the files—XIPKERNEL.BIN,

CHAIN.BIN, and NK.BIN. There is an another file, chain.lst, that is selected by the platform builder to download the BinFS image. The property page in a BinFS file system is shown in Figure 5.

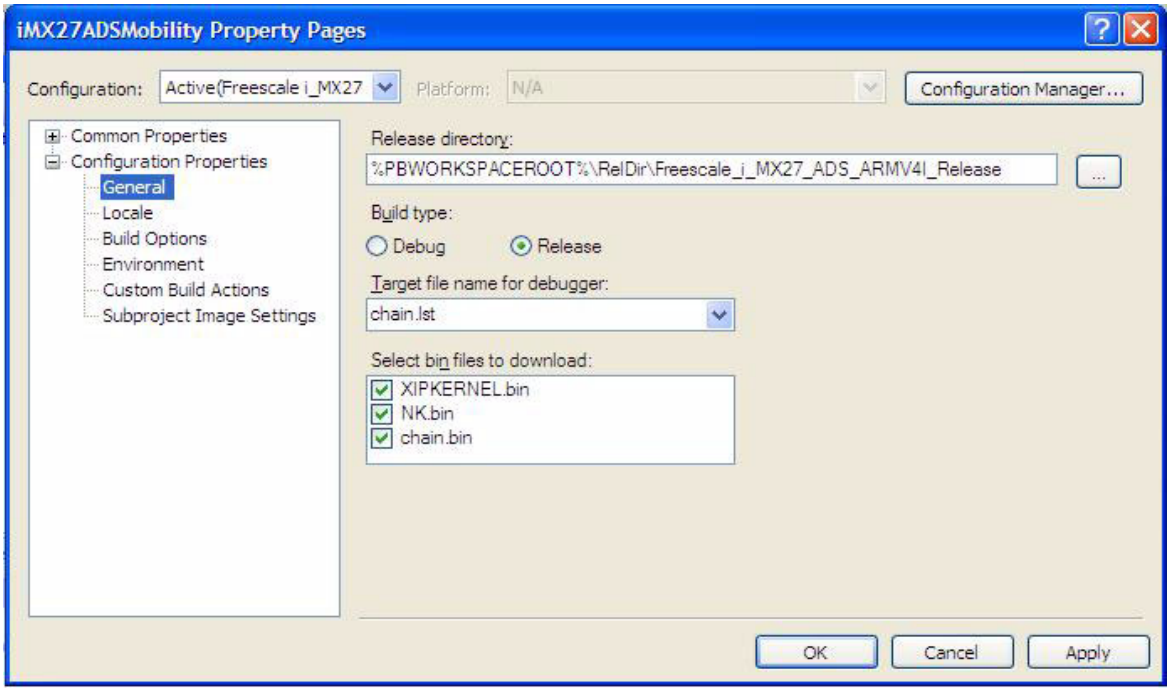

Figure 5. i.MX27ADSMobility Property Page

All the changes that are required to be done in the i.MX27ADS F15 WinCE 6.0 BSP to support the BinFS are listed as follows:

NOTE

All the changes listed are made in the BSP folder.

- File WINCE600\PLATFORM\iMX27ADS\iMX27ADS.bat:
  - Rename the file, MX27ADS.bat to iMX27ADS.bat.
  - Add set IMGNAND = 1.
  - Add set SYSGEN_FLASHMDD = 1 to support the WinCE 6.0 R2 Model Device Driver (MDD) and Platform-Dependent Driver (PDD) based Flash driver.
  - Add set BSP_NAND_PDD = 1 to use the Flash PDD driver, and include the related support codes.
  - Change set BSP_NAND_FMD = to avoid the usage of old NAND Flash Media Driver (FMD) driver.
  - Add set BSP_SUPPORT_DEMAND_PAGING = 1 for BinFS support.

    These changes are coded as follows:

```
set IMGNAND=1
set BSP_NAND_FMD=
set BSP_NAND_PDD=1
set SYSGEN_FLASHMDD=1
set BSP_SUPPORT_DEMAND_PAGING=1
if "%BSP_SUPPORT_DEMAND_PAGING%"=="1" set BSP_NONANDDISK=
if "%BSP_SUPPORT_DEMAND_PAGING%"=="1" set SYSGEN_BINFS=1
```

- File WINCE600\PLATFORM\iMX27ADS\sources.cmn
  Add macro definition for the BSP_NAND_PDD file. 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
```

- File WINCE600\PLATFORM\iMX27ADS\FILES\Config.bib
  Change memory map to support the BinFS.

NOTE

The config.bib file can be found in the zip file, AN4137SW.zip, that is included with the application note.

- File WINCE600\PLATFORM\iMX27ADS\FILES\MakeBinfsBib.js
  Java script file is used to process the ce.bib file for multi-BIN support.

NOTE

The MakeBinfsBib.js file can be found in the zip file, AN4137SW.zip, that is included with the application note.

- File WINCE600\PLATFORM\iMX27ADS\FILES\Platform.bib:
  - Use macro to define the different BIN names.
  - Add block driver for the BinFS support.
  - Add the Flash PDD driver, flashpdd_nand.dll, and the auto-run application, InstallNand.exe, into the MODULES section in the platform.bib file.

NOTE

The platform.bib file can be found in the zip file, AN4137SW.zip, that is included with the application note.

- File WINCE600\PLATFORM\iMX27ADS\FILES\Platform.reg:
  - Add registry setting for the multi-NAND disks support.
  - Add registry setting for the BinFS 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

IF BSP_SUPPORT_DEMAND_PAGING
IF BSP_NONANDDISK !
; HIVE BOOT SECTION
[HKEY_LOCAL_MACHINE\System\StorageManager\AutoLoad\NSDisk]
     "DriverPath"="Drivers\\BlockDevice\\NandDisk"
     "LoadFlags"=dword:1
     "MountFlags"=dword:11
     "BootPhase"=dword:0
```

```
IF SYSGEN_FSREGHIVE
     "Flags"=dword:1000
ENDIF

[HKEY_LOCAL_MACHINE\Drivers\BlockDevice\NandDisk]
     "Prefix"="DSK"
     "Dll"="NandDisk.dll"
     "Order"=dword:0
     "Ioctl"=dword:4
     "Profile"="NandDisk"
     "FriendlyName"="BINFS Flash Driver"
     "MountFlags"=dword:11
     "BootPhase"=dword:0

IF SYSGEN_FSREGHIVE
     "Flags"=dword:1000
ENDIF

; Bind BINFS to the block driver
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\NandDisk]
     "DefaultFileSystem"="BINFS"
     "PartitionDriver"="mspart.dll"
     "AutoMount"=dword:1
     "AutoPart"=dword:1
     "MountFlags"=dword:0
     "Folder"="BINFS"
     "Name"="BINFS Flash Disk"
     "BootPhase"=dword:0
     "MountAsROM"=dword:1
     "MountHidden"=dword:1
     "MountPermanent"=dword:1

IF SYSGEN_FSREGHIVE
     "Flags"=dword:1000
ENDIF
; END HIVE BOOT SECTION
ENDIF
ENDIF  ;BSP_SUPPORT_DEMAND_PAGING
```

- File `WINCE600\PLATFORM\iMX27ADS\FILES\PreRomImage.bat`
  This file is called before the platform builder, romimage stage. It processes the `ce.bib` file for the multi-BIN support.

NOTE

The `PreRomImage.bat` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- File `WINCE600\PLATFORM\iMX27ADS\src\inc\image_cfg.h`
  Add NAND address definition for the MBR that is required for the BinFS:

```
#define IMAGE_BOOT_MBR_NAND_OFFSET              (IMAGE_BOOT_IPLIMAGE_NAND_OFFSET +
IMAGE_BOOT_IPLIMAGE_NAND_SIZE)
#define IMAGE_BOOT_MBR_NAND_PA_START           (IMAGE_BOOT_NANDDEV_NAND_PA_START +
IMAGE_BOOT_MBR_NAND_OFFSET)
#define IMAGE_BOOT_MBR_NAND_UA_START
((DWORD)OALPAtoUA(IMAGE_BOOT_MBR_NAND_PA_START))
```

```
#define IMAGE_BOOT_MBR_NAND_CA_START
((DWORD)OALPAtoCA(IMAGE_BOOT_MBR_NAND_PA_START))
#define IMAGE_BOOT_MBR_NAND_SIZE                (128 * 1024)
#define IMAGE_BOOT_MBR_NAND_PA_END              (IMAGE_BOOT_MBR_NAND_PA_START +
IMAGE_BOOT_MBR_NAND_SIZE - 1)

#define IMAGE_BOOT_NKIMAGE_NAND_OFFSET          (IMAGE_BOOT_MBR_NAND_OFFSET +
IMAGE_BOOT_MBR_NAND_SIZE)
```

- File `WINCE600\PLATFORM\iMX27ADS\src\inc\image_cfg.inc`
  Add NAND address definition for the MBR that is required for the BinFS:

```
IMAGE_BOOT_MBR_NAND_OFFSET          EQU (IMAGE_BOOT_IPLIMAGE_NAND_OFFSET +
IMAGE_BOOT_IPLIMAGE_NAND_SIZE)
IMAGE_BOOT_MBR_NAND_PA_START        EQU (IMAGE_BOOT_NANDDEV_NAND_PA_START +
IMAGE_BOOT_MBR_NAND_OFFSET)
IMAGE_BOOT_MBR_NAND_SIZE            EQU (128 * 1024)
IMAGE_BOOT_MBR_NAND_PA_END          EQU (IMAGE_BOOT_MBR_NAND_PA_START +
IMAGE_BOOT_MBR_NAND_SIZE - 1)

IMAGE_BOOT_NKIMAGE_NAND_OFFSET      EQU (IMAGE_BOOT_MBR_NAND_OFFSET +
IMAGE_BOOT_MBR_NAND_SIZE)
```

- File `WINCE600\PLATFORM\iMX27ADS\src\inc\Ioctl_cfg.h`
  Add define 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
```

- File `WINCE600\PLATFORM\iMX27ADS\src\inc\Ioctl_tab.h`
  Add link between the macro, `IOCTL_HAL_NANDFMD_ACCESS`, and the function, `OALIoCtlHalNandfmdAccess()`:

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

- File `WINCE600\PLATFORM\iMX27ADS\src\inc\oemaddrtab_cfg.inc`
  Map more addresses for the NAND Flash Control (NFC) to support multi-BIN images in the EBOOT:

```
   DCD 0x9C200000, CSP_BASE_REG_PA_NANDFC,     48      ; EMI modules (NANDFC + ESDRAMC
+ WEIM + M3IF + PCMCIA)
DCD 0x9F200000, CSP_BASE_MEM_PA_VRAM,        1        ; VRAM (45K)
```

- Folder `WINCE600\PLATFORM\iMX27ADS\src\common\nandfmd`
  Update the `nandfmd_lib` file for multi-NAND disks support. The multi-NAND disks can support both 2-Kbyte and 512-byte page size SLC NAND Flash.

NOTE

The `NANDFMD.zip` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- Folder `WINCE600\PLATFORM\iMX27ADS\src\drivers\block\nandpdd`
  Add this folder to support the new PDD Flash driver.

NOTE

The `NANDPDD.zip` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- Folder `WINCE600\PLATFORM\iMX27ADS\src\drivers\block\nanddisk`
  Add this folder to support the BinFS block device driver.

NOTE

The `NandDisk.zip` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- File `WINCE600\PLATFORM\iMX27ADS\src\drivers\block\DIRS`:
  - Add NANDPDD into the DIRS file.
  - Add NANDDISK into the DIRS file.
- File `WINCE600\PLATFORM\iMX27\SRC\OAL\OALLIB\nfc.cpp`
  New file to support the `NANDFMD_LIB` in OEM Adaptation Layer (OAL)

NOTE

The `nfc.cpp` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- File `WINCE600\PLATFORM\iMX27\SRC\OAL\OALLIB\sources`
  Add nfc.cpp file to the sources file for compilation.
- File `WINCE600\PLATFORM\iMX27\SRC\OAL\OALLIB\ioctl.c`
  Implement `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
```

- File `WINCE600\PLATFORM\iMX27\SRC\OAL\OALEXE\sources`
  Add support to use the `nandfmd_lib.lib` file in OAL:

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

- Folder `WINCE600\PLATFORM\iMX27ADS\SRC\TOOLS\InstallNand`
  This is an auto-run application, which is used to load the Flash storage disk driver after the WinCE is booted up.

NOTE

The `Tools.zip` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- File `WINCE600\PLATFORM\iMX27ADS\SRC\DIRS`
  Add the TOOLS folder for compilation.
- Folder `WINCE600\PLATFORM\iMX27ADS\SRC\BOOTLOADER\XLDR\NAND`
  Update this folder to support the MT29F4G08AB and the other 5-cycle address mode NAND Flash.

NOTE

The `nandchip.inc`, `xldr.s`, and `sources` files can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

- Folder WINCE600\PLATFORM\iMX27ADS\SRC\BOOTLOADER\EBoot
  Update this folder to support the multi-BIN images and BinFS in the EBOOT.

NOTE

The flash.c, main.c, and nand.c files can be found in the zip file, AN4137SW.zip, that is included with the application note.

- File WINCE600\PLATFORM\iMX27ADS\SRC\OAL\OALLIB\oal_startup.c
  Add the following code (refer to line 80 in the oal_startup.c file) to support both the 2-Kbyte and 512-Kbyte page size NAND Flash:

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

3 EBOOT Reference

The main functions of EBOOT to support the multi-BIN images and BinFS files are as follows:

- To use a different RAM buffer to receive the downloaded multi-BIN files
- To make the MBR for the WinCE NAND disks driver before programming the image into the NAND Flash
- To check the MBR before loading the WinCE image to the RAM, and to copy only the RAMIMAGE data to IMAGE_BOOT_NKIMAGE_RAM_PA_START

While downloading the image from the PC to the board, the platform builder sends the BIN files one after the other based on the chain.lst file. In old EBOOT, all the BIN files were received with the same RAM buffer, and the EBOOT was unable to receive multi-BIN images. However, the new EBOOT uses a different RAM address to receive the BIN files, based on the different starting address of each BIN file. The EBOOT also records the starting address and size of each BIN, after all the BIN files are downloaded into the board. Based on the recorded starting address and size, EBOOT can program the address and size to the NAND Flash sequentially. In i.MX27ADS F15 BSP, the first BIN is RAMIMAGE and the last is chain. Before programming the BIN data, the EBOOT should create an MBR for the received WinCE image. The created MBR includes two partitions: one for the RAMIMAGE (PART_RAMIMAGE) and the other for the NANDIMAGE (PART_BINFS). The size for each partition is calculated from the received BIN files. Generally, a 128-Kbyte memory is reserved for the MBR region. However, the real data for the MBR is only 512 bytes.

The first three bytes of the MBR are 0xE9, 0xFD, and 0xFF, and the last two bytes of the MBR are 0x55 and 0xAA. At the end of MBR, there are four partition tables structured as follows:

```
typedef struct _PARTENTRY {
BYTE            Part_BootInd;           // If 80h means this is boot partition
BYTE            Part_FirstHead;         // Partition starting head based 0
BYTE            Part_FirstSector;       // Partition starting sector based 1
BYTE            Part_FirstTrack;        // Partition starting track based 0
BYTE            Part_FileSystem;        // Partition type signature field
BYTE            Part_LastHead;          // Partition ending head based 0
BYTE            Part_LastSector;        // Partition ending sector based 1
```

```
BYTE            Part_LastTrack;         // Partition ending track based 0
DWORD       Part_StartSector;       // Logical starting sector based 0
DWORD       Part_TotalSectors;      // Total logical sectors in partition
} PARTENTRY;
```

After each BIN file is programmed to the NAND Flash, the EBOOT reads back the data for verification. In the old EBOOT, the read back data is stored in the RAM that followed the receiver buffer. However, this cannot be applied in multi-BIN images as another BIN data can occupy the address. Therefore, in the new EBOOT, checksum is used to verify the BIN data.

3.1 Updated Source Files for EBOOT

The updated source files for the EBOOT are as follows:

- `WINCE600\PLATFORM\iMX27ADS\SRC\Bootloader\Eboot\flash.c`
- `WINCE600\PLATFORM\iMX27ADS\SRC\Bootloader\Eboot\main.c`
- `WINCE600\PLATFORM\iMX27ADS\SRC\Bootloader\Eboot\nand.c`
- `WINCE600\PLATFORM\iMX27ADS\SRC\inc\image_cfg.h`
- `WINCE600\PLATFORM\iMX27ADS\SRC\inc\image_cfg.inc`
- `WINCE600\PLATFORM\iMX27ADS\SRC\inc\oemaddrtab_cfg.inc`

These source files are described in the following sections.

3.1.1 flash.c

It includes two functions as follows:

- `LPBYTE OEMMapMemAddr (DWORD dwImageStart, DWORD dwAddr)`
  Adds RAM map for the multi-BIN images, if `IMGNAND` is not set.
- `BOOL OEMWriteFlash(DWORD dwStartAddr, DWORD dwLength)`
  While programming the first BIN file, a message, WARNING: Flash update requested, appears and allows the user to select the Flash update. Before programming the first BIN file, the `NANDMakeMBR()` function is called to create the MBR first. After the last BIN is programmed, the file shows the message, Reboot the device manually, and calls the `SpinForever()` function.

3.1.2 main.c

It includes two functions as follows:

- `void OEMMultiBINNotify(const PMultiBINInfo pInfo)`
  Before downloading the image, the platform builder sends the `MultiBINInfo` variable, and this function is called once. This function records the number of multi-BIN images and saves to the global variable, `g_dwTotalBinNum`. The function also initializes `g_dwCurrentBinNum` variable to 0, which means to start from the first BIN.
- `BOOL OEMVerifyMemory(DWORD dwStartAddr, DWORD dwLength)`
  This function does the following operations:
  — Before each BIN file transfer, this function is called to verify the address and to translate the RAM buffer address.
  — Increments `g_dwCurrentBinNum` to calculate the BIN number.

— Adjusts the BinFS NK address between `IMAGE_BOOT_MBR_NAND_PA_START` and `IMAGE_BOOT_NANDDEV_NAND_PA_END`.

— Calls the `NANDCheckImageAddress()` to check if each BIN file size is aligned in the NAND Flash block size.

— Starting address and size for each BIN is stored in the `g_dwBinAddress` and `g_dwBinLength` arrays, respectively.

— After the last BIN is processed, this function `sets g_dwCurrentBinNum` to 0 for using the `OEMWriteFlash()` function.

3.1.3 nand.c

This is the main file that supports the multi-BIN images and BinFS in the EBOOT NAND Flash. This file includes the following functions:

- `static CHSAddr LBAtoCHS(FlashInfo *pFlashInfo, LBAAddr lba)`
  This fuction converts the Logical Block Addressing (LBA) address to Cylinder-Head-Sector (CHS) address and is used to create the partition table.
- `DWORD CheckSum(void *pAddr, DWORD dwLen)`
  CheckSum function is used to verify the data. The input to this function is the data buffer and the function returns the calculated checksum value.
- `static DWORD NANDGetRealBlockAddress(DWORD dwBlockLogAddress)`
  This function finds the real NAND Flash block address from the input logical address and skips the bad blocks. If bad blocks are not found, the real block address is taken as the logical address.
- `BOOL NANDWriteXldr(DWORD dwStartAddr, DWORD dwLength)`
- `BOOL NANDWriteEboot(DWORD dwStartAddr, DWORD dwLength)`
- `BOOL NANDWriteIPL(DWORD dwStartAddr, DWORD dwLength)`
  This fuction changes the data verification method to checksum.
- `BOOL NANDWriteNK(DWORD dwStartAddr, DWORD dwLength)`
  This function is called once to program the BIN file to the NAND Flash. To program three BIN files, this function is called thrice. For each BIN file, this function calculates the real data size aligned in the block size. The function then calculates the physical starting block address for the BIN region and programs the BIN data to the NAND Flash. The data is then read back for verification.
- `BOOL NANDMakeMBR(void)`
  This function creates MBR based on the received BIN files. For a BinFS image, the MBR creates two partitions: one for the RAMIMAGE and other for the NANDIMAGE. If the received image is an `NK.NB0` signal instead of a BinFS image, then the MBR creates only the RAMIMAGE partition. When the data is fully loaded into the MBR, the data is programmed into the NAND Flash MBR region and is read back to verify.
- `BOOL NANDStartWriteBinDIO(DWORD dwStartAddr, DWORD dwLength)`
- `BOOL NANDContinueWriteBinDIO(DWORD dwAddress, BYTE *pbData, DWORD dwSize)`
- `BOOL NANDLoadIPL(VOID)`
  This function uses the `NANDGetRealBlockAddress()` to convert logical block address to physical block address. The old code for the `NANDLoadIPL()` function used to start reading the data from the

fixed physical block address. However, the new code uses the logical block address to start reading the data.

- `BOOL NANDLoadNK(VOID)`
  This function checks the MBR and gets the starting block address and size of the RAMIMAGE. The function then reads the data from the NAND Flash to the RAM address, `IMAGE_BOOT_NKIMAGE_RAM_PA_START`.
- `BOOL NANDCheckImageAddress(DWORD dwPhyAddr)`
  This function checks if the BIN region starts from the aligned block address and returns TRUE for the aligned blocks.
- `BOOL NANDFormatNK(void)`
  This function uses `NANDGetRealBlockAddress()` function to convert the logical block address to physical block address. The old code for the `NANDFormatNK()` function used to start deleting the data from the fixed physical block address. However, the new code uses the logical block address to start deleting the data.

3.1.4 image_cfg.h

In this file, the NAND offset definition for the MBR region, in C code of size 128 Kbytes, is added between the IPL and NK regions.

3.1.5 image_cfg.inc

In this file, the NAND offset definition for MBR region, in ASM code of size 128 Kbytes, is added between the IPL and NK regions.

3.1.6 oemaddrtab_cfg.inc

In this file, the address map for the NFC space is changed from 1 Mbyte to 48 Mbytes. When the multi-BIN file is implemented, the starting address of the bin files can exceed 1Mbyte.

4 NAND Disk Reference

`nanddisk.dll` is the BinFS block device driver used to manage the MBR and multi-BIN regions in the NAND Flash.

4.1 NAND Disk Source Files

The NAND disk source files are as follows:

- `WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDDISK\makefile`
- `WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDDISK\nanddisk.def`
- `WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDDISK\nanddisk.h`
- `WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDDISK\sources`
- `WINCE600\PLATFORM\iMX27ADS\SRC\DRIVERS\BLOCK\NANDDISK\system.c`

These source files are described in the following sections.

4.1.1 nanddisk.def

This def file exports the DSK interface for the NAND disk device driver.

4.1.2 nanddisk.h

This file includes some definitions related to the MBR and the sector size. `nanddisk.h` file also defines the data structure for the DSK device:

```
typedef struct _DISK {
struct _DISK * d_next;
CRITICAL_SECTION d_DiskCardCrit;// guard access to global state and card
DISK_INFO d_DiskInfo;    // for DISK_IOCTL_GET/SETINFO
DWORD d_StartBlock;
DWORD d_TotalSize;
LPWSTR d_ActivePath;    // registry path to active key for this device
} DISK, * PDISK;
```

Some important variables used in this file are as follows:

- `d_StartBlock`
  This is the physical block address corresponding to the starting NAND Flash block of the NAND disk.
- `d_TotalSize`
  This gives the size of the NAND disk in bytes.

4.1.3 system.c

`system.c` is the main file in the NAND disk device driver of the BinFS disk. The BinFS disk is read-only, and the bad block manage method is simple in this driver. Here, a list has been created for the bad block and while accessing a block, the block is checked if it is in the bad block table.

This file includes the following variables and functions:

- `static DWORD g_dwBad[MAX_BADBLOCK_COUNT];`
  This array represents the bad block table.
- `static DWORD g_dwBadBlockNumber;`
  This variable gives the number of bad blocks in the bad block table.
- `static BOOL NAND_ReadSector(SECTOR_ADDR startSectorAddr, LPBYTE pSectorBuff, PSectorInfo pSectorInfoBuff, DWORD dwNumSectors)`
  This function reads the requested sector data and metadata from the Flash media and transfers the request to the OAL. The OAL then finishes the real NAND Flash access.
- `static DWORD NAND_GetBlockStatus(DWORD dwBlockID)`
  This function returns the status of the NAND Flash block.
- `static void InitBadBlockTable(PDISK pDisk)`
  This function initializes the NAND Flash bad block table, `g_dwBad[ ]`, and the variable, `g_dwBadBlockNumber`.
- `static DWORD GetRealBlockAddress(DWORD dwBlockLogAddress)`
  This function converts the logical block address to physical block address based on the bad block table.

- `static BOOL IsValidMbr(DWORD dwBlockID, DWORD * pdwDiskSize)`
  This function verifies the MBR for the NAND disk. If the MBR is valid, the function calculates the disk size and returns `pdwDiskSize`, which gives the disk size.
- `static BOOL GetNandDiskInfo(PDISK pDisk)`
  This function is called when the NAND disk driver is initialized. The function searches the MBR and initializes the `pDisk->d_StartBlock` and `pDisk->d_TotalSize`.
- `static PDISK CreateDiskObject(VOID)`
  This function creates a DISK structure and initializes some fields.
- `static BOOL IsValidDisk(PDISK pDisk)`
  This function verifies if the pDisk points to anything in the list. The function returns `TRUE` if the pDisk is valid and `FALSE` if the pDisk is invalid.
- `static HKEY OpenDriverKey(LPTSTR ActiveKey)`
  This function opens the driver key specified by the active key. The caller is responsible for closing the returned `HKEY`.
- `BOOL GetDeviceInfo(PDISK pDisk, PSTORAGEDEVICEINFO pInfo)`
  This function fills the `STORAGEDEVICEINFO` structure for `IOCTL_DISK_DEVICE_INFO`.
- `static BOOL GetFolderName(PDISK pDisk, LPWSTR FolderName, DWORD cBytes, DWORD * pcBytes)`
  This function retrieves the folder name value from the driver key. The folder name is used by FATFS to name the disk volume. The `GetFolderName()` function is used by the `DISK_IOCTL_GETNAME` and `IOCTL_DISK_GETNAME` macros.
- `static VOID CloseDisk(PDISK pDisk)`
  This function frees all the resources associated with the specified disk.
- `static DWORD DoDiskIO(PDISK pDisk, DWORD Opcode, PSG_REQ pSgr)`
  This function performs the requested input/output operation. This function is called from the `DSK_IOControl` function. The requests in the fuction are serialized by using the critical section of the disk.
- `static DWORD GetDiskInfo(PDISK pDisk, PDISK_INFO pInfo)`
  This function returns the disk information in response to the `DISK_IOCTL_GETINFO` macro.
- `static DWORD SetDiskInfo(PDISK pDisk, PDISK_INFO pInfo)`
  This function stores the disk information in response to `DISK_IOCTL_SETINFO`.
- `static DWORD GetSectorAddr(PDISK pDisk, DWORD dwSector)`
  This function converts the data address from the sector address in response to the `IOCTL_DISK_GET_SECTOR_ADDR` variable.
- `BOOL WINAPI DllEntry(HINSTANCE DllInstance, DWORD Reason, LPVOID Reserved)`
  This function gives dll entry to the NAND disk driver.

The interface functions for the WinCE block device are as follows:

- `DWORD DSK_Init(DWORD dwContext)`
- `BOOL DSK_Close(DWORD Handle)`
- `BOOL DSK_Deinit(DWORD dwContext)`
- `DWORD DSK_Open(DWORD dwData, DWORD dwAccess, DWORD dwShareMode)`
- `BOOL DSK_IOControl(DWORD Handle, DWORD dwIoControlCode, PBYTE pInBuf, DWORD nInBufSize, PBYTE pOutBuf, DWORD nOutBufSize, PDWORD pBytesReturned)`
- `DWORD DSK_Read(DWORD Handle, LPVOID pBuffer, DWORD dwNumBytes)`

- DWORD DSK_Write(DWORD Handle, LPCVOID pBuffer, DWORD dwNumBytes)
- DWORD DSK_Seek(DWORD Handle, long lDistance, DWORD dwMoveMethod)
- void DSK_PowerUp(void)
- void DSK_PowerDown(void)

5 Registry and BIB Settings

This section describes the registry and BIB settings.

5.1 Platform.reg

The block device driver, `nanddisk.dll`, is a read-only Flash driver to manage the RAMIMAGE and BinFS disk. The registry setting to support the BinFS on NAND disk is as follows:

```
IF BSP_SUPPORT_DEMAND_PAGING
IF BSP_NONANDDISK !
; HIVE BOOT SECTION
[HKEY_LOCAL_MACHINE\System\StorageManager\AutoLoad\NSDisk]
         "DriverPath"="Drivers\\BlockDevice\\NandDisk"
         "LoadFlags"=dword:1
         "MountFlags"=dword:11
         "BootPhase"=dword:0

IF SYSGEN_FSREGHIVE
         "Flags"=dword:1000
ENDIF

[HKEY_LOCAL_MACHINE\Drivers\BlockDevice\NandDisk]
         "Prefix"="DSK"
         "Dll"="NandDisk.dll"
         "Order"=dword:0
         "Ioctl"=dword:4
         "Profile"="NandDisk"
         "FriendlyName"="BINFS Flash Driver"
         "MountFlags"=dword:11
         "BootPhase"=dword:0

IF SYSGEN_FSREGHIVE
         "Flags"=dword:1000
ENDIF

; Bind BINFS to the block driver
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\NandDisk]
         "DefaultFileSystem"="BINFS"
         "PartitionDriver"="mspart.dll"
         "AutoMount"=dword:1
         "AutoPart"=dword:1
         "MountFlags"=dword:0
         "Folder"="BINFS"
         "Name"="BINFS Flash Disk"
         "BootPhase"=dword:0
         "MountAsROM"=dword:1
         "MountHidden"=dword:1
         "MountPermanent"=dword:1
```

```
IF SYSGEN_FSREGHIVE
        "Flags"=dword:1000
ENDIF
; END HIVE BOOT SECTION
ENDIF
ENDIF  ;BSP_SUPPORT_DEMAND_PAGING
```

5.2 Platform.bib

The `Platform.bib` file decides how to divide the BSP modules and files into XIPKERNEL and NK regions. To support both multi-BIN and non-multi-BIN systems, some macros are defined in the file header:

```
IF BSP_SUPPORT_DEMAND_PAGING !
        #define XIPKERNELNK
        #define NK        NK
ENDIF

IF BSP_SUPPORT_DEMAND_PAGING
        #define XIPKERNELXIPKERNEL
        #define NK        NK
ENDIF
```

5.3 Config.bib

The `Config.bib` file defines the memory map for the multi-BIN image. In retail building, the three BIN mapped address is followed.

The layout followed by the RAM is given as follows:

```
ARGS            88000000        00001000        RESERVED

VPU             88001000        000FF000        RESERVED

FRAMEBUFFER     88100000        00100000        RESERVED

XIPKERNEL       88200000        004FC000        RAMIMAGE

CHAIN           886FC000        00004000        RESERVED

NK              88700000        03B00000        NANDIMAGE


RAM             88700000        07900000        RAM
```

NANDIMAGE region based address can be of any kernel address value. It is used only by the EBOOT to download to the NAND Flash and to program it. The NANDIMAGE region based address does not affect the run-time. Here, the address follows the XIPKERNEL and CHAIN regions, and this makes the platform builder to generate the NB0 file (`XIP.NB0`) successfully. This file is merged with all the BIN files.

The XIPKERNEL region must map to the address it runs. In the i.MX27ADS BSP, the XIPKERNEL region runs from the physical RAM address, `IMAGE_BOOT_NKIMAGE_RAM_PA_START (0xA0200000)`, and so the `config.bib` is mapped to the address, `0x88200000 (IMAGE_BOOT_NKIMAGE_RAM_UA_START)`, with reference to `oemaddrtab_cfg.inc`.

6 Support Scripts

The bat and script files help the platform builder to generate the multi-BIN image successfully. This section describes some of the support scripts that help platform builder.

6.1 MakeBinfsBib.js

The `MakeBinfsBib.js` java script is used to process the final ce.bib file before making the image. All the WinCE public files that are required to be put into the XIPKERNEL region are listed in the file header. After running the script, the region name of the files changes from NK to XIPKERNEL.

NOTE

If the BSP variable, BSP_SUPPORT_DEMAND_PAGING, is not set, this script does not effect the ce.bib file.

6.2 PreRomImage.bat

`PreRomImage.bat` file is called by the platform builder automatically before the ROMIMAGE stage. In this file, the java script, `MakeBinfsBib.js`, is used to process the `ce.bib` file. With the processed `ce.bib` file, the ROMIMAGE can generate the multi-BIN images as designed by the user.

7 Patch for i.MX27ADS WinCE 6.0 F15 BSP

Unzip the patch file and overwrite to old the BSP folders, `sysgen` and `built`. The patch is tested on the 512-byte page size SLC NAND K9K1G08U0B.

The `iMX27ADS_BINFS.zip` file can be found in the zip file, `AN4137SW.zip`, that is included with the application note.

NOTE

The XLDR and EBOOT should be updated first.

8 Revision History

Table 1 provides a revision history for this application note.

Table 1. Document Revision History

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

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

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: AN4137
Rev. 0
06/2010