BINFS Implement Guide

AUTHOR	SIGN-OFF SIGNATURE #1	SIGN-OFF SIGNATURE #2
Justin. Jiang		
SIGN-OFF SIGNATURE #3	**SIGN-OFF SIGNATURE #4**	**SIGN-OFF SIGNATURE #5**

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	12.11.2008	Qiang.li	First version.
0.2	05.08.2009	Qiang.li	Added patch for iMX31PDK 1.4 WinCE6.0 BSP.
0.3	09.04.2009	Qiang.li	Added patch for iMX35PDK 1.5 WinCE6.0 BSP.
0.4	07.09.2010	Justin. Jiang	Added patch for iMX51PDK 1.7 WinCE6.0 BSP.
1.0	10.29.2010	Justin. Jiang	Added HIVE registry support for iMX51PDK1.7 WINCE6.0 BSP

Table of Contents

1 Introduction

1.1 Purpose

In traditional file system, the WinCE image is a signal file “NK.NB0”/”NK.BIN”. And when using NAND flash for storage, since it can’t support XIP, the total “NK.NB0” need be copied into RAM before running. The EBOOT will do this copy. In this way, there are two main shortages: Long boot time and big size RAM requirement. If the WinCE image is big (Included more features), these issues will be critical.

The BINFS can fix those two issues fine. It gave the chance to use 32MB RAM run 64MB WinCE image, this can cost down the final products.

In BINFS file system, the final WinCE image will be divided into multi-BIN files, and only the XIPKERNEL BIN (Less than 7 MB) need be copied into RAM by EBOOT. The files in other BIN will work with demand paging mode. These files will be loaded into RAM only when they need run.

With the BINFS support, the boot time (From start reading Flash image to show WinCE desktop) can be reduced to 4 seconds on Zhonghong board, and the free RAM size can grow to MB for storage and program.

1.2 Scope

The scope of this document is limited to address the following things:

- Introduce the BINFS support in Freescale WinCE 6.0 BSP based on NAND Flash.

1.3 Audience Description

This document is intended for anyone who was concerned for the NAND Flash boot time and the RAM using for NK image.

1.4 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
BSP	Board support package
CSP	Chip support package
OAL	OEM Adaptation Layer
BINFS	Binary ROM image file system

XIP	Execute In Place
BIN	WinCE Binary Image

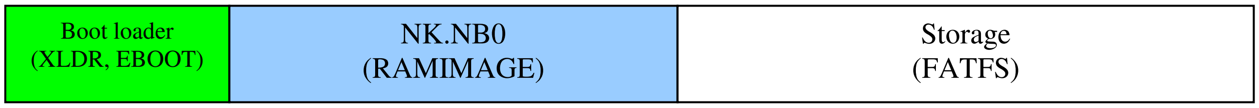
2 Design References

2.1 Framework description

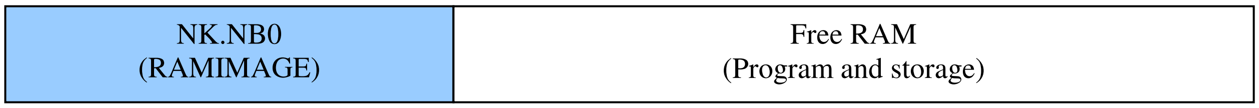
In this document, the BINFS will base on the Multi-NandDisks solution (Reference to Multi-NandDisks_Implement_Guide.doc); we will create a read-only disk (NANDDISK) to manage the BINFS partition:

- EBOOT: Support multi-BIN WinCE image, support BINFS.
- NANDDISK: The block device driver to manage the BINFS partition.
- Registry setting and bib files: To support BINFS.
- Support scripts: To process the “ce.bib” for BINFS support.

In not BINFS file system, the total WinCE image is RAMIMAGE, the followed is the NAND Flash and RAM layout:



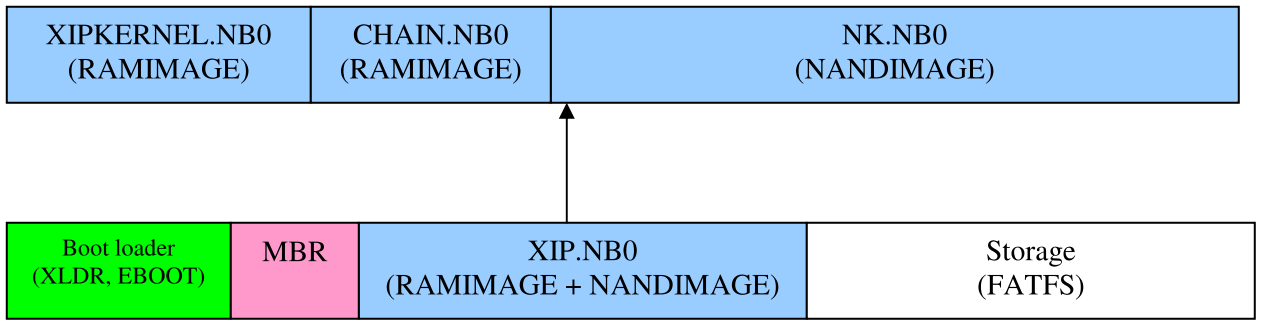
NAND Flash Layout



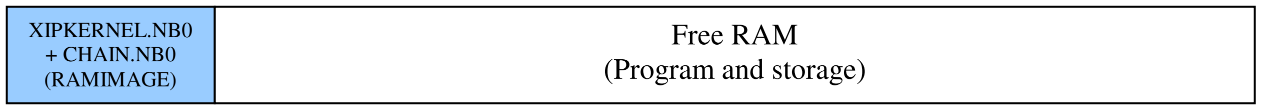
RAM Layout

In this layout, when EBOOT loading the “NK.NB0” into RAM to run, it will read 48MB fixed size data from NAND Flash and copy to WinCE NK run RAM base address “IMAGE_BOOT_NKIMAGE_RAM_PA_START” (Defined in image_cfg.h).

In BINFS file system; only the XIPKERNEL.NB0 and CHAIN.NB0 is RAMIMAGE, others are NANDIMAGE. The total RAMIMAGE size can be less than 7MB. The “CHAIN.NB0” can be used for EBOOT to locate each BIN region, it was only required on old version WinCE. The followed is the NAND Flash and RAM layout:



NAND Flash Layout


XIPKERNEL.NB0 + CHAIN.NB0 (RAMIMAGE)
Free RAM (Program and storage)

RAM Layout

In the new layout, when EBOOT loading WinCE image, it will first check the MBR, in the MBR, it will include the RAMIMAGE partition (Partition type 0x23, PART_RAMIMAGE) and NANDIMAGE partition (Partition type 0x21, PART_BINFS). Only the RAMIMAGE partition needs be copied into RAM. And EBOOT can also get the logical start sector address and sector size of the RAMIMAGE partition, and then we can calculate the real size of data that needs be copied to WinCE NK run RAM base address "IMAGE_BOOT_NKIMAGE_RAM_PA_START".

Both the MBR and XIP.NB0 region will be managed by the NAND Flash block device driver: Nanddisk.dll.

In the XIPKERNEL.NB0, there are only necessary modules for boot up. This region is where files that must be loaded prior to implementation of BINFS are stored. Because the kernel must reside in RAM and the XIPKERNEL region is RAMIMAGE, the files loaded into the XIPKERNEL region typically include everything needed for the kernel. We use the script "FILES\MakeBinfsBib.js" to decide which modules need be added into XIPKERNEL region. The BSP drivers, which needs be put into XIPKERNEL region, should be specified in "platform.bib" directly.

Microsoft suggestion the followed modules should be put into XIPKERNEL:

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)
The flash driver

If the flash driver is loaded by the device manager, also add device.dll, devmgr.dll, regenum.dll, busenum.dll, and pm.dll to the XIPKERNEL region.

For kernel independent transport layer (KITL) support, also add kitl.dll to the XIPKERNEL region.

For debugging support, also add hd.dll, osaxst0.dll, and osaxst1.dll to the XIPKERNEL region.

2.2 Configurations

In this example, we will create three BINs for BINFS support: XIPKERNEL.BIN, CHAIN.BIN and NK.BIN.

In not BINFS file system, the final WinCE image is "NK.BIN" and "NK.NB0". But in BINFS file system, the final image will be "XIP.BIN" and "XIP.NB0". In fact these two files are merged from "XIPKERNEL.BIN", "CHAIN.BIN" and "NK.BIN", there will be another file "chain.lst". This file will be selected by the platform builder to download the BINFS image.

List all changes need be done in the iMX51 WinCE 6.0 BSP to support the BINFS, all changes are in BSP folder:

a. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\iMX51-EVK-PDK1_7.bat":**

```
REM NAND Flash Driver
set IMGNAND=1
set BSP_NAND_FMD=1
set BSP_NONANDDISK=1
if "%BSP_NAND_FMD%"=="1" set SYSGEN_FLASHMDD=1

REM Binfs suport.
set BSP_SUPPORT_MULTIBIN=1
if "%BSP_SUPPORT_MULTIBIN%"=="1" set BSP_NONANDDISK=
if "%BSP_SUPPORT_MULTIBIN%"=="1" set SYSGEN_BINFS=1
```

b. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\sources.cmn":**
Add macro define for "BSP_SUPPORT_MULTIBIN", this macro will be used in

```
!IF "$(BSP_SUPPORT_MULTIBIN)" != ""
CDEFINES=$(CDEFINES) -DBSP_SUPPORT_MULTIBIN
ADEFINES=$(ADEFINES) -pd "BSP_SUPPORT_MULTIBIN SETL {TRUE}"
!ENDIF
```

c. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\FILES\Config.bib":**
Change the memory map to support BINFS.

d. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\FILES\MakeBinfsBib.js"**

Java script file used to process the "ce.bib" for multi-BIN support.

e. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\FILES\Platform.bib":**
Use macro to define different BIN name.
Add the block driver for BINFS support.
Add the Flash PDD driver "flashpdd_nand.dll" and the auto-run application "InstallNand.exe" in MODULES.

f. **File "\WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\DRIVERS\BLOCK\NANDFMD\nand.reg":**
Add registry setting for multi-NANDDisks support.
Add registry setting for BINFS support.

g. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\FILES\ PreRomImage.bat":**
This file will be called before platform builder "romimage" stage; it will process the "ce.bib" file for multi-BIN support.

h. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\INC\image_cfg.h":**
Add NAND address definition for MBR, needed for BINFS.

i. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\INC\image_cfg.inc":**
Add NAND address definition for MBR, needed for BINFS.

j. **File "WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\INC\oemaddrtab_cfg.inc":**
Map more address for NFC to support multi-BIN in EBOOT.

k. **File "\WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\BOOTLOADER\XLDR\NAND\xldr.bib":**
Change the ROMOFFSET to support BINFS.

l. **File "\WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\BOOTLOADER\EBOOT\eboot.bib":**
Change the ROMOFFSET to support BINFS.

3 EBOOT Reference

The main function for the EBOOT to support multi-BIN and BINFS:

a. Use different RAM buffer to receive the downloaded multi-BIN files;

b. Make MBR for WinCE NANDDISK driver before program image to NAND Flash;

c. Check the MBR before loading WinCE image to RAM, only copy the RAMIMAGE data to IMAGE_BOOT_NKIMAGE_RAM_PA_START.

When downloading the image from PC to board, platform builder will send the BIN file one by one based on the file "chain.lst". The old EBOOT will receive all BIN files with a same RAM buffer; it can't be used to receive the multi-BIN image. The new EBOOT will use different RAM address to receive those BIN files, based on the different start address of each BIN file.

And the EBOOT will also record the start address and size for each BIN, after all BIN files were downloaded to board, based on the recorded start address and size, EBOOT can program them to NAND Flash one by one. In this example, the first BIN is the RAMIMAGE and the last BIN is the chain. Before programming the BIN data, the EBOOT need create a MBR for the received WinCE image. Here, the MBR will include two partitions, one for RAMIMAGE (PART_RAMIMAGE) and another for NANDIMAGE (PART_BINFS). The size for each partition should be calculated from the received BIN files. We reserved 128KB for the MBR region, but the real data for MBR is only 512 Bytes.

The first three bytes of the MBR are "0xE9 0xFD 0xFF"; and the last two bytes of the MBR are "0x55 0xAA". At the end of MBR, there are for partition tables, structure as followed:

```
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 one BIN file was programmed to NAND Flash, EBOOT will read back the data to verify. In the old EBOOT, the read back data will be stored to RAM that followed the receive buffer. This can't be applied on multi-BIN image, another BIN data maybe is at that address. So we use "check sum" to verify the BIN data.

3.1 EBOOT Source Files

WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\BOOTLOADER\COMMON\flash.c

WINCE600\PLATFORM\iMX51-EVK-PDK1_7\SRC\BOOTLOADER\COMMON\main.c

WINCE600\PLATFORM\COMMON\SRC\SOC\COMMON_FSL_V2_PDK1_7\BOOT\FMD\
NAND \nandboot.c

WINCE600\PLATFORM\iMX51-EVK-PDK1_7 \SRC\INC\image_cfg.h

WINCE600\PLATFORM\iMX51-EVK-PDK1_7 \SRC\INC\image_cfg.inc

WINCE600\PLATFORM\iMX51-EVK-PDK1_7 \SRC\INC\oemaddrtab_cfg.inc

3.2 Flash.c

LPBYTE OEMMapMemAddr (DWORD dwImageStart, DWORD dwAddr):

Add RAM map for multi-BIN files, if without "IMGNAND=1".

BOOL OEMWriteFlash(DWORD dwStartAddr, DWORD dwLength):

Only when programming the first BIN file; it will print "WARNING: Flash update requested." And let user to select for flash update.

Before programming the first BIN file; it will call NANDMakeMBR() to create the MBR first.

After the last BIN programmed, it will show message "Reboot the device manually..." and call SpinForever().

3.3 Main.c

void OEMMultiBINNotify(const PMultiBINInfo pInfo):

Before downloading the image, platform builder will send the "MultiBINInfo". And this function will be called once. This function will record the number for multi-BIN and save to global variable “g_dwTotalBinNum”. It will also initialize “g_dwCurrentBinNum” to 0; that means start from the first BIN.

BOOL OEMVerifyMemory(DWORD dwStartAddr, DWORD dwLength):

Before each BIN file transfer, this function will be called to verify the address and translate the RAM buffer address.

Added “g_dwCurrentBinNum ++;” to calculate the BIN number.

Adjusted the BINFS NK address between “IMAGE_BOOT_MBR_NAND_PA_START” and “IMAGE_BOOT_NANDDEV_NAND_PA_END”.

Each BIN file size should align in NAND Flash block size, this function will call NANDCheckImageAddress() to check this.

Start address and size for each BIN will be stored into “g_dwBinAddress” and “g_dwBinLength” array.

After the last BIN was processed, it will set “g_dwCurrentBinNum” to 0 for OEMWriteFlash() using.

3.4 Nandboot.c

This is the main file to support the multi-BIN and BINFS on NAND flash in EBOOT.

static CHSAddr LBAtoCHS(FlashInfo *pFlashInfo, LBAAddr lba):

Convert from LBA address to CHS address, used to create the partition table.

DWORD CheckSum(void *pAddr, DWORD dwLen):

Check sum function, used to verify the data. Input the data buffer and return the calculated check sum value.

static DWORD NANDGetRealBlockAddress(DWORD dwBlockLogAddress):

This function find the real NAND Flash block address from the input logical address, skip bad blocks. If there's no bad block, the real block address is same as the logical address.

BOOL NANDWriteXldr(DWORD dwStartAddr, DWORD dwLength):

BOOL NANDWriteEboot(DWORD dwStartAddr, DWORD dwLength):

BOOL NANDWriteNK(DWORD dwStartAddr, DWORD dwLength):

When programming one BIN file to NAND Flash, this function will be called once. For the three BIN, this function will be called thrice.

For each BIN, this function will calculate the real data size aligned in block size. Then it will calculate physical start block address for the BIN region, and program the BIN data to NAND Flash, read back to verify.

BOOL NANDMakeMBR(void):

This function will create the MBR based on the received BIN files. For BINFS image, it will create two partitions, one for RAMIMAGE and another for NANDIMAGE.

If the received image is not BINFS, just a signal "NK.NB0", then the MBR will include only one RAMIMAGE partition.

After filled the MBR data, it will program the data to NAND Flash MBR region, and read back to verify.

BOOL NANDStartWriteBinDIO(DWORD dwStartAddr, DWORD dwLength):

BOOL NANDContinueWriteBinDIO(DWORD dwAddress, BYTE *pbData, DWORD dwSize):

BOOL NANDLoadNK(VOID):

This function will check the MBR and get the RAMIMAGE start block address and size, and then read the data from NAND Flash to RAM address "IMAGE_BOOT_NKIMAGE_RAM_PA_START".

BOOL NANDCheckImageAddress(DWORD dwPhyAddr):

This function will be used to check if the BIN region starts from aligned block address. Return TRUE for aligned.

BOOL NANDFormatNK(void):

Use NANDGetRealBlockAddress() to convert logical block address to physical block address. Replaced the fixed start block address.

3.5 Image_cfg.h

Added the NAND offset definition for MBR region in C code, size is 2M, between Eboot region and NK region.

3.6 Image_cfg.inc

Added the NAND offset definition for MBR region in ASM code, size is 2M, between Eboot region and NK region.

3.7 Oemaddrtab_cfg.inc

Changed the address map for NFC space, changed from 5MB to 16MB. After implemented multi-BIN, the start address for the BIN files can overtop the 1 MB space.

4 NANDDISK Reference

The “nanddisk.dll” is the BINFS block device driver to manage the MBR and multi-BIN regions on NAND Flash.

4.1 NANDDISK Source Files

WINCE600\PLATFORM\iMX51-EVK-PDK1_7 \SRC\DRIVERS\BLOCK\NANDDISK\ makefile

WINCE600\PLATFORM\ iMX51-EVK-PDK1_7 \SRC\DRIVERS\BLOCK\NANDDISK\ nanddisk.def

WINCE600\PLATFORM\ iMX51-EVK-PDK1_7 \SRC\DRIVERS\BLOCK\NANDDISK\ nanddisk.h

WINCE600\PLATFORM\ iMX51-EVK-PDK1_7 \SRC\DRIVERS\BLOCK\NANDDISK\ sources

WINCE600\PLATFORM\ iMX51-EVK-PDK1_7 \SRC\DRIVERS\BLOCK\NANDDISK\ system.c

4.2 Nanddisk.def

This def file exports the DSK interface for the Nanddisk device driver.

4.3 Nanddisk.h

This file included some definition for sector size and MBR related.

It also defined the data structure for 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;
```

d_StartBlock:

The start NAND Flash block for the nanddisk. This is the physical block address.

d_TotalSize:

The size of nanddisk, in bytes.

4.4 System.c

This is the main file for NAND disk device driver to BINFS disk. This disk is read-only, the bad block manage method is simple in this driver, we only created a list for bad block, when accessing a block, just check if it is in the bad block table.

static DWORD g_dwBad[MAX_BADBLOCK_COUNT];

The bad block table.

static DWORD g_dwBadBlockNumber;

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. It just transfer the request to OAL, the OAL will finish the real NAND Flash access.

static DWORD NAND_GetBlockStatus(DWORD dwBlockID):

This function returns the status of a NAND Flash block.

static void InitBadBlockTable(PDISK pDisk):

This function initialized the bad table, fill in g_dwBad[ ] and g_dwBadBlockNumber.

static DWORD GetRealBlockAddress(DWORD dwBlockLogAddress):

This function converted the logical block address to physical block address, based on the bad block table.

static BOOL IsValidMbr(DWORD dwBlockID, DWORD * pdwDiskSize):

This function verified the MBR for the nanddisk, if it is a valid MBR, it will calculate the disk size and return from "pdwDiskSize".

static BOOL GetNandDiskInfo(PDISK pDisk):

This function will be called when nanddisk driver initialized, it will search the MBR and initialize the "pDisk->d_StartBlock" and "pDisk->d_TotalSize".

static PDISK CreateDiskObject(VOID):

Create a DISK structure, initialize some fields and link it.

static BOOL IsValidDisk(PDISK pDisk):

Verify that pDisk points to something in our list. Return TRUE if pDisk is valid, FALSE if not.

static HKEY OpenDriverKey(LPTSTR ActiveKey):

Function to open the driver key specified by the active key. The caller is responsible for closing the returned HKEY.

BOOL GetDeviceInfo(PDISK pDisk, PSTORAGEDEVICEINFO pInfo):

Fill the STORAGEDEVICEINFO structure for IOCTL_DISK_DEVICE_INFO.

static BOOL GetFolderName(PDISK pDisk, LPWSTR FolderName, DWORD cBytes, DWORD * pcBytes):

Function to retrieve the folder name value from the driver key. The folder name is used by FATFS to name this disk volume. Related to DISK_IOCTL_GETNAME and IOCTL_DISK_GETNAME.

static VOID CloseDisk(PDISK pDisk):

Free all resources associated with the specified disk.

static DWORD DoDiskIO(PDISK pDisk, DWORD Opcode, PSG_REQ pSgr):

Perform requested I/O. This function is called from DSK_IOControl. Requests are serialized using the disk's critical section.

static DWORD GetDiskInfo(PDISK pDisk, PDISK_INFO pInfo):

Return disk info in response to DISK_IOCTL_GETINFO.

static DWORD SetDiskInfo(PDISK pDisk, PDISK_INFO pInfo):

Store disk info in response to DISK_IOCTL_SETINFO.

static DWORD GetSectorAddr(PDISK pDisk, DWORD dwSector):

Convert data address from sector address. Response to IOCTL_DISK_GET_SECTOR_ADDR.

BOOL WINAPI DllEntry(HINSTANCE DllInstance, DWORD Reason, LPVOID Reserved):

Dll entry for the nanddisk driver.

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):

Interface functions for DSK device.

5 Support Scripts

These bat and script files can help the platform builder to generate the multi-BIN image successfully.

5.1 MakeBinfsBib.js

This java script will be used to process the final “ce.bib” file before make image. All WinCE public files that need be put into XIPKERNEL region will be listed at the file header. After this script run, there files region name will be changed from “NK” to “XIPKERNEL”.

If BSP variable “BSP_SUPPORT_MULTIBIN” hasn’t been set, this script will do nothing for the “ce.bib”.

5.2 PreRomImage.bat

This BAT file will be called by platform builder automatically before “ROMIMAGE” stage. In the file, we will use the “MakeBinfsBib.js” to process the “ce.bib” file. With the processed “ce.bib”, ROMIMAGE can generate the multi-BIN images as we designed.

6 How to Download Multi-Bin Image

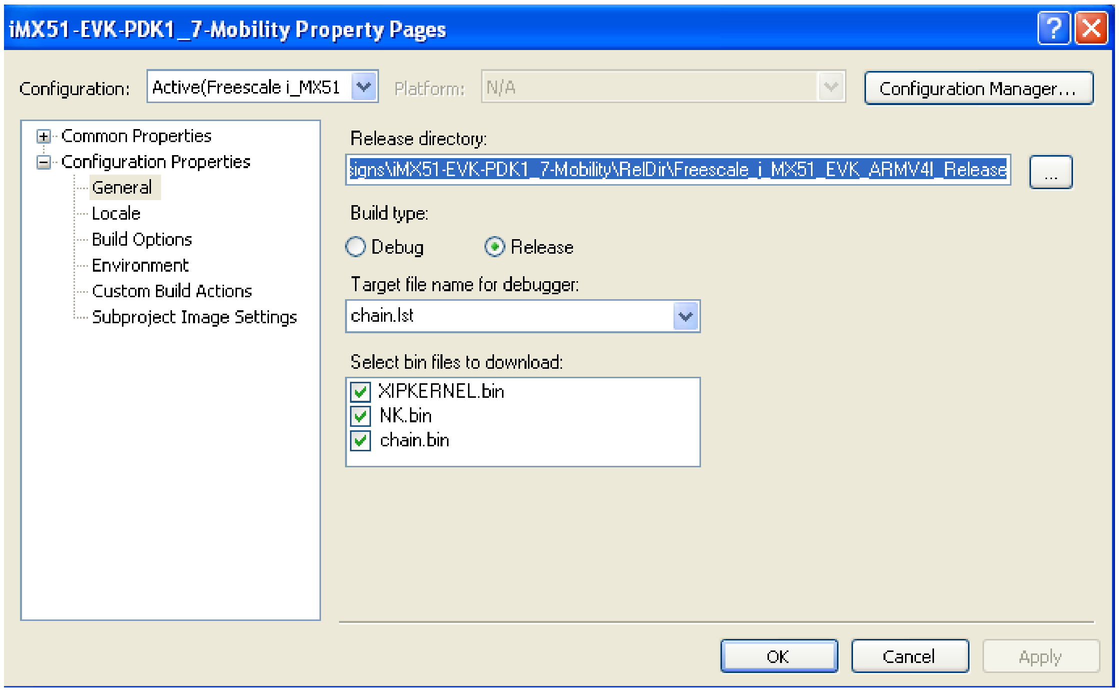
There are two methods to download the multi-bin image.

6.1 Download by VS2005

After you build the multi-bin image, there will have three bin files need to download to the device one by one.

a. Xipkernel.bin
b. NK.bin
c. Chain.bin

Before connect to the device, please set on VS2005. Go to "Project ->Properties->Configuration Properties->General"
In the "Target file name for debugger", please select "chain.lst"
Then you can see in "Select bin files to download" have the three files be checked in the checkbox.



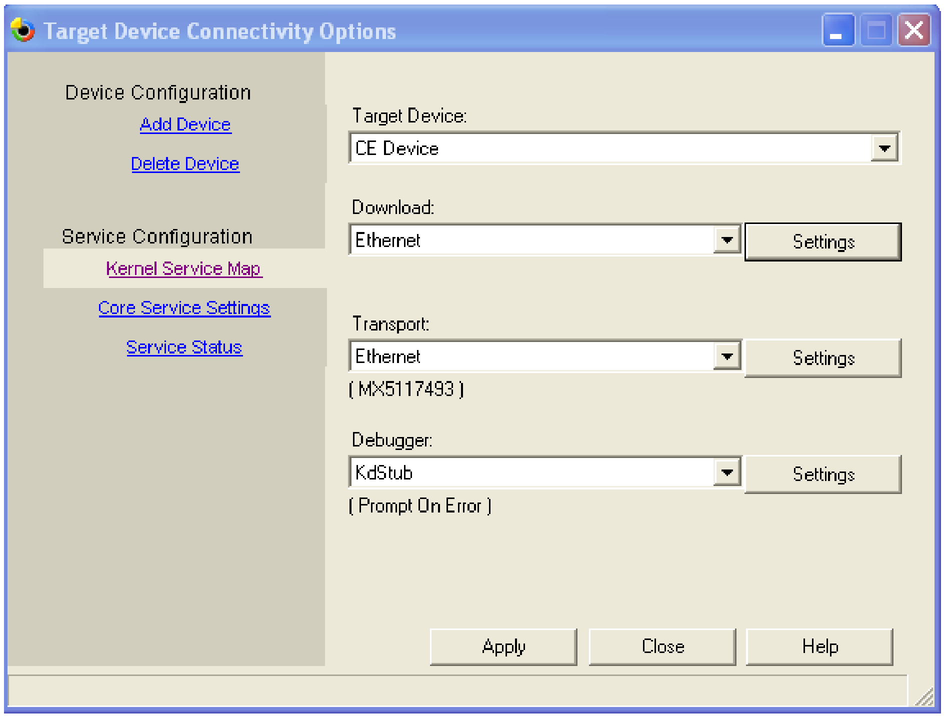
Configuration: Active(Freescale i_MX51
Platform: N/A
Release directory: signs\iMX51-EVK-PDK1_7-Mobility\RelDir\Freescale_i_MX51_EVK_ARMV4I_Release
Build type: Debug [✓ Release]
Target file name for debugger: chain.lst
Select bin files to download:
✓ XIPKERNEL.bin
✓ NK.bin
✓ chain.bin

Press "Attach Device" button, the download will start. During the download, Eboot will create a MBR region in the flash for the RAMIMAGE and BINFS partition.
In the eboot, please use USB RNDIS or FEC to download the multi-bin image, USB Serial is not support for multi-bin image download.

You can refer below setting, set the IP address in the PC to 192.168.0.XXX in the same area with the device.

```
---------------------------------------------------------------------------
Freescale iMX SOC Menu Item
---------------------------------------------------------------------------
 [0] IP Address : 192.168.0.100
 [1] Set IP Mask : 255.255.255.0
 [2] Boot Delay : 5
 [3] DHCP : Disabled
 [4] Reset to Factory Default Configuration
 [5] Select Boot Device : NK from NAND
 [6] Set MAC Address : E0-B8-20-33-44-55
 [7] Format OS NAND Region
 [8] Format All NAND Regions
 [9] Bootloader Shell
 [I] KITL Work Mode : Interrupt
 [K] KITL Enable Mode : Disable
 [P] KITL Passive Mode : Disable
 [S] Save Settings
 [D] Download Image Now
 [L] Launch Existing Flash Resident Image Now
 [E] Select Ether Device : USB RNDIS
 [M] MMC and SD Utilities
```

In the Visual Studio 2005 set the connectivity Option as below:



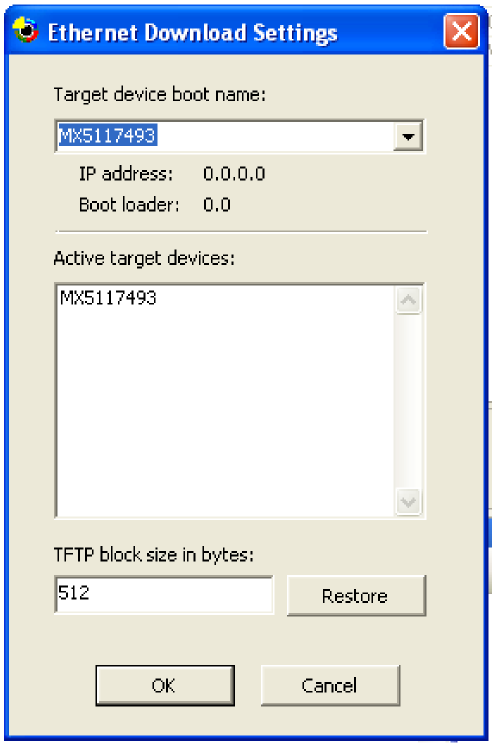
Target Device: CE Device

Download: Ethernet

Transport: Ethernet
(MX5117493)

Debugger: KdStub
(Prompt On Error)

When the USB RNDIS is connected, you can see the Active Target Device of "MX5117493".


Ethernet Download Settings

Target device boot name: MX5117493

IP address: 0.0.0.0

Boot loader: 0.0

Active target devices:

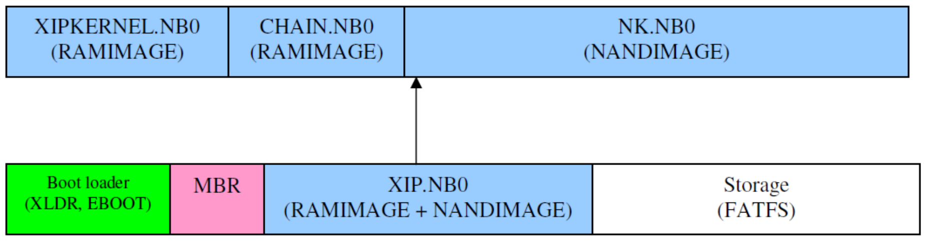
MX5117493

TFTP block size in bytes: 512

6.2 Download by Advanced Toolkit

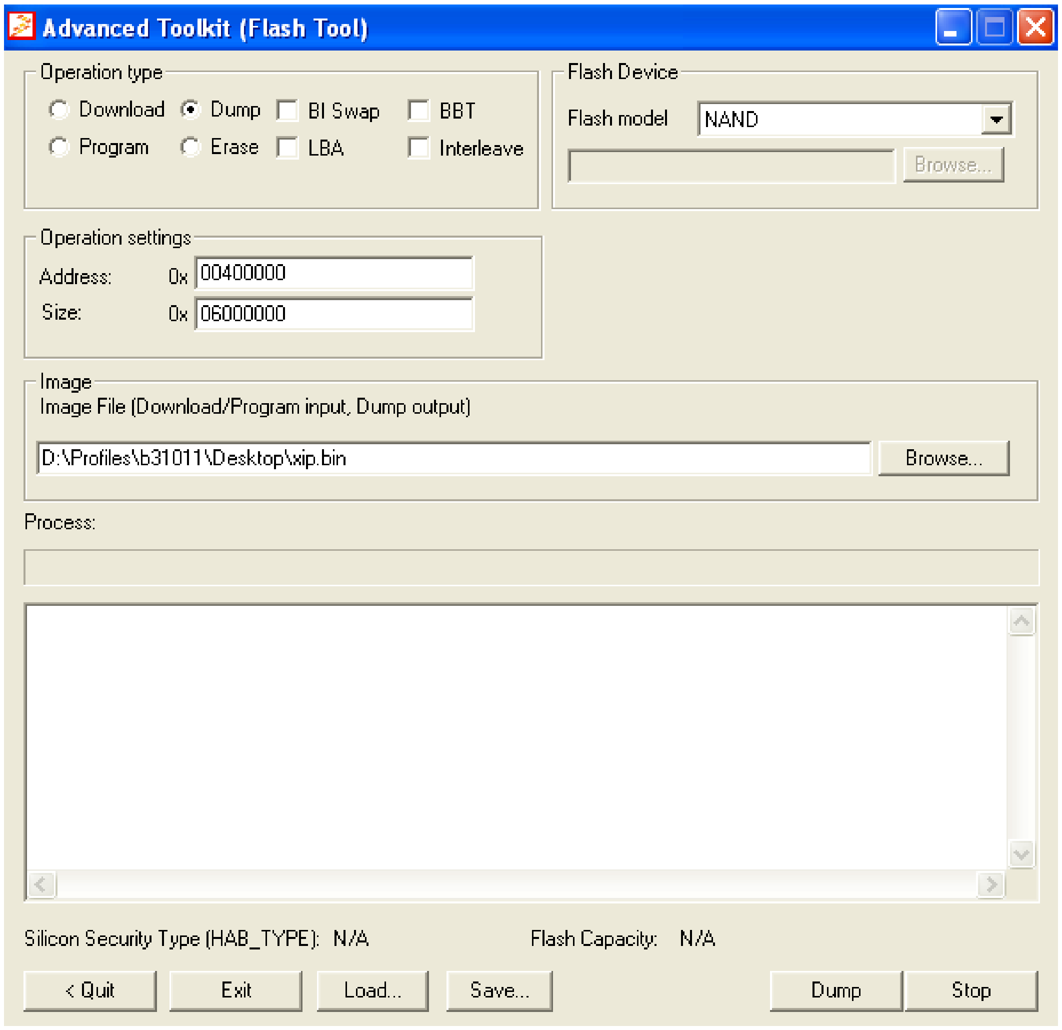
Because we need create a MBR for multi-bin image, and it is created during download by VS2005. So before we download image by Advanced Toolkit, we MUST use VS2005 to down-load the images for the first time. Then we can dump the binary image from the flash by Advanced Toolkit. After that, we can download the dumped BIN file by Advanced Toolkit.

a. Dump the multi-Bin files
We need dump the files of "MBR+Xipkernel+Chain+NK"
for the multi-bin images. See below picture:



Set the start Address in Advanced Toolkit of the MBR start address. It can be calculated by the XLDR and EBOOT size. For example, we set the XLDR and EBOOT 2MB in the Patch, so the start Address is: 0x400000. And the Size is the MBR(2MB)+IMAGE(94MB)=96MB.

See below picture, click Dump button to dump to “xip.bin” file.


Advanced Toolkit (Flash Tool)

Operation type: ( ) Download [✓ Dump] ( ) Program ( ) Erase ☐ BI Swap ☐ LBA ☐ BBT ☐ Interleave

Flash Device — Flash model: NAND

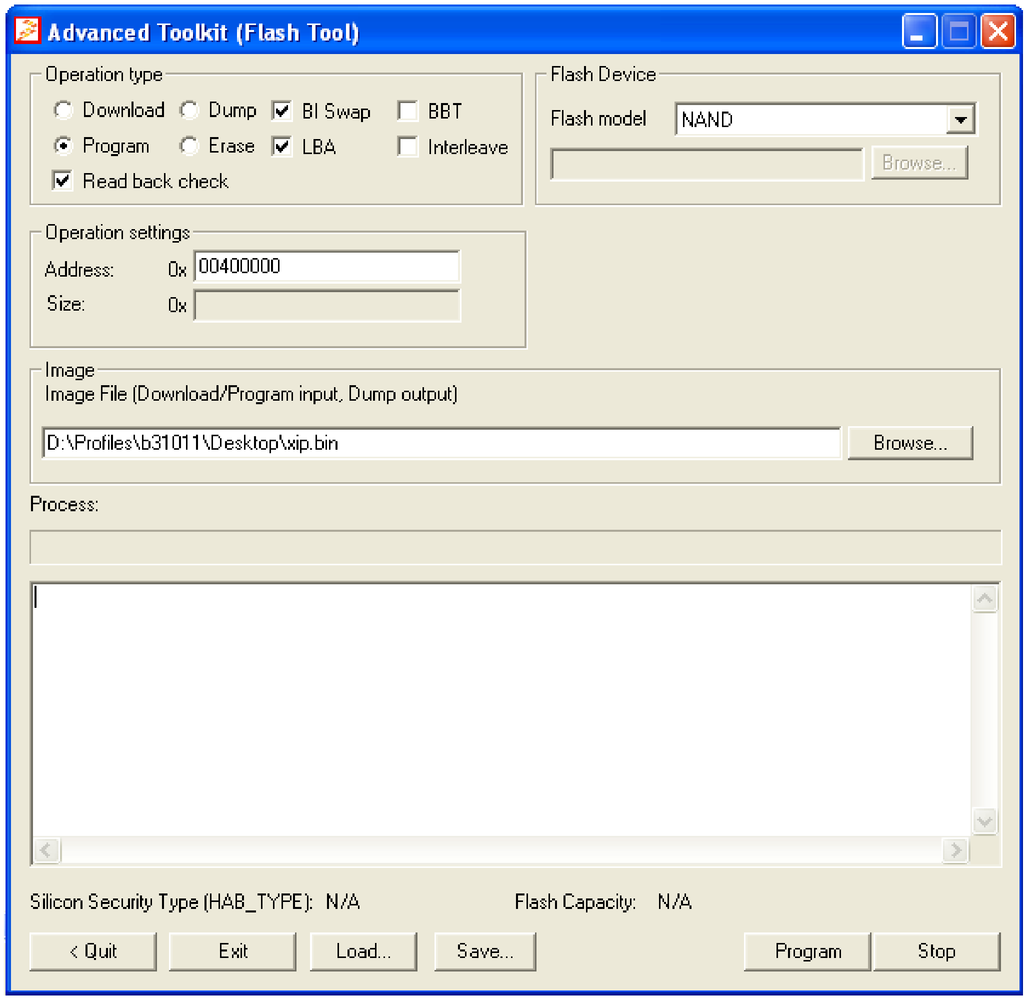
Operation settings
Address: 0x 00400000
Size: 0x 06000000

Image
Image File (Download/Program input, Dump output)
D:\Profiles\b31011\Desktop\xip.bin

Process:

Silicon Security Type (HAB_TYPE): N/A
Flash Capacity: N/A

b. Download the dumped file.
Use the Advanced Toolkit to download the dumped file of xip.bin to the start Address of the MBR address 0x400000.
Click “Program” to start download it.



Advanced Toolkit (Flash Tool)

Operation type: Download [✓ Program] Dump Erase; BI Swap ✓; LBA ✓; BBT ☐; Interleave ☐; Read back check ✓

Flash Device — Flash model: NAND

Operation settings — Address: 0x 00400000; Size: 0x

Image — Image File (Download/Program input, Dump output): D:\Profiles\b31011\Desktop\xip.bin

Process:

Silicon Security Type (HAB_TYPE): N/A Flash Capacity: N/A