How to store the Windows CE image on a NAND Flash memory

Introduction

Windows CE is a prevalent operating system for embedded systems, in particular, consumer electronics. Windows Mobile, an OS-based Windows CE architecture, is becoming popular in high-tier mobile phones, especially smart phones and Pocket PCs. NAND Flash or NAND-based storage devices are increasingly deployed in these high-end mobile phones as a data storage medium. Because of their relative low cost compared with NOR Flash, NAND devices are also used for code storage including OS image files and data storage.

This application note describes NAND Flash from the viewpoint that it is the main storage medium in a Windows CE system. It is used to store Windows CE images and other types of data storage. This solution is becoming more and more popular in cellular phones that use Windows CE or Windows Mobile.

This project is implemented on a small page NAND Flash, the NAND512W3, however, it is easy to port the solution to a large page NAND Flash by providing a NAND Flash media driver for large page NAND Flash. Our platform runs Windows CE.NET 4.2, and the principles of storing OS images on NAND Flash memories are similar to the later versions of Windows CE, such as 5.0/6.0.
Contents

1 Overview ................................................................. 5
2 Eboot architecture ...................................................... 6
3 NAND Flash allocation .................................................. 8
4 Eboot functionalities ................................................... 10
5 Storing single binary OS images on a raw NAND ............... 13
6 Storing a multiple region OS image on BINFS ................ 14
7 Summary ................................................................. 16
8 References ............................................................... 17
9 Revision history ......................................................... 18
List of tables

Table 1. Eboot options and functions .................................................. 11
Table 2. Document revision history ...................................................... 18
List of figures

Figure 1. Trends in mobile phone memory subsystems ............................................. 5
Figure 2. Eboot block diagram ................................................................. 6
Figure 3. OMAP5912 platform memory map ....................................................... 8
Figure 4. OMAP5912 platform memory map if Eboot is saved on NAND .................. 8
Figure 5. UBOOT command prompt ............................................................. 10
Figure 6. Eboot window ........................................................................ 11
Figure 7. Platform settings window .............................................................. 13
Figure 8. Platform Settings window of Platform Builder ...................................... 14
1 Overview

This project is implemented on the OMAP5912 platform, and OS Windows CE.NET. The WinCE BSP (board support package) is developed internally. OMAP5912 is a dual-core processor, comprising ARM926EJ-S core plus a DSP core. On this platform, NAND Flash Controller (NFC) is implemented with a SRAM/NOR memory controller in a microprocessor.

In old cellular phones, NOR Flash dominates the market, however, its market share is now being gradually invaded by NAND Flash due to the advantages of NAND Flash of lower cost and better program performance. NAND Flash is a required memory device in high-end mobile phones. Because the hardware NAND controller is integrated in a digital baseband processor or application processor, it is now becoming the code storage media as well as data storage. Figure 1 illustrates the memory subsystem trend in mobile phones.

Figure 1. Trends in mobile phone memory subsystems

In the development phase, the WinCE image should be downloaded from Platform Builder through an Ethernet connection, however, in the final products, the OS image must be burned into a Flash memory or masked into ROM.

A bootloader is an integral part during the Windows CE device development phase, and in some cases, the bootloader is also packaged in the final product. The basic purpose of a bootloader is to place the OS image into the memory and then jump to the OS startup routine. The bootloader can obtain the OS image in several different ways, including loading the image over a cabled connection, such as an Ethernet, USB or serial cable. In the final product, the bootloader may load the OS image from a local storage device, such as a native Flash memory (NOR/NAND), compact Flash or hard disk. The bootloader may store the OS image in a RAM with a battery backup, or in non-volatile memory. An example of such a bootloader is Ethernet Boot Loader (Eboot).
2 Eboot architecture

Eboot uses an Ethernet cable as the communication channel with the host PC. An Ethernet connection can be replaced with other connection channels, such as a USB connection, which is common on the mobile phone platform. In many mobile phones, a USB cable is not only used as a data transportation channel, but also as a power charging cable.

In the current project, MDK5912 uses an Ethernet port in the bootloader. Eboot is composed of:
- Eboot code library
- BLCOMMON library
- OEM code
- Network device driver
- Memory device driver

Figure 2 shows a typical Eboot block diagram.

The BLCOMMON library is a standard code library provided by Microsoft, and should not be modified. BLCOMMON implements the basic framework of the bootloader. It handles relocating the bootloader to RAM for faster execution, decoding the .bin file contents, verifying checksums and keeping track of load progress. The library calls well-defined OEM functions throughout the bootloading process to handle platform or solution-specific customizations.

The Eboot library provides the Dynamic Host Configuration Protocol (DHCP), Trivial File Transfer Protocol (TFTP), and User Datagram Protocol (UDP) services.

The BOOTPART library provides storage partitioning routines, which assist the bootloader in the partitioning storage, such as Flash memory, and in reading from or writing to the storage device. The library works with lower-level Flash memory access routines. This library can be used to separate multiple BIN regions or to multi-execute in place (XIP), from file system storage areas on a NAND Flash memory-based device.

OEM code initializes the critical hardware components after Eboot starts to manage the system. The Eboot library handles the real work of a bootloader.

Figure 2. Eboot block diagram
In devices based on WINCE versions earlier than 4.0, the OS image is stored in a NOR Flash memory, EEPROM or Compact Flash card, from where the image code can execute without being loaded into RAM. These memories are XIP devices. However, XIP memories are usually more expensive compared to NAND Flash memories, which are currently prominent in consumer electronic devices.

To support NAND Flash, Eboot must be enabled with a NAND Flash driver. This driver is provided as a Flash Media Driver so that the BOOTPART library can operate on NAND Flash.

For more detailed information about Eboot, please refer to the Microsoft Developer Network (MSDN).
3 NAND Flash allocation

The OS image is stored in the NAND Flash. When Eboot takes over the system, it prepares the working environment for Windows CE, and prints the optional menu on the hyperterminal. The OS image can be loaded by Eboot.

Eboot can also be stored in NAND memories and loaded by a first boot code. The first boot code is responsible for initializing the CPU, preparing the RAM, reading Eboot code from the NAND, and handing over the system management rights to Eboot.

In the OMAP5912 development platform, the NAND Flash is a 64-Mbyte small page NAND Flash, and the OS image occupies about 10 Mbytes. Boot-up of the board is performed by a customized Uboot on the NOR Flash at address 0x0. Eboot is also saved in the NOR Flash. Eboot is called by Uboot with a “go 60000” command. “0x60000” is the start address of Eboot on the NOR Flash.

Figure 3. OMAP5912 platform memory map

![OMAP5912 platform memory map](image)

Figure 4. OMAP5912 platform memory map if Eboot is saved on NAND

![OMAP5912 platform memory map if Eboot is saved on NAND](image)

Because the size of the DRAM on the OMAP5912 platform is limited, and to save RAM space, EBOOT is located in the NOR Flash and executes in place.

The master boot record (MBR) occupies the first page of the NAND Flash. It is a dedicated page for partition information used by the Windows CE partition manager. To guarantee the FAT file system works well, the first block is left on purpose for the MBR. The start address of the data region for the FAT file system should be block-aligned. The regions for the OS image can have two different cases:

1. The image region in case 1 is raw NAND blocks, and has no BINFS mounted. The OS image is saved block after block, and bad blocks are ignored. It is suggested that the image region in this case be reserved behind the data region.

2. The image region in case 2 is BINFS mounted and there are multiple XIP bin files in this region. BINFS is usually behind the MBR block, with FATFS following.
With the development of system-on-chip (SOC) technology, more and more processors have an integrated hardware NAND Flash controller and support booting from NAND Flash. The memory subsystem used for this kind of processor is usually composed of a NAND Flash plus a SDRAM. A portion of NAND is for code storage and the rest is for data storage.

Direct booting from the NAND is usually implemented as follows:

1. After power-on, a small piece of code is automatically loaded by the CPU into the internal RAM from the first several pages of NAND Flash.
2. This piece of code initializes the platform, prepares the SDRAM, and loads more code from the NAND to the SDRAM.
3. The code in the SDRAM takes over the system.

If Windows CE works on this kind of CPU, a small piece of code called “bootstrap code”, is saved in the first several pages of the NAND Flash. Bootstrap code is loaded automatically to execute in the internal RAM. Bootstrap code copies Eboot code from the NAND to the SDRAM and hands the system over to Eboot. Eboot prepares the environment for Windows CE and for loading the OS image from the NAND to the SDRAM.
4  Eboot functionalities

This section explains how to run Eboot on OMAP5912 or on a customized platform. The functionalities of Eboot are also described.

1. Windows CE Platform Builder and the hyperterminal should be opened on the development PC.

2. The development platform should be connected to the PC through a serial cable and the board should be powered on. The UBOOT command prompt “OMAP5912 OSK #” appears on the hyperterminal window. The user should type “go 60000” after the command prompt. (see Figure 5).

3. The Eboot window appears on the hyperterminal (see Figure 6).

4. Eboot has 1~7 basic configuration options and B~W functionalities. Table 1 describes how to use Eboot.

Figure 5.  UBOOT command prompt

![UBOOT command prompt image]
Table 1.  Eboot options and functions

<table>
<thead>
<tr>
<th>Option/function</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 1</td>
<td>Sets the IP address and subnet mask on the development board.</td>
<td>Uses the same class-C IP address and net mask as the host PC.</td>
</tr>
<tr>
<td>Option 2</td>
<td>Delays [how many] seconds before running the existing OS.</td>
<td>Default is 15, or user-assigned value.</td>
</tr>
<tr>
<td>Option 3</td>
<td>Shows the DHCP or static IP address.</td>
<td>Static IP address recommended if DHCP server is not available on LAN.</td>
</tr>
<tr>
<td>Option 4</td>
<td>Resets all options to the default value.</td>
<td>-</td>
</tr>
<tr>
<td>Option 5</td>
<td>Starts the existing OS image or downloads from Platform Builder.</td>
<td>Enabled: LAUNCH EXISTING  Disabled: DOWNLOAD NEW</td>
</tr>
<tr>
<td>Option 6</td>
<td>The downloaded image is written to the NAND Flash, disabled or enabled.</td>
<td>If option 5 is “LAUNCH EXISTING” Option 6 must be “DISABLED”.</td>
</tr>
<tr>
<td>Option 7</td>
<td>Shows the MAC address of the development board Ethernet MAC controller.</td>
<td>Changes the MAC address of the Ethernet MAC controller.</td>
</tr>
<tr>
<td>Function B</td>
<td>Creates a BINFS partition and a FATFS partition behind the MBR block.</td>
<td>This function is customized by the Eboot developer and used only for development demos.</td>
</tr>
<tr>
<td>Function C</td>
<td>Creates only a FATFS partition.</td>
<td>This function is used when the OS image is saved on the raw NAND.</td>
</tr>
<tr>
<td>Function D</td>
<td>Connects to Platform Builder</td>
<td>Sends BOOTME packet to Platform Builder and waits for Platform Builder response. If connected, the OS image is downloaded from the host PC.</td>
</tr>
</tbody>
</table>
Table 1. Eboot options and functions (continued)

<table>
<thead>
<tr>
<th>Option/function</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function F</td>
<td>Erases all the valid NAND blocks and create the MBR.</td>
<td>This function is implemented with the BOOTPART function</td>
</tr>
<tr>
<td>Function L</td>
<td>Manually launches the OS image in the raw NAND blocks.</td>
<td>This function is used for development demos. The OS image must exist on the NAND, otherwise Eboot falls into an infinite loop.</td>
</tr>
<tr>
<td>Function M</td>
<td>Manually launches multiple XIP OS images from BINFS partition.</td>
<td>This function is used for development demos. The OS image must exist on the NAND, otherwise Eboot falls into an infinite loop.</td>
</tr>
<tr>
<td>Function R</td>
<td>Prints the current Eboot configurations to the hyperterminal window.</td>
<td>This function is used for development demos.</td>
</tr>
<tr>
<td>Function W</td>
<td>Writes the changed option setting to the board.</td>
<td>This function is used for development demos.</td>
</tr>
</tbody>
</table>

Note: Eboot is a tool in the development phase. In the final products, it will have one function: to load the OS image and run. The functionalities listed in Table 1 are used to demonstrate how to store the OS image on memories. These functions can be customized by the developer.
5 Storing single binary OS images on a raw NAND

The OS image on a NAND Flash can not execute in place and must be loaded into the RAM to run. This means the environment variable IMG_FLASH cannot be set to "1". In the Platform Settings window, the option “Enable Image for Flash” should not be selected. This option enables the OS image to execute in place.

![Platform settings window](image)

The OS image generated using these options is called RAMIMAGE. The image can only execute if it is loaded into the RAM. RAMIMAGE is also used during the development process. During the development phase, the RAMIMAGE “nk.bin” is usually downloaded to the RAM on the target platform from Platform Builder using Eboot.

Before programming the Windows CE RAMIMAGE file to reserved blocks of NAND Flash, Eboot must mark these blocks BLOCK_STATUS_RESERVED and BLOCK_STATUS_BAD. BLOCK_STATUS_RESERVED is used to distinguish normal bad blocks from reserved bad blocks. Deliberately marking the reserved blocks as "bad" means the partition manager driver does not count these blocks as available for data storage partition. This is a trick used in the Eboot of Window CE 4.2/5.0.

Theoretically, the location of reserved blocks can be arbitrary. However, to make the data area continuous and easy to manage, these blocks are usually reserved in the back part of the NAND Flash. If Eboot is saved on the NAND Flash, the blocks accommodating the Eboot code have the same block status as those for the OS image.

Eboot uses two functions to program and read RAMIMAGE to/from the NAND. The two RAMIMAGE operation functions are called WriteRamImageToBootMedia() and ReadRamImageFromBootMedia().

WriteRamImageToBootMedia is called after the RAMIMAGE is downloaded into a RAM cache in the target board if option 5 is “DOWNLOAD NEW” and option 6 is “ENABLED”.

ReadRamImageFromBootMedia is called by choosing the function “L” if option 5 is “LAUNCH EXISTING”. This function reads from the same address where WriteRamImageToBootMedia programs RAMIMAGE.
6 Storing a multiple region OS image on BINFS

Since Windows CE version 4.2, a multiple region OS image can be built with Platform Builder. A multiple-region OS image contains several BIN files such as Nk.bin, Drivers.bin, and App.bin. The Nk.bin contains the OS kernel and critical components for system running. This bin file is loaded to the RAM and never dumped. The Drivers.bin contains device drivers. This bin file is loaded into RAM only if the driver in it is required. The App.bin usually contains some user application programs. It is loaded into the RAM when one program inside is called and dumped when all the applications inside it end.

A multiple region OS image must be saved in the BINFS partition mounted on a storage media. This kind of OS image has the following advantages:

- Independent updates to a single bin file because the modified file only influences its own bin file.
- Saves more RAM space for key applications because only the required bin files are loaded into the RAM.
- Less power consumption because less RAM is used.

Building a multiple-region OS image is more complex than building a single RAM image. The OS configuration files, config.bib and platform.bib should be changed according to the guidance of the MSDN.

In the Platform Settings window of Platform Builder (PB), the IMGMULTIBIN environment variable must be defined as “1”. This environment variable makes the build system of PB build bin files based on the definitions in config.bib and platform.bib.
The output of a successful multiple-XIP MAKEIMG process produces the following files:

- A .bin file for every XIP region.
- A single .bin file containing all XIP regions, named Xip.bin.
- A .bin file for the XIP chain, named Chain.bin.
- A .nb0 file, if ROMSTART, ROMWIDTH, or ROMSIZE are set in Config.bin. The .nb0 file is a layout of all the .bin files, including Chain.bin, as they should appear in the ROM.

Eboot also provides two functions to program/read the multiple region OS image: WriteRegionsToBootMedia() and ReadKernelRegionFromBootMedia().

Before programming the multiple region OS image, the NAND Flash must be formatted with Eboot. A MBR record is created on the NAND Flash. It is also suggested to create a BINFS on the NAND Flash. Eboot, implemented for OMAP5912, provides such a utility “B”, which can create a BINFS and a FATFS on the NAND. WriteRegionsToBootMedia checks if BINFS is created on the NAND. If no BINFS exists, this function creates a BINFS following the MBR block, and a FATFS on the remaining available blocks. These two partitions are updated into the MBR, so that after Windows CE starts working onboard the two partitions can be identified with Storage Manager in the control panel. ReadKernelRegionFromBootMedia reads the kernel region, namely nk.bin, from the BINFS partition by choosing the function “M” if option 5 is “LAUNCH EXISTING”. 
Summary

This application note describes how to use Eboot to store a Windows CE OS image into NAND Flash media. For RAMIMAGE, the nk.bin file is usually saved in the reserved raw NAND blocks, while the multiple region image is normally stored on a BINFS partition created by Eboot. It is recommended to refer to the MSDN web site for related and further detailed information as this document only gives a brief outline of how to implement the NAND Flash as the Windows CE image storage medium.
8 References

- Microsoft MSDN and Newsgroup
9 Revision history

Table 2. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Feb-2008</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale. Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEvere PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS “AUTOMOTIVE GRADE” MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2008 STMicroelectronics - All rights reserved

STMicroelectronics group of companies
Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com

19/19