AMD

Application Note

Windows® CE Persistent Registry Storage using DiskOnChip®

NOVEMBER-2001

LIMITED WARRANTY

(a) AMD warrants that the Licensed Software — **prior to modification and adaptation by Licensee —** will conform to the documentation provided by AMD. AMD does **not** warrant that the Licensed Software will meet the needs of the Licensee or of any particular customer of Licensee, nor does it make any representations whatsoever about Licensed Software that has been modified or adapted by Licensee.

(b) Subsection (a) above sets forth Licensee's sole and exclusive remedies with regard to the Licensed Software.

AMD MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE LICENSED SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO OTHER WARRANTIES WITH RESPECT TO THE LICENSED SOFTWARE ARISING FROM ANY COURSE OF DEALING, USAGE, OR TRADE OR OTHERWISE.

IN NO EVENT SHALL AMD BE LIABLE TO LICENSEE FOR LOST PROFITS OR OTHER INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, WHETHER UNDER THIS AGREEMENT, IN TORT OR OTHERWISE.

(c) Licensee shall not make any promise, representation, warranty or guaranty on behalf of AMD with respect to the Licensed Software except as expressly set forth herein.

Note: The Licensed Software is not warranted to operate without failure. Accordingly, in any use of the Licensed Software in life support systems or other applications where failure could cause injury or loss of life, the Licensed Software should only be incorporated in systems designed with appropriate and sufficient redundancy or back-up features.

Contents

1. Introduction

The ability to save modifications of the Windows CE Registry Entries and retain the modifications on subsequent boot-up of the target platform is a feature much desired by platform developers. Unfortunately Windows CE saves the Registry Entries in RAM, which is volatile, and upon subsequent boot the updates are lost. Starting from Windows CE 2.11 Persistent Registry Entries were introduced, which allows the developer to save modifications to the Windows CE Registry Entries on a non-volatile media, such as the DiskOnChip.

2. Document Organization

This Application Note discusses the Windows CE Registry Entries and the various scenarios to store them on the DiskOnChip. It is assumed that the reader has basic familiarity with the Windows CE Embedded Tool Kit or OEM Adaptation Kit. A functional copy of either of these is required.

Following is an overview of the sections in this user manual:

- Chapter 3: A brief discussion on the Windows CE Registry Entries.
- Chapter 4: A description of two possible scenarios to store Persistent Registry Entries on the DiskOnChip.
- Chapter 5: Describes the Backup Registry Mechanism
- Chapter 6: A step by step implementation of a Single-boot solution – special implementation that enables to save the Windows CE Persistent Registry Entries (PRE) on the DiskOnChip Binary partition.
- Chapter 7: Explains how to format the DiskOnChip in order to create on it a boot partition which will store the PRE.
- Chapter 8: Additional information.

3. Windows CE Registry Entries Definition

Windows CE defines two types of Registry Entries:

- **Default Registry Entries**
- **Persistent Registry Entries**

Windows CE always works with the Registry Entries that are currently RAM-based.

The Registry Entries that are part of the `NK.BIN` image, and are copied to the RAM during boot time are called the **Default Registry Entries (DRE)**.

Note: Updates to the Registry Entries will only influence the RAM-based copy and not the copy that is located inside NK.BIN. This implies that after each power shutdown any changes to the Registry Entries are lost.

Registry Entries that are stored separately from the `NK.BIN` are known as **Persistent Registry Entries (PRE)**. The PRE can be copied to the RAM automatically (during boot time) to overwrite the Default Registry Entries in RAM. Since these Registry Entries are stored outside `NK.BIN`, it is possible to save the changes that are made to the Registry Entries. Using the Persistent Registry Entries instead of the Default Registry Entries allows the user to make changes to the Entries that will be reflected in the next boot (such as Touch Panel calibration data, IP info etc).

Since Windows CE always uses the Default Registry Entries upon boot, a special scheme is required to overwrite them with the Persistent Registry Entries at boot time. Depending on where the Persistent Registry Entries are stored on the DiskOnChip (FAT partition or Binary partition), a single-boot solution or double-boot solution is required.

In the following paragraphs, both solutions will be described in further detail.

4. Boot Solutions

There are two methods to load and save the Persistent Registry Entries:

1. **Double-boot solution:** Save PRE as a file on the FAT partition of the DiskOnChip by using the function `RegCopyFile()` to save registry information and the function `RegRestoreFile()` to load registry information. This solution can only be used in platforms that support warm boot.
2. **Single-boot solution:** Save PRE in a special Binary (hidden) partition of the DiskOnChip. Implement the function `readRegistryFromOEM()` to load the registry information from the Binary partition and the function `writeRegistryToOEM()` to save the registry information in the Binary partition. The functions `readRegistryFromOEM()`and `writeRegistryToOEM()` are implemented by using API provided by AMD's Boot Developer Kit (BDK)[1] to access the Binary partition on the DiskOnChip.

[1] The BDK can be obtained from AMD for free and under a license agreement. The BDK includes source code examples and an Application note.

4.1 Double-Boot Solution

Windows CE provides two functions to save and load the Persistent Registry Entries from the FAT partition of the DiskOnChip. `RegCopyFile()` is used to save the Persistent Registry Entries on the FAT partition, while `RegRestoreFile()` is used to load the Persistent Registry Entries into RAM[2]. Both these functions access the FAT partition of the DiskOnChip through the TrueFFS driver.

Storing the Persistent Registry Entries in the FAT partition of the DiskOnChip results in a double-boot of the system. Upon the first (cold) boot the Default Registry Entries are copied into RAM. At the end of the boot process, `RegRestoreFile()` is called in order to copy the Persistent Registry Entries into RAM and overwrite the Default Registry Entries. A warm boot is required to make the new RAM-based Registry Entries (i.e. Persistent Registry Entries) the active one.

During operation, a call to the `RegCopyFile()`saves the changes to the Persistent Registry Entries on the FAT partition of the DiskOnChip.

Note: Saving PRE to the FAT area is the responsibility of Windows CE and is described in this document for the designer's attention. For further help on this issue, refer to the Platform Builder help files.

4.2 Single-Boot Solution

Registry Entries hold critical data for the operating system and it is therefore preferable to store them in an area that is protected from user access. DiskOnChip provides the possibility of storing the Windows CE Persistent Registry Entries in a special Binary (hidden) partition.

4.2.1 Principal of Operation

Storing the Persistent Registry Entries in the Binary partition of the DiskOnChip results in a single-boot of the system.

Windows CE exposes two global pointers named `pWriteRegistryToOEM`, and `pReadRegistryFromOEM`. The developer is required to implement the functions `writeRegistryToOEM()` and `readRegistryFromOEM()` and initialize the pointers `pWriteRegistryToOEM` and `pReadRegistryFromOEM` to point to these two functions. The functions `writeRegistryToOEM()` and `readRegistryFromOEM()` in the OEM adaptation layer use functions provided by AMD's Boot Developer Kit (BDK) to access the Persistent Registry Entries in the Binary partition[3].

During registry initialization (one of the first steps taken at boot time), Windows CE checks the pointer `pReadRegistryFromOEM`. If the pointer contains a **valid** address, the kernel will call the function `readRegistryFromOEM()` and the Persistent Registry Entries will be copied from the Binary

[2] The APIs RegCopyFile() and RegRestoreFile() are documented in the Windows CE ETK or SDK.

[3] See Application-Note "DiskOnChip Boot Developer Kit"

partition on the DiskOnChip into the RAM. This will overwrite the Default Registry Entries that were already located in RAM.

Note: If the media is not found or the data on the DiskOnChip is invalid, the Default Registry stored in ROM will be loaded to RAM.

When saving a modification made to the registry while the target platform is operating, a series of function calls must be made through a Windows CE application to start the write process. As a result, the kernel calls the function `writeRegistryToOEM()` to write the data to the DiskOnChip. Modifications to the registry stored in RAM may be done as often as necessary, and it is up to the user to decide when they should be stored in the PRE. Microsoft recommends that write access to the Persistent Registry Entries storage is done after a group of changes have been made to the registry to avoid degradation of OS performance. To read the new registry settings from persistent registry storage, a system reboot is required.

In order to initiate the PRE saving process the function `RegFlushKey()`[4] has to be called to update the Persistent Registry Entries on the DiskOnChip. The function `RegFlushKey()` calls the function `writeRegistryToOEM()`.

Note: A source code file `Registry.C` containing an implementation example of the `readRegistryFromOEM()` and `writeRegistryToOEM()` APIs can be obtained from AMD.

Note: There are two versions of the `Registry.C` source file. One version based on the BDKv121 source code, and another based on the BDKv122 source code. The reason is that the BDKv121 and BDKv122 differ in their API functions.

4.2.2 How Does the Binary Partition on the DiskOnChip Relate to the Registry?

The Binary partition contains binary data in the format below. In this example, the segment address of the DiskOnChip is assumed to be `D000h`.

`D000` (DiskOnChip address)

DWORD Registry Magic	DWORD Registry Size	DWORD Checksum	BYTES (of Registry Size) Registry Data

When the registry is read, the Registry Magic is checked to verify whether or not data has been properly written to the Binary partition. The registry size is then checked against the defined constant, `REGISTRY_IMG_LENGTH`; the registry size provided by the OS kernel cannot exceed this constant. Finally, a checksum is calculated for the registry data and checked against the checksum written as the third DWORD in the Binary partition. If everything is correct, the registry data is read and stored in RAM. If things are not correct, the OS kernel extracts the registry stored in ROM and loads it into RAM.

[4] RegFlushKey (hKey) – Call the `RegFlushKey` with the predefined handle value `HKEY_LOCAL_MACHINE`.

5. Backup Registry

To prevent corruption of the registry (this can be caused by a write failure during a power outage), the Binary partition contains two copies of the PRE registry. This means that the registry is copied with a counter that indicates which of the copies is the updated one.

This feature is **optional** and can be set by the changing constants as detailed in 6.2.3.

5.1 Backup Registry Mechanism.

The Backup Registry mechanism is similar to a double buffering mechanism. When updating the registry, the new registry is copied to the other area of the last updated copy and the counter is then incremented by one. If there is no valid registry, the first area is updated and the counter value is set to 0. The registry header, located in the beginning of every copy, defines whether the specific copy is the updated one. Table 1 details the Registry Header Fields:

Table 1: Registry Header

Field Name	**Description**	**Length**
Magic Number	A Signature to identify the copy	4 BYTES
Copy data length	The length in bytes of the copy	4 BYTES
Checksum	The checksum of data bytes (not including the header)	4 BYTES
Counter	The copy counter	4 BYTES

A registry copy is detected as valid by checking the copy checksum. The updated registry copy is determined by the counter value located in every registry header (the largest one is selected as the updated copy).

During boot time both copies must be read to determine which of the copies is the updated one. If there is no valid copy (checksum failure in both copies) the kernel loads the default registry. Where there is one valid copy (checksum failure in one of the copies), the valid copy determines the counter value and is selected as the updated one. When both copies are valid (checksum success for both copies), the counter in every registry header determines which of the copies is the updated registry copy.

6. Step-by-Step Implementation of Single Boot Solution

To store the Persistent Registry Entry in the Binary partition of the DiskOnChip, thereby implementing a Single Boot Solution, perform the following steps:

6.1 Format the DiskOnChip

To create a Binary (boot) partition on the DiskOnChip, format the DiskOnChip using AMD's `DFORMAT` utility. Note that the size of the created partition should be at least the size of the PRE you want to save, and should be created with a boot partition signature. The signature must be the same as the signature being used in the code (BDK code) that implements the functions `writeRegistryToOEM()`and `readRegistryFromOEM().`

For formatting the DiskOnChip under Windows CE, refer to section 7. Formatting the DiskOnChip Under Windows CE and in Production. For formatting the DiskOnChip in DOS environment, refer to application note "DiskOnChip Boot Software Development Kit".

6.2 Modify the Required Source Files

There are three source files provided with the BDK source code that need to be customized to your specific requirements. These files are added to the CE OAL to provide the functions necessary to save and restore the PRE on the Binary partition of the DiskOnChip.

6.2.1 DOC_BDK.H File

This is the header file for the BDK source file `DOC_BDK.C`. It contains compile-time definitions, type declarations, and BDK function prototypes. The compile-time definitions should be set to correspond with the target platform.

`#define WRITE_IMAGE`	// when uncommented, this enables write access to the Binary partition
`#define FAR_LEVEL 0`	// 0 – (recommended) use flat memory model or have no far pointers
	// 1 – only the DiskOnChip window may be far
	// 2 – only the DiskOnChip window and RAM window may be far
	// 3 – DiskOnChip window, RAM window and pointer(s) transferred to the entry-point function may be far
`#define DOC_ACCESS_TYPE 8`	// supported bus widths: 8 (default), 16, 32

The constants defined below determine the specific address or address range to search for the DiskOnChip. If constants have the same address value, no search is performed and the kernel will assume the device resides at that specific address. If the addresses are different, the kernel will scan the address range to search for the device. If a range is specified, the low address must have a lower value than the high address. The high address should be the last address that can be used as an ending address for the DiskOnChip window.

`#define DOC_LOW_ADDRESS 0xA00d0000L`	// start address of search
`#define DOC_HIGH_ADDRESS 0xA00d0000L`	// end address of search

6.2.2 DOC_BDK.C File

This file is the source file that provides functions to access the Binary partition. Specific modifications to the original file provided by AMD are listed below.

The following code block, which exists in two locations in this file, should be commented out to eliminate the “cannot find include file <dos.h>” compiler warning.

```
/* #ifdef DOS */
/* #include <dos.h> */
/* #endif */ /* DOS */
```

6.2.3 Registry.C File

This file contains the interface between the kernel and the BDK functions. The kernel functions `readRegistryFromOEM()` and `writeRegistryToOEM()` are defined in this file. Specific modifications to the original file provided by AMD are listed below.

The following compile-time definitions may be commented out if output of debug messages and/or calculation of a checksum are not desired. However, it is highly recommended that a checksum be written to the Binary partition, as this verifies that the registry data written to the partition is valid.

```
#define REG_DEBUG
#define CHECK_SUM
```

The following defined constant **must** correspond with the actual formatted size of the Binary partition. In the below example, the Binary partition was formatted for 256K bytes. So the registry image length must be 0x40000, the hex value of 256K.

Warning: Formatting the DiskOnChip, and setting the value for the defined constant `REGISTRY_IMG_LENGTH`, are two independent manual tasks. AMD recommends that the value of the constant correspond with the physical size of the Binary Partition. The effects are unknown if these two do not correspond.

```
#define REGISTRY_IMG_LENGTH 0x40000         // 256 KBytes
```

Make sure the following `#define` fits the signature you used while formatting and creating the Binary partition:

```
#define BOOT_IMG_SIGNATURE "BIPO"
```

6.2.4 Changes to the File that Contains the OEMinit() Function

Changes to the kernel are required to notify the operating system that the APIs `readRegistryFromOEM()` and `writeRegistryToOEM()` exist to handle the saving and restoring of registry data from the DiskOnChip Persistent Registry storage area. This file, which typically resides in the kernel\hal subdirectory of the platform subdirectory[5], requires the following modifications:

```
extern BOOL (*pWriteRegistryToOEM)( DWORD dwFlags, LPBYTE lpData, DWORD cbData);
extern DWORD (*pReadRegistryFromOEM)( DWORD dwFlags, LPBYTE lpData, DWORD cbData);
extern DWORD readRegistryFromOEM (DWORD dwFlags, LPBYTE lpBuf, DWORD len);
extern BOOL writeRegistryToOEM(DWORD dwFlags, LPBYTE lpBuf, DWORD dwLen);
OEMInit()
…
{
pWriteRegistryToOEM = writeRegistryToOEM;
pReadRegistryFromOEM = readRegistryFromOEM;
…
}
```

6.3 Compile the PRE Saving Code

Add `DOC_BDK.H, DOC_BDK.C`, `DOC_ECC.C` and `REGISTRY.C` to the kernel\hal subdirectory of the platform directory (the same directory where the file that includes the function `OEMInit()`is located) to reflect the modifications described in the above sections. Edit the Sources file to build the .C files. For example:

```
SOURCES= \
    debug.c \
    cfwpc.c \
    …
    doc_bdk.c\
    doc_ecc.c\
    registry.c\
```

You can now compile the Windows CE image.

[5] Gor CEPC the file that contains the function `OEMinit()` is called `Cfwpc.c`.

7. Formatting the DiskOnChip Under Windows CE and in Production[6]

If the DiskOnChip is not formatted, a dialog box will pop up during the boot process. The user will be asked to confirm the formatting process. If confirmed, the DiskOnChip will be formatted and prepared for use. The boot sequence will continue seamlessly and re-boot will not be required.

In order to format the DiskOnChip manually, a Windows CE `dformat` utility is available from AMD. This utility can be incorporated in the image or can be run remotely from the host (through PPSH/CESH).

`dformat` utility usage:

```
dformat -w: DOC_base_address
  -l: Binary Partition length
  -n: Binary Partition signature. Default: BIPO
  -e: {firmware} / {!} (! = to remove firmware)
  -s: {binary partition image} / {!} (! = to remove binary
       partition)
  -p: base_address shift. Options: 0 or 8. Default: 8
  -a: Access type. Options: 8, 16 or 32. Default: 8
```

If the use of a hexadecimal number is desired, add the prefix `0x.`

Formatting the DiskOnChip during boot time is made possible by using the OSAK (OS Adaptation Kit) from AMD, which enables formatting and accessing the DiskOnChip FAT area in an OS-less environment. The OSAK[7] can be modified to fit different production environments (such as VxWorks, QNX, DOS, etc.) to enable formatting and partitioning of the DiskOnChip during production. The OSAK is supplied in source code and requires a license agreement. Writing the Windows CE image and data into the DiskOnChip Binary Partition, through files or buffers, is possible by using the BDK source code, provided by AMD. The TrueFFS driver under Windows CE makes writing files to the FAT partition possible.

Important: You must reboot your system in order to remount the DiskOnChip.

[6] For other methods of formatting DiskOnChip in production phase, refer to Application Note: Programming of the DiskOnChip TSOP.

[7] The OSAK –DiskOnChip OS Adaptation Kit – is a source code driver package available from AMD under license agreement.

8. Additional Information

Additional information about DiskOnChip, including application notes, can be found at http://www.amd.com

How to Contact Us

Internet: http://www.amd.com

AMD

One AMD Place
P.O. Box 3453
Sunnyvale, California 94088-3453
(408) 732-2400
(800) 538-8450
TWX:910-339-9280
TELEX:34-6306

Technical Support
USA & CANADA
(800) 222-9323 or
(408) 749-5703

USA & Canada & Latin America E-Mail:
Hw.support@amd.com
Portugues.tech@amd.com
Spanish.support@amd.com

ARGENTINA: 001-800-200-1111,
after tone 800-859-4478
CHILE: 800-532-853
MEXICO: 95-800-222-9323

EUROPE & UK
+44-(0)1276-803299
Fax: +44-(0)1276-803298
FRANCE: 0800-90-8621
GERMANY: 089-450-53199
ITALY: 800-877224

EUROPE E-mail: euro.tech@amd.com

FAR EAST Fax: (852) 2956-0599
JAPAN Fax: 03-3346-7848

Literature Ordering
USA & CANADA: (800) 222-9232
USA & CANADA & LATIN AMERICA E-mail:
amdlit@gomez.amd.com
EUROPE E-mail: euro.lit@amd.com
FAR EAST Fax: (852) 2956-0599
JAPAN Fax: 03-3346-9628