

Understanding CEPC Boot Sequence in Windows Embedded Compact 7

Douglas Boling
Boling Consulting Inc.

About Douglas Boling

- Independent consultant specializing in Windows Mobile and Windows Embedded Compact (Windows CE)
 - On-Site Instruction
 - Consulting and Development
- Author
 - Programming Embedded Windows CE
 - Fourth Edition

Agenda

- Storage Basics
- PC Boot Basics
- Building the Bootloader
- Extending the Bootloader

Storage

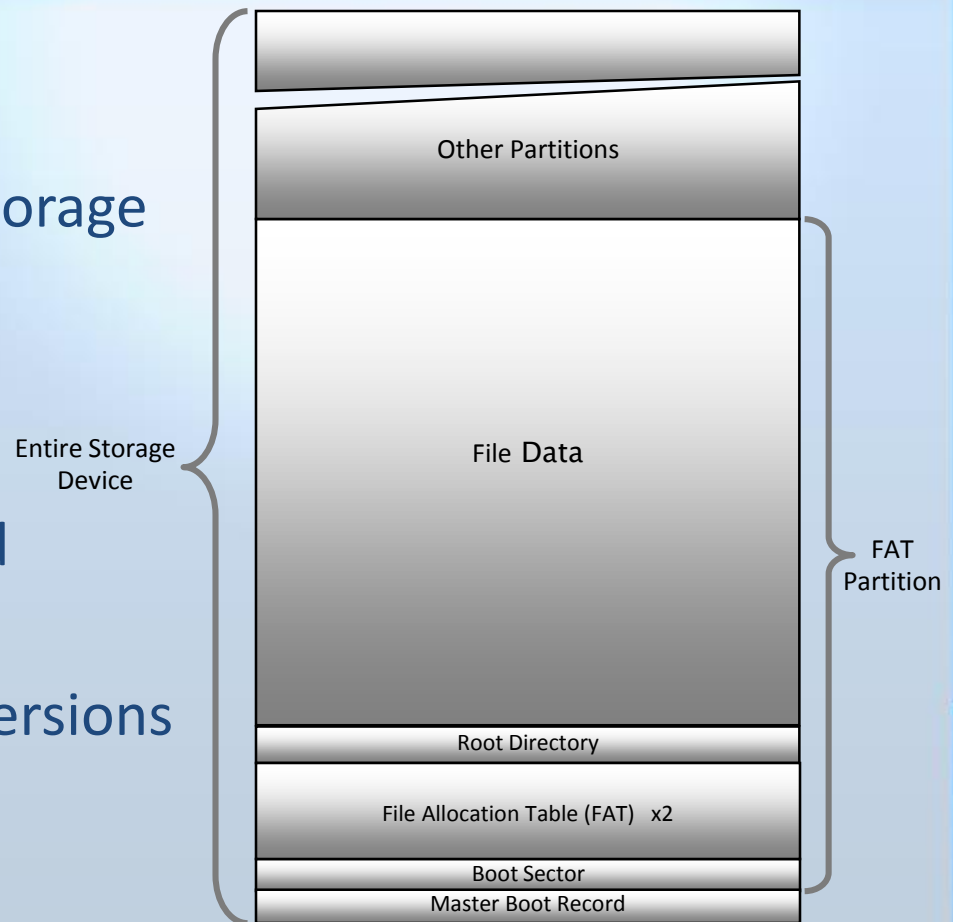
- Can be floppys, hard disks, USB sticks, SD/CF cards, Raw Flash
- Physical devices can be logically partitioned
- x86 systems need one storage device to contain operating system files
 - “Boot drive”

The Partition Table (Master Boot Record)

- Logically divides storage medium into ‘partitions’
 - Table allows 4 partitions
- Each partition is considered a “Logical Disk”
 - Partitions are each formatted to whatever file system is desired
- One partition must be marked “Active” for bootable storage
 - Almost all PCs won’t boot unless the an Active partition is found
- Windows Embedded Compact can support all 4 partitions
- Windows Desktop has strict limitations on partitions

FAT File System

- FAT == File Allocation Table
- Intergalactic standard for storage
- Fairly simple to understand
- ExFAT sparsely documented
- Boot sectors differ across versions



Desktop limitations on The Partition Table

- Windows expects 2 partitions
 - Primary
 - Extended
- Windows expects the 'extended' partition to be sub-partitioned into 2 partitions
 - Secondary, Extended
 - Each 'extended' partition can be subdivided allowing for numerous partitions
- Windows ignores other partitions

The Boot Sector

- First sector on the storage device
 - 512 bytes by defacto PC standard
- BIOS loads first sector into RAM (in real mode) at 0:7C00
 - Then jumps to address 0:7C00
- Bootsector searches disk for specific file and loads it into RAM
 - Name of file and search method depends on format type
 - Boot code then jumps to first byte of loaded sectors
- If code fails to find specific file, it displays message and halts

The Directory Structure

- FAT storage has at least one “root” directory
 - At a known location
 - Other directories stored as data and located by entries in their parent directories
- Directories are tables of 32 byte entries
 - Each entry can describe a file in 8.3 name format
 - For long file names, multiple entries are used

FAT Table Entries

- Large table that describes sector assignments
- Table entries are either 12, 16 or 32 bits in size
 - The very definition of FAT12/FAT16/FAT32
- Zero indicates free sector
- Non-zero number $< 0xffff\ fff0$ indicates sector used
 - Number points to next sector of file
 - Value $\geq 0xffff\ fff8$ indicates last sector of file

PC Boot Basics

- PCs come with one of two interfaces to firmware
 - BIOS – Basic Input Output System Interface
 - Historic
 - Well understood
 - Massive acceptance
 - Register based, 16 bit interface
 - (U)EFI – (Universal) Extensible Firmware Interface
 - New, Intel defined standard
 - 32 bit interface
 - Extensible
 - Intel has a UEFI loader for Windows Embedded Compact

BIOS Interface

- Interrupt based interface
 - Int 10, Int 13, Int 15
- On boot, reads first sector of primary storage system into RAM at 0:7C00
 - 1 sector == 512 bytes
 - Yes, that is a 16 bit, segmented address

BIOS Interrupts

- Interrupt 10 Video Display
- Interrupt 13 Storage I/O
- Interrupt 14 Serial Port
- Interrupt 15 System level control
- Interrupt 16 Keyboard services
- Interrupt 1A RTC and PCI Services

Calling BIOS Interrupts

- Must be called in 16 bit, Real Mode
 - Code must transition back into real mode before call
- Register calling model
 - Command in AH
 - Subcommand in AL
- Buffers must be in RAM < 0x10000 (<1Meg)

CEPC Bootloaders

- LoadCEPC
 - DOS based bootloader
 - Requires a formatted FAT12/16 partition with DOS system files
- BIOSLOADER (BLDR)
 - No need for DOS
 - Works with FAT12/16/32 and ExFAT
- WCELDR (XLDR / BLDR)
 - Rewritten loader for WEC 7
 - Allows for a larger and more functional bootloader
 - Can't mix BIOSLOADER and WCELDR components

Modifying LoadCEPC

- Need access to a 16 bit compiler
 - MSVC 1.52 available from MSDN
- Code location
 - WINCE700\platform\CEPC\src\bootloader\dos\loadcepc

Installing LoadCEPC

- Typically device is booted into DOS with system diskette
- Format permanent storage with DOS system
 - Must be a FAT12/FAT16 partition or disk
- Partition must be marked bootable
 - DOS can do this with FDISK
- CESys application can run under DOS to initialize disk
- Image files for floppy disks included in WINCE tree

BIOSLoader

- Relatively fast boot
 - No need for DOS
- Uses BIOS for system I/O
 - Works on all BIOS based x86 systems
- Supports all FAT formats 12,16,32,Ex
 - With supporting source code
- Configurable with text file on boot disk
- Available on CE R2 and later

BIOSLoader Limitations

- Code designed to communicate via serial port only
- No way to change configuration information from bootloader
- Difficult to build
 - Requires patch scripts that won't run under Win64 systems
- Code size limitations
 - Current memory layout limits code size to 32 KBytes or smaller
 - Difficult to use full error message version of loader

WCELDR

- Combination of XLDR and updated BLDR components
 - XLDR – Simple loader that loads remainder of BLDR
- Provides optional menu based interaction with user
- Can save configuration information across reboots
- Supports both serial and console (display) interaction
- New to Windows Embedded Compact

WCELDR Limitations

- No documented method for building boot sectors
- Boot sector code for FAT16 and FAT32 only
- No simple way to preconfigure bootloader settings
- Incompatible with original BLDR boot sector code
 - Don't mix boot sectors from one loader to the other
- This is not a simple boot loader
 - Lots of features means complexity

WCELDR Sequence

- Boot sector
 - Finds WCELDR in the root directory
 - Loads 68 sectors of WECLDR into RAM at address 0x1000
 - Jumps to 0x1000, entry point of XLDR component
- XLDR component
 - Switches to protected mode
 - Finds WCELDR in root directory
 - Scans file for 0xB000FF signature which is start of .bin file
 - Reads packets of .bin file and places in RAM
 - Jumps to start of new .bin image (BLDR)

WCELDR Sequence (2)

- BLDR
 - Displays prompt for menu mode vs download
 - If download, downloads from current download source (Ethernet/disk)
 - If menu, displays menu and executes command
 - Show Current Settings
 - Set Boot Device
 - Select KITL Device
 - Network Settings
 - Display Settings
 - Debug Port Settings
 - Save Settings
 - Exit and Continue

Modifying WCELDR

- Modifications would generally be made to BLDR component
 - XLDR really doesn't know much except the directory structure
- BLDR mods can be made as in any other BSP component
 - No special build steps needed
- Main modification will be to add available Ethernet device

Adding an Ethernet Lib to WCELDR

- Modify Init.C in WINCE700\platform\CEPC\src\boot\bldr
 - Add new BootEdbgDriver_t structure for your Ethernet controller

```
static const BootEdbgDriver_t s_Ne2000 =  
{  
    NULL,                // InitDMABuffer  
    NE2000Init,          // InitNICControler  
    NULL,                // DeInitNICControler  
    NE2000SendFrame,    // SendFrame  
    NE2000GetFrame,     // GetFrame  
    NULL                 // Filter  
};
```

- Same Entry points as used in earlier EBOOT bootloaders

Modifying WCELD R (2)

- Modify init.c to include structure in “devices” table

```
static const Device_t s_devices[] = {  
    { L"Boot Disk", DeviceTypeStore, (enum_t)IfcTypeUndefined, 0, NULL },  
    { L"RTL8139", DeviceTypeEdbg, IfcTypePci, 0x12111113, &s_Rtl8139 },  
    { L"RTL8139", DeviceTypeEdbg, IfcTypePci, 0x13001186, &s_Rtl8139 },  
    { L"RTL8139", DeviceTypeEdbg, IfcTypePci, 0x813910EC, &s_Rtl8139 },  
    { L"DEC21140", DeviceTypeEdbg, IfcTypePci, 0x00091011, &s_Dec21140 },  
    { L"NE2000", DeviceTypeEdbg, IfcTypePci, 0x09401050, &s_Ne2000 },  
    { L"NE2000", DeviceTypeEdbg, IfcTypePci, 0x802910EC, &s_Ne2000 }  
};
```

- Add EthDbg library for your Ethernet controller
 - Modify sources in
 - .\boot\bldr\serial
 - .\boot\bldr\console

WCELDR Construction

- Created from 3 separate files
 - XLDR.nb0
 - Binary image of XLDR
 - BLDRcfg.nb0
 - Mostly empty file 512 byte file with “BLDRCFG” as the header characters
 - Used to save configuration changes
 - BLDR.BIN
 - Actual bootloader
 - Automatically loads from configured boot device or goes to menu
 - Menu driven commands can persistently configure the loader

Top of file



Base of file

Building WCELDR

- Builds must be made from the boot directory
 - NOT from the BLDR directory
 - This allows the final build step in the XLDR directory to build the final WCELDR image
 - Final step combines XLDR.nb0, BLDRcfg.nb0 and BLDR.BIN
- Two resulting files
 - WCELDRC – The console version
 - WCELDRS – The serial version
- Rename the file you want to WCELDR
 - Install with `WINCE700\platform\CEPC\src\boot\tools\bin\i386\CeSys`

Installing BLDR/WCELDLDR On A Disk

- Platform Builder has new Windows-based CeSys
 - Can be used to update boot sector and transfer WCELDLDR
- Need to add nk.bin *after* using CeSys
- No documented method of preconfiguring WCELDLDR
 - BLDRCfg could be pre-written but no tool currently exists

Summary

- Lots of choices for boot loading
 - Choices can lead to confusion
- Don't mix the bootloader solutions
- Pick the solution that works for you
- Modifications to add Ethernet libraries fairly simple

Questions...

Doug Boling

Boling Consulting Inc.

www.bolingconsulting.com

[dboling @ bolingconsulting.com](mailto:dboling@bolingconsulting.com)

Microsoft[®]

Building A Boot Sector

- Need access to an older assembler
 - MASM 6 available from MSDN
 - Take care to use the proper linker
- Understand GetBSect.scr
 - A script for DEBUG.EXE (Won't run on Win64 systems)
 - Trims the length of image to code
- You will need a way to install the code on a boot disk
 - CESys for WEC 7 has a prebuilt boot sector internally