

Windows CE

Oliver Kapaun
TU-Darmstadt

Kapaun@rbg.informatik.tu-darmstadt.de

Abstract

In mobilen Systemen werden andere Anforderungen an die Hard- und Software gestellt als in stationären. Windows CE ist ein Betriebssystem für derartige mobile Systeme. Diese Systeme werden von einem Embedded System Developer entworfen und zusammengestellt. Seine Aufgabe ist es, eine Umgebung bereitzustellen, die den Anforderungen der Applikation genügt. Dafür müssen Hard- und Software aufeinander abgestimmt und Kostenaspekte mit einbezogen werden. Das Aufgabengebiet des Embedded System Developers ist Inhalt dieser Arbeit. Dieser Entwickler stellt zuerst das System zusammen und untersucht dann ob verschiedene Kenngrößen mit den Anforderungen der Applikation verträglich sind. Der erste Teil dieser Arbeit behandelt die Konfiguration des Systems und klärt grundlegende Sachverhalte, während im zweiten Teil, dem Anhang, ein vertiefter Einblick in Treiberkonzepte und den Echtzeitbetrieb dieser Systeme gegeben wird. Zum Abschluß wird noch ein Überblick zur aktuellen Marktsituation von Windows CE gegeben.

1. Einführung

Die Informations- und Kommunikationstechnologie (IuK) ist weltweit ein Wachstumssektor. Durch die Vernetzung verschiedener Rechnerwelten sind in den vergangenen Jahren die Anwendungsmöglichkeiten der IuK drastisch gestiegen. Während die Vernetzung der stationären Systeme augenblicklich noch nicht abgeschlossen ist, ergeben sich weitere Möglichkeiten durch die Vernetzung mobiler Systeme. Dadurch verändern sich die Anforderungen, die bislang an Software und Hardware gestellt wurden. Windows CE ist ein Betriebssystem, das besonders für die Erfordernisse sogenannter eingebetteter Systeme (*embedded Systems*) entwickelt wurde. Diese Systeme können sowohl mobil als auch stationär eingesetzt werden, ganz nach dem Motto des „mobile Computing“, die Vernetzung der Welt mittels unterschiedlicher Rechnerarchitekturen. Eingebettete Systeme sind Systeme, die nicht zwingend wie herkömmliche Desktop PCs im Büro oder an anderen Orten stationär betrieben werden müssen. Der Laptop ist so gesehen mit eingebetteten Systemen verwandt. Allerdings sind letztere oft für einen bestimmten Zweck

entwickelt worden. Wenn Laptops allen gestellten Anforderungen gerecht würden, also beliebig klein, robust oder preiswert wären, erübrigte sich die Entwicklung dieser speziellen Systeme. Folgende zusätzliche Anforderungen gelten für ein eingebettetes System: Es kann strengen Bedingungen hinsichtlich Speicher-verbrauch und Hardwareressourcen unterliegen. Es kann, muß aber keine graphische Benutzerschnittstelle bieten und es kann von zeitkritischen Bedingungen abhängig sein. Es ist möglich, daß es für nur eine einzige Anwendung entwickelt worden ist oder daß es Verwendung auf einer einzigen Plattform findet. Eine genaue Trennung zwischen dem Betriebssystem und der Anwendung ist daher oft nicht exakt möglich.[9]

Der Bedarf nach Vereinfachung in eingebetteten Systemen ist oft auf die Kosten zurückzuführen, die für ein bestimmtes System entstehen dürfen. Aber auch die Zuverlässigkeit des Systems, die äußeren Abmessungen und Einschränkungen bei der Stromversorgung können eine bedeutende Rolle spielen.

Das Betriebssystem Windows CE (Win CE) wurde ursprünglich für den Einsatz in eingebetteten Systemen entwickelt. Diese Systeme werden für eine Vielzahl von Aufgaben eingesetzt und kommen an ganz unterschiedlichen Orten zum Einsatz. Eine flexible Anpassung des Betriebssystems an unterschiedlichste Gegebenheiten ist daher notwendig. Oft wird in diesem Zusammenhang der Begriff Embedded OS verwendet. Allerdings besitzen auch andere moderne Betriebssysteme beispielsweise Linux Eigenschaften, die sie für den Einsatz unter derartigen Bedingungen qualifizieren.

In eingebetteten Systemen finden verschiedenste Peripheriegeräte Einsatz, die oftmals nur für einen bestimmten Verwendungszweck entwickelt wurden. Heute gibt es bereits eine Vielzahl von eingebetteten Win CE Lösungen, die sowohl Konsumgüterbereich als auch im Industriebereich vorzufinden sind.

Die Anpassungsfähigkeit des Betriebssystems wird in Win CE durch Komponenten erreicht. Die Komponenten müssen hinsichtlich Leistung und Effizienz besonderen Anforderungen genügen. Obergrenzen für Verzögerungen müssen einschätzbar und der Betriebssystemkern muß skalierbar sein. Der sogenannte *Plattform Builder* ist das entsprechende Werkzeug, um individuelle Win CE Lösungen zu erstellen.

Win CE wurde zudem so entwickelt, daß es mit verschiedenen Prozessorfamilien zusammenarbeitet. Um die leichte Portierbarkeit der bereits vorhandenen Applikationen der Windows Desktops zu erreichen, wird eine Untermenge von ungefähr 1500 Win32-API-Schnittstellen in Win CE angeboten. Darüber hinaus finden sich Teilmengen der folgenden weiteren Schnittstellen, welche bereits bei den Desktopvarianten vorhanden sind:

- Component Object Model (COM)
- Microsoft ActiveX controls
- Microsoft Active Template Library (ATL)

COM (*Component Object Model*) und *ActiveX Controls* sind modulare Programmbibliotheken, welche hinsichtlich Größe und Geschwindigkeit optimiert sind. Microsoft Windows CE *Toolkit for Visual Basic 5.0* und Microsoft Windows CE *Toolkit for Visual C++ 5.0* sind die entsprechenden Tools, welche die oben genannten Bibliotheken verwenden. Beide werden zur Erstellung von Applikationen benötigt.

Zusätzlich wurden die sogenannten *Microsoft Foundation Classes* (MFC) entwickelt (siehe Abbildung 1). Sie machen Gebrauch von der Win 32 API, besitzen jedoch kompaktere Schnittstellen als diese, was sie für den Einsatz in eingebetteten Systemen besonders qualifiziert. [10]

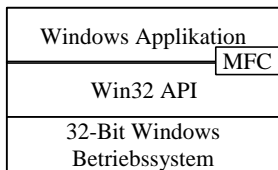


Abbildung 1: Die MFC bietet eine kompakte Schnittstelle für Applikationen

Windows CE (Win CE) fand erstmals 1996 Einsatz in Handheld PCs. Heute umfaßt die Produktpalette von Microsoft Versionen für Handheld PCs, Palm Sized PCs und Auto PCs.

Durch die besonderen Systemanforderungen unterscheidet sich die Applikationsentwicklung auf eingebetteten Systemen wesentlich von der auf herkömmlichen Desktop PCs. Die Applikation stellt bestimmte Anforderungen an die Hardware und das Betriebssystem. Der *Embedded System Developer* übernimmt die Auswahl des Betriebssystems und der Hardware und ist für die Sicherstellung der benötigten Anforderungen, welche durch die Applikation vorgegeben werden, verantwortlich. Die folgenden Abschnitte beschäftigen sich mit dem Aufgabengebiet des *Embedded System Developers*, das sich mit nachstehenden Sachverhalten auseinandersetzt.

- Konfiguration der Softwarekomponenten
- Konfiguration der Hardware, insbesondere des Speichers

- Wahl und Integration verschiedener Treibertypen zur Steuerung der Hardware
- Konfiguration des Systems im Hinblick auf Echtzeitbedingungen

Abschnitt D faßt wesentliche Inhalte der Arbeit zusammen und stellt die momentane Marktsituation von Win CE dar. Zusätzlich befindet sich noch ein Glossar im Anhang indem sich alle verwendeten Abkürzungen wiederfinden.

In folgenden Abschnitt wird zunächst ein kurzer Überblick über die Win CE Architektur gegeben, bevor auf weitere Einzelheiten eingegangen wird.

2. Windows CE Architektur und Komponenten

In diesem Abschnitt sollen die wichtigsten Bestandteile der Win CE Architektur dargestellt werden. Diese Architektur wird im Laufe der Arbeit wieder aufgegriffen und konkretisiert.

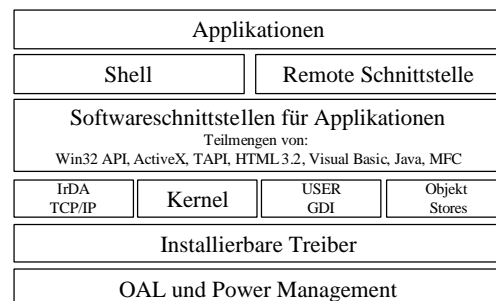


Abbildung 2: Die Architektur von Win CE

Der Betriebssystemkern unterstützt das *Standard Win32 Prozeß- und Threadmodell*. Es ist für das *Scheduling* verantwortlich und stellt Synchronisationsmechanismen, wie Mutexe usw. bereit. Der Programmcode kann direkt aus dem ROM oder anderen Bausteinen ausgeführt werden, was als *Execute In Place* (XIP) Fähigkeit bezeichnet wird.[2]

Da Win CE für verschiedene Prozessorfamilien entwickelt wurde, würde jeder zusätzlich zu unterstützende Prozessor eine zusätzliche Version großer Bestandteile des Betriebssystems erfordern. Um dieses Problem zu lösen, wurde der sogenannte *OEM Adaption Layer* (OAL) entwickelt.

Der OAL ist eine Schicht, die sich zwischen der Hardware und dem eigentlichen *Kernel* befindet. Sie stellt der darüberliegenden Schicht eine einheitliche Schnittstelle zur Verfügung. Dadurch können darüberliegende Schichten, wie das *Kernel*, unverändert auf andere Plattformen übertragen werden.

Das *Kernel* und der OAL befinden sich nach dem Booten des Betriebssystems im Hauptspeicher. Üblicherweise werden in anderen Betriebssystemen die Applikationen ebenfalls in den Hauptspeicher geladen und dann ausgeführt. In eingebetteten Systemen ist der Einsatz von Festplatten in Kombination mit D-RAM Bausteinen nicht

immer üblich, statt dessen verwendet man z.B. ROM Bausteine, um das Betriebssystem für das System zugreifbar zu machen. Dieser sogenannte *Flash Speicher* kann langsam sein und kann dann wie eine Festplatte genutzt werden, oder schnell, und dann als Hauptspeicher-Ersatz eingesetzt werden. Damit der Applikationsentwickler und Anwender mit diesen verschiedenen Gerätecharakteristika problemlos umgehen kann, wurde ein sogenannter *Objekt Store* eingeführt.

Der *Objekt Store* ist ein abstrakter Behälter, der nach außen eine Schnittstelle zur Datenhaltung anbietet. Diese Schnittstelle abstrahiert von den Eigenschaften des darunterliegenden Speichers. Der *Objekt Store* kann als Ablagespeicher und als Programmspeicher genutzt werden. Wenn der *Objekt Store* als Ablagespeicher genutzt wird, kann er einen Speicher für ein Dateisystem, eine Datenbank und eine *Registry* bereitstellen. Abbildung 3 zeigt wie der *Objekt Store* auf einem *Handheld PC* verwendet wird.

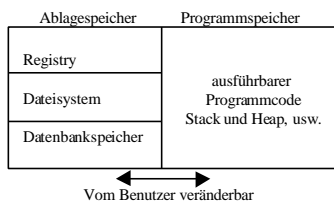


Abbildung 3: Die Architektur des Objektstores auf einem Handheld PC

Der Ablagespeicher, der sich innerhalb des *Objekt Stores* befindet (vgl. Abbildung 3) kann eine *Systemregistry*, ein Dateisystem und eine Datenbank enthalten. Das Dateisystem enthält Applikationen und Dateien. Die *Systemregistry* enthält die Systemkonfiguration und andere Informationen, welche die Applikation zur Laufzeit benötigt. Der *Registry-Speicher* wird vom Betriebssystem und den Applikationen benötigt, um dort Informationen über die Systemkonfiguration abzufragen und zu hinterlegen. Die Datenbank ist ein strukturierter Speicher auf dem einfache Datenbankoperationen ausgeführt werden können. Die Datenbankschnittstelle entspricht keiner herkömmlichen Win32-Datenbank-Schnittstelle. Der Programmspeicher wird zur Ausführung von Programmen benötigt und enthält Speicherstrukturen wie *Stacks*, *Heaps* und evtl. den Programmcode.

Damit die Hardware einheitlich angesprochen werden kann, sind Gerätetreiber notwendig, die von den Eigenschaften der Hardware abstrahieren. Der *Device Manager* ist für die Verwaltung eines großen Teils aller im System vorhandenen Treiber verantwortlich. Diese Treiber werden teils von Microsoft mitgeliefert oder müssen von dem *Original Equipment Manufacturer* (OEM), dem Hersteller der Hardware, entwickelt werden. Damit Win CE mit seiner Umwelt kommunizieren kann, ist eine graphische Benutzeroberfläche nötig, die nicht zwingend in das System eingebunden werden muß. Ist die

Kommunikation mit dem Benutzer nicht erwünscht, werden oft Schnittstellen für externe Geräte verschiedenster Art notwendig. In diesem Zusammenhang sind die Begriffe *Graphics*, *Window und Event Manager Subsystem*, *Windows CE Desktop*, *NDIS* und *Universal Serial Bus* von Bedeutung, die im folgenden dargestellt werden.[15]

Das *Graphics, Window und Event Manager Subsystem* (GWES) stellt elementare Funktionen zur Verfügung, die benötigt werden, um Eingaben des Benutzers entgegenzunehmen und eine graphische Ausgabe bereitzustellen. Die Pakete des *User* und *Graphics Device Interfaces* (GDI) werden analog zu den Windows Desktopvarianten verwendet. Jedoch wurden hier beide zu einer Einheit zusammengefaßt. Das *User-Paket* vereinigt den Teil des GWES, der Benutzereingaben externer Geräte, wie der Tastatur, bereitstellt. Dabei ist die GDI für die Komponenten zuständig, die eine graphische Ausgabe zur Verfügung stellen.[1]

Der *Windows CE Desktop* ist eine Sammlung von Komponenten, die Shell Dienste bereitstellen. Diese Dienste umfassen u.a. Grafikfunktionen für den Hintergrund, einen Kommandoprozessor und den Task Manager.

Die *Kommunikation* [11] ist in mobilen Systemen äußerst wichtig, um sie mit anderen Systemen in Beziehung zu setzen. Bereits die Darstellung der nötigen Grundlagen kann aus Platzgründen hier nicht mehr aufgenommen werden. Hier sei nur soviel gesagt, daß mit der *NDIS* [8] ein kompletter *OSI-Protokollstack* zur Verfügung gestellt wird. Der *Universal Serial Bus* [5] ist externer Bus. Er wurde speziell für den flexiblen Umgang und für die einfache Integration mit verschiedenen Komponenten entwickelt. Hersteller von Kommunikationshardware fügen darüber hinaus eigene Treiber und zusätzliche Protokolle hinzu.

Im nächsten Abschnitt soll dargestellt werden wie die einzelnen Win CE Konfigurationen zusammengestellt werden und welcher Speicherbedarf daraus für das eingebettete System entsteht.

3. Komponenten und Speicherverbrauch der Softwarekonfiguration

Der *Original Equipment Manufacturer* (OEM) entwickelt Peripheriegeräte und die zur Integration nötigen Gerätetreiber, während der *Embedded System Developer* entscheidet, welche Soft- und Hardwarekomponenten er einsetzt, um ein spezifisches System zusammenzustellen. In diesem Abschnitt werden zunächst die Softwarekomponenten betrachtet, während im darauf folgenden näher auf die Hardwarekomponenten eingegangen wird. Die eingesetzten Softwarekomponenten stehen in engem Zusammenhang mit den verwendeten Hardwarekomponenten. Durch die von dem *Embedded System Developer* vorgegebene Konfiguration

des Systems wird der Speicherbedarf bestimmt, der später benötigt wird, um das System zu betreiben. Die Komponentenbauweise von Win CE erlaubt das Anpassen des Betriebssystems an unterschiedliche Gegebenheiten.

Ein Modul ist in Win CE eine komplett funktionierende Einheit, die im System vorhanden ist oder nicht. Das Dateisystem besteht beispielsweise aus den Komponenten für ein Dateisystem auf Basis von RAM, ROM und jeweils einer Einheit für die zu unterstützenden Datenbank- und Registryeinheiten (vergleiche Abbildung 3). Einige große Module bestehen aus mehreren Komponenten, die ebenfalls in Abhängigkeit von der zu unterstützenden Funktionalität weggelassen oder hinzugenommen werden können. Die Granularität dieser Komponenten ist entscheidend, um individuelle Lösungen bedarfsgerecht abzustimmen.

Der Speicherbedarf, der zu verwendenden Module und Komponenten wird vom *Embedded System Developers* benötigt, um Entscheidungen hinsichtlich des Systems zu treffen. Er verwendet den *Platform Builder*, um für sie eigene Win CE Versionen zusammenzustellen. Diese untersucht er auf die von ihm gewünschten Erfordernisse. Eine Zusammenstellung aller möglichen Konfigurationen ist daher nicht möglich, schließlich erstreckt sich eine Win CE-Distribution über mehrere CD-ROMs und einige Module sind nur wenige KB groß. Eine Zusammenstellung einiger aussagekräftigen Konfigurationen befindet sich in Abschnitt 3.1.

Eine Win CE-Konfiguration enthält typischerweise RAM- und ROM-Speicher. Der Speicherinhalt des RAM wird durch eine Batterie gehalten, wenn das System ausgeschaltet wird. Das ROM enthält typischerweise ausführbare Programme, Systemdateien und Dateien für den lesenden Zugriff.

Der RAM Programmspeicher wird zur Ausführung von Programmen benötigt. Er enthält neben dem Programmcode, eine Datensektion für Daten, einen *Heap* und einen *Stack*. [12]

Der Programmspeicher kann in der Größe dynamisch verändert werden, ohne daß ein erneutes Booten des Systems notwendig wird. Der RAM Ablagespeicher enthält üblicherweise das Dateisystem, *Registry* und Datenbankspeicher, sowie Speicher für Daten mit Schreib-/ Lesezugriffen.

Das Dateisystem enthält Applikationen und Daten, die vom Benutzer installiert oder erstellt worden sind. Das Dateisystem ist immer komprimiert. (Vgl. Abbildung 3 im vorherigen Abschnitt.)

Während der Systeminitialisierung werden zunächst größere Datenblöcke eingelagert. Kleine Speicherbereiche, die unbenutzt geblieben sind, werden danach mit kleinen Datenblöcken, wie z.B. Fonts aufgefüllt. [3]

In dem folgenden Abschnitt wird davon ausgegangen, daß der *Embedded System Developer* die notwendigen Komponenten zusammengestellt hat und nun einzelne Hardwarekomponenten aufeinander abstimmt.

3.1 Beispielkonfigurationen und Speicherverbrauch

Der Speicherverbrauch unterschiedlicher Beispielkonfigurationen von Win CE 2.0 auf einer Hitachi D9000 Plattform mit SH3-Prozessor nach Systemstart ist in Abbildung 4 dargestellt. Es handelt sich hierbei um eine willkürlich gewählte Plattform. Aufgrund der Einheitlichkeit werden alle anderen Testwerte ebenfalls auf Grundlage dieser Plattform angegeben und weitere technische Details dazu angegeben.[3]

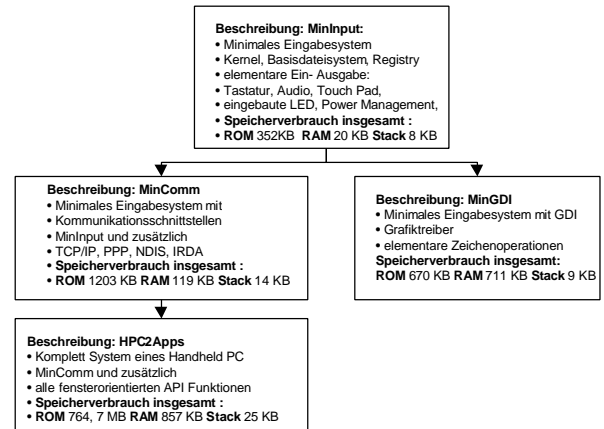


Abbildung 4: Speicherverbrauch verschiedener Win CE Konfigurationen

4. Konfiguration der Hardware

Im folgenden sollen verschiedene Hardwarekonfigurationen vorgestellt werden. Der Speicher ist ein besonders grundlegender Teil einer solchen Hardwarekonfiguration. Dies läßt sich dadurch begründen, daß einerseits der Systemstart ohne irgendeine Form von Hauptspeicher nicht denkbar wäre und andererseits andere externe Geräte auf den Hauptspeicher abgebildet werden können, und somit auf einfache Weise das Ansprechen externer Geräte ermöglicht wird. Eine besondere Rolle soll dabei *Flash Memory* als Komponente eines eingebetteten Systems spielen, da diese in vielerlei Hinsicht flexibler ist, als herkömmliche D-RAM Chips sind, aber auch einige Einschränkungen mit sich bringt. Die Techniken im Umgang mit diesen Einschränkungen können auf andere Karten und externe Geräte übertragen werden.

Der Einsatz von Festplatten ist in eingebetteten Systemen unter bestimmten Bedingungen nicht empfehlenswert. Die Festplatten sind vergleichsweise groß und für einige Einsatzgebiete nicht robust genug. *Flash Memory* soll hier als Systemkomponente für RAM und ROM allgmein und als Ersatz für Festplatten diskutiert werden.[14]

4.1 Eigenschaften von Flash Memory Bausteinen

Die Eigenschaften von *Flash Memory* können folgendermaßen angegeben werden: Die Zugriffszeit von Flash Memory liegt im lesenden Zugriff zwischen dem einer Festplatte und dem von D-RAM Bausteinen. *Flash Memory* ist preiswert, benötigt wenig Platz und hält Belastungen stand, denen die Mechanik von Festplatten nicht gewachsen ist. Er ist zudem sparsam bei Zugriffen auf das Medium. Im Gegensatz zum DRAM hat er den Vorteil, daß er keinen Strom benötigt, um Daten persistent abzulegen. Er besitzt jedoch einige Eigenschaften, die das Arbeiten mit ihm umständlich machen. Beispielsweise dauern Schreibzugriffe vergleichsweise lang, was sich insbesondere bei Echtzeitbedingungen störend auswirken kann. Um dieses Problem zu umgehen, gibt es bei manchen Karten ein spezielles Kommando, welches eine Schreiboperation vorzeitig suspendiert. Ein weiterer gravierender Nachteil ist, daß die einzelnen Zellen des *Flash Memory* nicht beliebig oft wieder beschreibbar sind. Dies zwingt Entwickler, Konzepte zu entwerfen, welche die vergangenen Schreibzugriffe auf das Medium speichern, um die Zugriffe in Zukunft so zu gestalten, daß die Speicherzellen dauerhaft zugreifbar bleiben. Eine weitere Beschränkung, die ebenfalls für Schwierigkeiten sorgt, ist, daß eine kleinste löschbare Speichereinheit existiert, die üblicherweise im Bereich zwischen 1KB und 4KB liegt.

Im folgenden sollen nun zwei verschiedene Lösungen von unterschiedlichen Herstellern betrachtet werden. Der eine Hersteller ist M-Systems und der andere Intel. Während M-Systems [14] eine Softwareemulation anbietet, die den Speicher wie eine Festplatte ansprechen läßt, setzt Intel [14] auf eine eigene Schnittstelle, die zwar leistungsfähiger ist, aber keinem bisher existierenden Standard entspricht.

Grundsätzlich lassen sich zwei Arten von *Flash Memory* unterscheiden: der *Datenflash* und der *Codeflash*. Auf den *Datenflash* kann im allgemeinen erst zugegriffen werden, wenn das Betriebssystem die Kontrolle hat, also d.h. der Bootvorgang abgeschlossen ist. Die Komponenten von M-Systems bilden hier eine Ausnahme, auf sie wird weiter unten noch Bezug genommen. Das *Kernel* greift auf den Gerätetreiber zu, um Zugriff auf die *Flash Memory* Bausteine zu erhalten. Beim *Codeflash* ist die CPU in der Lage, die Speicherzellen direkt zu adressieren, d.h. die CPU kann unmittelbar nach dem Einschalten des Systems auf die einzelnen Zellen zugreifen und dort beispielsweise den Initialisierungscode ausführen. Diese Fähigkeit wird als *Execute In Place* (XIP) Fähigkeit bezeichnet.

Folgende Abbildung zeigt, wie *Codeflash* als Ersatz für einen ROM-Baustein in einem eingebetteten System eingesetzt werden kann.

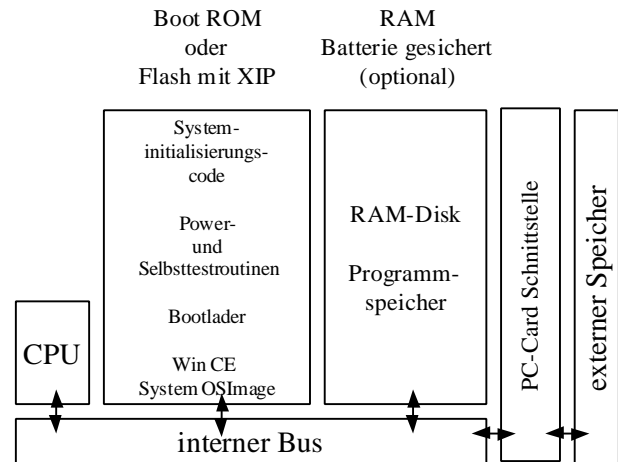


Abbildung 5: Elementare Bestandteile einer Handheld PC Plattform

In Abbildung 5 ist eine Plattform dargestellt, die typischerweise in Handheld PC Verwendung findet. Sie ist für eine Vielzahl von Einsatzgebieten gut geeignet. Es gibt jedoch Bedingungen, bei denen ein Überdenken der Konfiguration notwendig wird. Die Batterie dieses Systems muß regelmäßig gewartet werden, damit das System dauerhaft über persistenten RAM-Speicher verfügt. Zwar ist bei anderen Systemen auch eine Stromversorgung, z.B. in Form von Batterien nötig, jedoch kann u.U. auf einige Zellen verzichtet werden.

Grundsätzlich kann *Flash Memory* direkt an den internen Bus und/oder über eine PC-Kartenschnittstelle angeschlossen werden. Wenn der Zugriff auf eine PC-Kartenschnittstelle möglich ist, kann auf eine Vielzahl von Lösungen zurückgegriffen werden. Ein Beispiel für eine solche Karte sind *ATA-Flash-Memory-Karten*. Sie emulieren Festplatten. Diese Emulation wird von einem Chip durchgeführt, welcher sich auf der Karte befindet. Sie erscheinen wie normale Festplatten im System, so daß eine gesonderte Behandlung der hardware-spezifischen Eigenschaften des darunterliegenden *Flash Memory*s entfallen kann; d.h. aber auch, daß die einzelnen Speicherkomponenten des *Flash Memory*s nicht direkt ansprechbar sind.

In Win CE kann *Flash Memory* in Form von MiniCards (Intel), dem Industriestandard entsprechende PC-Karten, und als *built-in DiskOnChip* Geräte von der Firma M-Systems eingesetzt werden. Abbildung 6 zeigt diese Bausteine rechts unten. Auf die Lösung vom M-Systems und Intel wird im Verlaufe der Arbeit noch näher eingegangen. [6]

4.2 Konfiguration mit Daten-Flash

M-Systems ist ein Entwickler von *Flash Memory-Komponenten* und dazugehöriger Software, die sich darauf spezialisiert haben, *Flash Memory* wie eine Festplatte anzusprechen. Im Gegensatz zu den oben

erwähnten ATA-Karten wird hier eine Softwareemulation durchgeführt. Dieses Konzept wird *TrueFFS (True Flash File System)* und die dazugehörige Übersetzungsschicht mit *FTL (Flash Translation Layer)* bezeichnet.

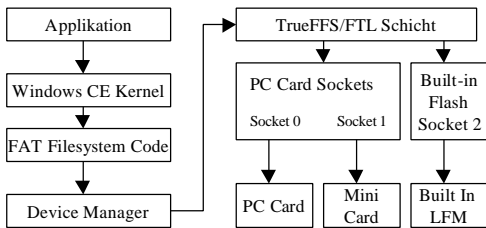


Abbildung 6: Zugriff auf das TrueFFS von M-Systems

Dieses Konzept stellt kein neues Dateisystem dar, wie der Name zunächst suggeriert. Die Applikation benutzt einen *Handle*, um Operationen auf dem Dateisystem auszuführen. Diese werden an das *Kernel* weitergereicht, welches vom FAT-Dateisystem Gebrauch macht, um die logische Blocknummer in eine physische umzuwandeln. Mit dieser Blocknummer kann nun geprüft werden, ob sich der entsprechende physische Block im Hauptspeicher befindet. Ist das nicht der Fall, greift das FAT-Dateisystem über den *Device Manager* auf das *TrueFFS* zu. Es benutzt dafür Kommandos, die physische Blöcke des Mediums adressieren. Die *FTL*-Schicht greift entweder auf die PC Kartenschnittstelle mittels *PC Card Sockets* oder über den internen Bus als *Built-in Flash* (Abbildung 6, rechter Teil) zu. Auf die dahinterstehenden Gerätetreiberkonzepte wird weiter unten nochmals eingegangen. Wesentlich ist, daß einerseits ein direkter Zugriff auf den Bus erfolgen kann. Der Baustein wird dann direkt angeschlossen. Andererseits kann aber auch zuerst über einen am Bus befindlichen Anschluß zugegriffen werden, um Zugriff auf das Gerät zu erhalten. Wird über den internen Bus zugegriffen, ändert sich die Systemkonfiguration, wie Abbildung 7 zeigt. Der Einsatz dieser Bausteine ist technisch auch bei anderen Betriebssystemen wie Palm OS möglich. Ob die entsprechenden Lizenzen und Gerätetreiber auch tatsächlich vorhanden sind, ist eine andere Frage.

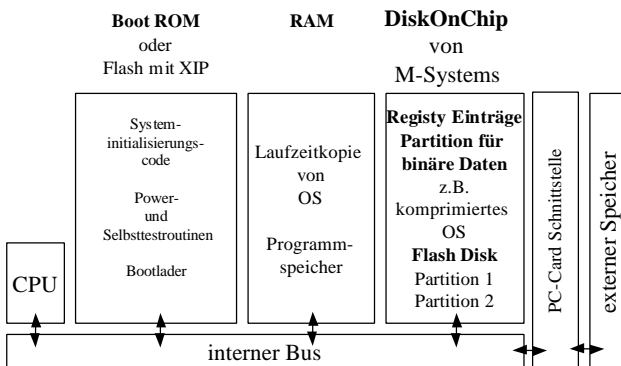


Abbildung 7: Erweiterung des Handheld PCs mit Flash Memory von M-Systems

Im Boot-ROM könnte das Image des Betriebssystems entfernt und auf eine Flash Memory Karte abgelegt werden. Diese kann zusätzlich für andere Zwecke verwendet werden. Sie kann beliebig partitioniert und so wie eine Festplatte (Flash Disk) eingesetzt werden oder zur Verwaltung der Registry-Einträge dienen (vergleiche Abbildung 7). [14]

4.3 Systemstart

Der Bootvorgang der ersten (durchgehende Linie, vgl. Abbildung 5 Handheld PC) und zweiten (gestrichelte Linie, vgl. Abbildung 7 Handheld PC mit M-Systems Flash) Konfiguration ist in Abbildung 8 dargestellt.

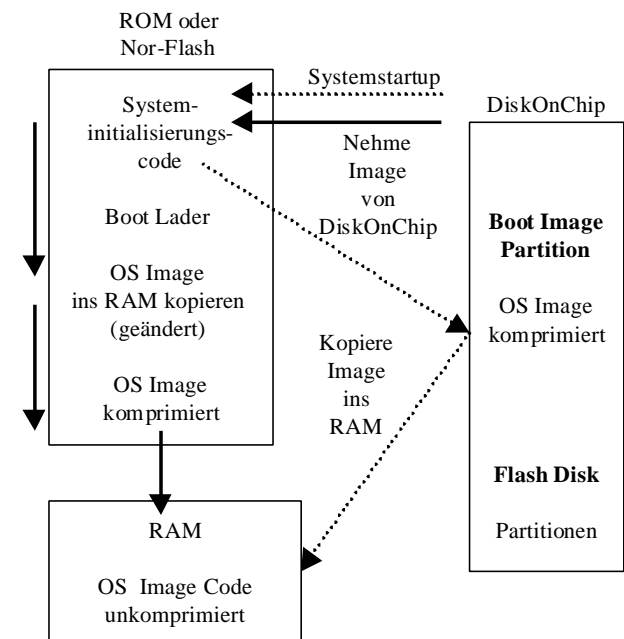


Abbildung 8: Bootvorgang nach und vor Einsatz des Flash Memorys von M-Systems

Während der Bootlader des ersten Systems das Betriebssystemimage aus dem ROM auspackt und ins RAM kopiert, springt der Bootlader des zweiten Systems in eine Routine, die ihm den Zugriff auf das *Flash Memory* Medium ermöglicht. Danach wird das Image aus dem *Flash Memory* ausgepackt und ins RAM kopiert. Der Intel StrataFlash Memory ist ein *Codeflash*. Das Besondere an diesem Speicher ist, daß die CPU direkt auf den Speicher zugreifen und dort Programme ausführen (XIP) kann. Der ROM Baustein kann entfallen, da unmittelbar nach dem Systemstart auf den Speicher zugegriffen werden kann. Die CPU kann zum Initialisierungscode springen und das Betriebssystem starten. Wenn das Betriebssystem nicht komprimiert im ROM abgelegt wurde, kann es dort direkt ausgeführt werden. Andernfalls kommt der On-Demand Paging Mechanismus zum Einsatz, der die gerade benötigten

Seiten ausgepackt und ins RAM kopiert (vergleiche Abbildung 9). Nachteil der letztgenannten Vorgehensweise ist, daß u.U. mehr RAM Speicher benötigt wird als bei der ersten.[14]

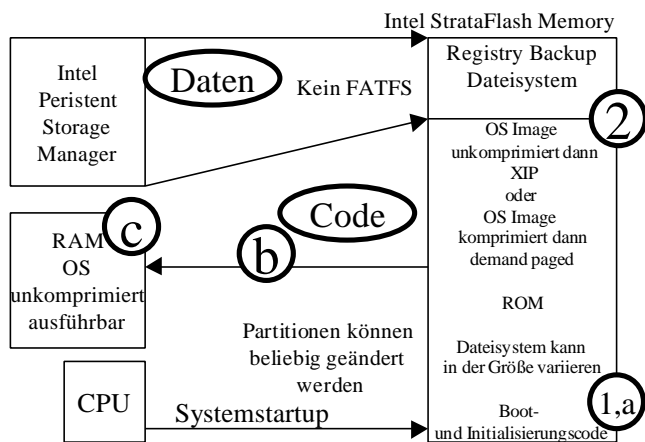


Abbildung 9: Intel StrataFlash Memory als Beispiel für einen Code Flash

Intel entwickelte für den Zugriff auf den Flash Memory ein eigenständiges Dateisystem mit ungefähr 20 Operationen. Die Lösungen von M-Systems und Intel unterscheiden sich in wesentlichen technischen Details. Eine ausführliche Darstellung beider Produkte findet sich in [14].

Nachdem nun dargestellt wurde, wie die Hardware zusammengestellt und das Betriebssystem gebootet wird, kann im folgenden davon ausgegangen werden, daß ein laufendes System (Hard- und Software) für weitere Betrachtungen vorhanden ist. Das laufende Betriebssystem besitzt die volle Kontrolle über alle Betriebsmittel und Prozesse und wird im folgenden näher betrachtet.

5 Speicherverwaltung und Prozeßmodell

Das Prozeßmodell von Win CE erlaubt es, 32 Prozesse auf das System zu bringen. In einem Prozeß können sich eine nur durch die Systemressourcen begrenzte Anzahl *Threads* befinden. Ein *Thread* innerhalb des Hauptkontrollflusses wird als primärer *Thread* bezeichnet. Er kontrolliert die übrigen. Der 4 GB große virtuelle Adreßraum wird in einen System- und einen Benutzeradreßraum aufgeteilt, wobei der Benutzer die unteren 2 GB benutzt und das Betriebssystem die oberen. Die 2 GB des Benutzeradreßraums werden in zwei Teile, zu je einem GB, aufgeteilt. Die untere Hälfte dieses Bereichs wird in 33 *Slots*, mit einer Größe von je 32 MB, aufgeteilt. Jedem Prozeß wird bei dessen Initialisierung ein offener *Slot* zugeteilt, den er während der gesamten Ausführung behält. Einzige Ausnahme stellt der gerade laufende Prozeß dar: Er befindet sich immer in *Slot 0* (vergleiche Abbildung 10).

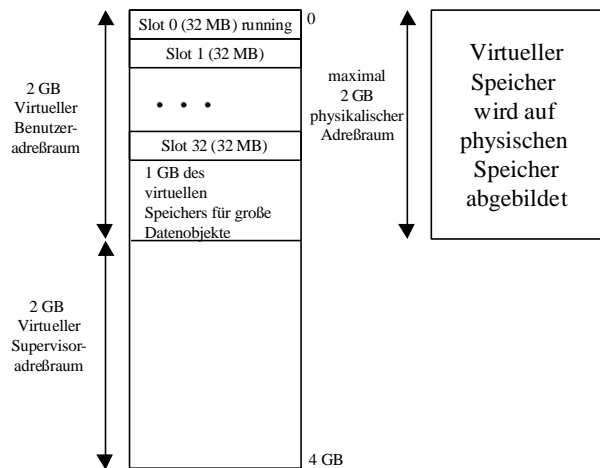


Abbildung 10: Das Windows CE Speichermodell

Nachdem nun das gesamte Speichermodell von Win CE grob beschrieben wurde, soll nun der einzelne Prozeß betrachtet werden. Zusätzlich erhält jeder Prozeß beim Start neben dem *Code* einen *Heap* und jeder *Thread* einen eigenen *Stack*. Der Speicher innerhalb des 32 MB Bereiches wird von oben nach unten mit folgenden Daten gefüllt: *Dynamic-Link Libraries* (DLLs), dem *Stack*, dem *Heap* und dem *Applikationscode* (exe-Dateien). Die unteren 64 KB bleiben immer frei. Diese Sachverhalte zeigen wie der in Abbildung 3 abstrakt dargestellte Objekt Store umgesetzt wird.

Die DLLs werden entweder von der Applikation oder dem *Device Manager* kontrolliert. Wenn der *Device Manager* eine DLL in den Speicher lädt, handelt es sich um sogenannte *Stream Treiber*. Diese Gerätetreiber werden weiter unten noch ausführlich behandelt. Der bereits angesprochene *TrueFFS*-Treiber der Firma M-Systems ist ein solcher *Stream Treiber*. Alle DLLs eines Prozesses werden in denselben Speicherbereich geladen. Falls dennoch zusätzlicher Speicher benötigt wird, kann außerhalb der 32 MB Grenze Speicher vom Betriebssystem angefordert werden. Bei den Betrachtungen dieses Abschnitts konnte davon ausgegangen werden, daß die einzelnen Prozesse und Betriebssystemkomponenten gegen unbefugte Zugriffe geschützt waren. Anders verhält es sich, wenn einzelne *Threads* in einen gemeinsamen Kontrollfluß laufen. Die sogenannten Leichtgewichtsprozesse sind besonderes bei ein-/ausgabeintensiven Aufgaben von Vorteil. Bei ihnen wird zugunsten der Ablaufgeschwindigkeit auf einen Speicherschutz verzichtet.[2]

6 Threads und Threadprioritäten

Die *Win CE Threads* werden durch einen *preemptiven Dispatcher* gesteuert, der auf Zeitscheiben und Prioritäten basiert. Es existieren 8 Prioritätsstufen (0 bis 7), wobei 0 die höchste Priorität ist. Die Stufen 0 und 1 werden typischerweise für Echtzeitbedingungen und

Gerätetreiber verwendet, während die Stufen 2 bis 4 für *Threads* des *Kernels* und andere „normale“ Applikationen verwendet werden. Die Stufen 5 bis 7 finden üblicherweise für Applikationen Verwendung, welche jederzeit durch andere verdrängt werden können. Die Verdrängungsstrategie basiert allein auf der Priorität der *Threads*. *Threads* mit einer höheren Priorität laufen vor allen *Threads* mit einer niedrigeren Priorität. *Threads* mit der selben Priorität laufen *Round-Robin*, d.h. jeder bekommt eine Zeitscheibe fester Größe zugeteilt.

Threads mit einer niedrigeren Priorität laufen, wenn kein *Thread* mit höherer Priorität im Zustand lafbereit oder laufend ist. Die *Threads* mit der höchsten Prioritätsstufe (0) laufen nicht nach Zeitscheibenverfahren. Sie laufen solange, bis sie ihre Ausführung abgeschlossen haben. Die Prioritäten der *Threads* werden vom *Kernel* nicht verändert bis auf eine einzige Ausnahme. Besitzt ein *Thread* mit niedriger Priorität eine Ressource, auf die ein *Thread* mit höherer Priorität wartet, kommt der *Thread* mit der höheren Priorität nicht zum Laufen (*priority inversion*). Um dieses Problem aufzulösen, erhält der *Thread* mit der niedrigen Priorität die höhere übertragen, bis dieser die Ressource freigegeben hat. Schließlich wird ihm wieder die ursprüngliche (niedrige) Priorität zugewiesen, damit der *Thread* mit der hohen Priorität zur Ausführung kommt.

Nachdem hier dargestellt wurde wie *Threads* ablaufen, werden im folgenden Gerätetreibermodelle vorgestellt. Mit diesen Gerätetreibern kann in einem nächsten Abschnitt dargestellt werden, wie letztendlich ein Zugriff auf die verschiedenen Geräte erfolgen kann.[13]

A Gerätetreibermodelle

In Win CE werden vier verschiedene Gerätetreibermodelle verwendet. Zwei dieser Modelle wurden speziell für die Erfordernisse eines eingebetteten Betriebssystems entwickelt, während die zwei weiteren aus anderen Betriebssystemen übernommen wurden. Die zwei Win CE spezifischen Treiber sind der *Native Device Treiber*, auch *built-in-driver* genannt, und der *Stream Interface Driver*, auch *installable-driver* genannt. Die zwei externen sind der *Universal Serial Bus Driver* und das *Network Driver Interface Specification* (NDIS). Die Treibermodelle unterscheiden sich in erster Linie durch die Softwareschnittstelle, die nach außen angeboten wird und nicht durch die Geräte, die sie ansprechen.[4]

A.1 Native Treiber

Native Treiber bedienen Geräte, welche fest in das System eingebaut sind, wie beispielsweise eine Tastatur oder ein *LCD-Touchscreen*. Diese Geräte haben eine besondere Bedeutung für das System, da sie bereits während des Bootvorgangs ansprechbar sind. Während des Bootvorgangs werden diese Geräte initialisiert und

der zugeordnete Native Treiber wird für eine Eingliederung ins Betriebssystem vorbereitet. In *Desktop-PCs* werden Karten üblicherweise an den Datenbus angeschlossen, um auf die entsprechenden Portadressen (*I/O Mapped I/O*) zugreifen zu können. Auf diese Portadressen kann allerdings nur im *Kernel Mode* zugegriffen werden. Daher müssen *Native Treiber* im *Kernel Mode* laufen. [16]

A.2 Stream Treiber

Der *Stream Interface Driver* ist typischerweise für Geräte verantwortlich, die vorübergehend an das System angeschlossen und wieder entfernt werden können. Unter Win CE müssen alle nicht eingebauten Geräte über externe Anschlüsse, wie einen *PC-Karten Socket* oder den *USB-Port* angeschlossen werden. Dadurch kommen einige Peripheriegeräte unter Win CE in eine ähnliche Situation wie Drucker auf herkömmlichen *Desktop PCs*. Der Gerätetreiber läuft im Benutzermodus, er benötigt eine eingebaute Hardwareschnittstelle, um die dort angeschlossenen Geräte anzusprechen. Der *Stream Interface Driver* ist eine DLL, welche von der Applikation oder dem *Device Manager* als Reaktion auf ein neu hinzugefügtes Gerät in den Hauptspeicher geladen werden kann (siehe Abschnitt 5). Aufgabe des *Stream Treibers* ist es, das Gerät für die Applikation, wie eine spezielle Datei im Dateisystem aussehen zu lassen. *Stream Treiber* können entweder direkt über eine ins RAM abgebildete Portadresse (*Memory-Mapped IO*) oder mittels eines *Native Treibers* auf die Hardware zugreifen. So muß beispielsweise ein *Native PC Card Socket Treiber* zugreifbar sein, damit auf die daran angeschlossene Hardware, wie z.B. eine *Flash Memory Karte*, zugegriffen werden kann.

Es ist denkbar, daß für einen *Native Treiber* zusätzlich ein *Stream Treiber* zur Verfügung gestellt werden soll, damit die Applikation das Gerät durch einfache Dateioperationen nutzen kann. [16]

A.3 USB- und NDIS Treiber

USB Treiber [5] bedienen USB Geräte. Dieser Treiber kann dann im Hinblick auf die individuellen Anforderungen einen *Native Treiber*, einen *Stream Treiber* oder eine API als Schnittstelle anbieten. Das *NDIS Treibermodell* [8] wurde von Win NT übernommen und stellt für Win CE den kompletten *OSI-Protokollstack* zur Verfügung. Sie sind für die folgenden Betrachtungen nicht von weiterem Interesse.

A.4 Zusammenfassung

Die nächste Abbildung 11 zeigt, daß ein Teil der *Native Treiber* durch das GWES verwaltet wird, während andere unter der Verwaltungshoheit des *Device Managers*

stehen. Das GWES stellt elementare Funktionen zur Verfügung, die benötigt werden, um Eingaben des Benutzers entgegen zu nehmen und eine graphische Ausgabe bereitzustellen. Er kontrolliert einen Teil der *Native-Gerätetreiber* und damit auch die Geräte (Abbildung 11 rechts), während ein anderer Teil vom *Device Manager* verwaltet wird. Der *Stream Treiber* kann zu verschiedenen Zeitpunkten in das System eingegliedert werden und somit auf die aktuelle Situation flexibel reagieren. Er kann während des Bootvorgangs, wenn ein zusätzlich angeschlossenes Gerät erkannt wird oder von der Applikation in den Hauptspeicher geladen werden (vergleiche Abbildung 11 links unten). [17]

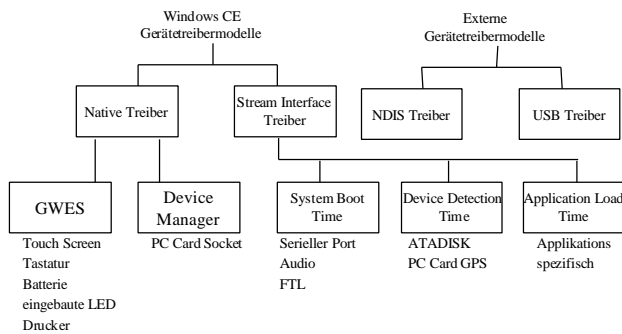


Abbildung 11: Gerätetreiberkonzepte in Windows CE

B Device Manager, Stream Interface Treiber und Native Treiber

Im diesem Abschnitt wird dargestellt, wie letztendlich ein Zugriff auf ein Gerät über *Native* und *Stream Treiber* erfolgt und wie dadurch die notwendige Kommunikation zwischen *Kernel*, *Device Manager*, *Native-* und *Stream Treiber* abgewickelt wird.

Der *Device Manager* lädt alle notwendigen Treiber während des Bootvorgangs in den Speicher oder wird aktiv beim Anschließen eines Gerätes oder während das Betriebssystem bereits die Kontrolle über das System hat. Der *Device Manager* ist ein Prozeß, der im Benutzermodus ausgeführt wird, d.h. er ist nicht Teil des *Kernel*s, aber er interagiert mit dem *Kernel* und verschiedenen Gerätetreibern (Abbildung 11 unten). Er verwaltet einen Teil der angeschlossenen Geräte, d.h. wenn ein Gerät neu angeschlossen oder entfernt wird, wird er aktiv, um die *Registry* zu aktualisieren. Die *Registry* ist ein Speicher, der Informationen über die Systemkonfiguration enthält. Er wird von Applikationen benutzt, um Systeminformationen in Erfahrung zu bringen. In Ausnahmefällen veranlaßt die Applikation das Laden eines *Stream Treibers*, wenn ein Gerät angeschlossen oder entfernt wurde. Der *Device Manager* registriert den Dateinamen eines *Stream Treibers* und die damit in Zusammenhang stehenden Geräte in Zusammenarbeit mit dem *Kernel*. Die Applikation kann

das Gerät über einen Dateinamen wie eine Datei ansprechen ohne Einzelheiten über das Gerät zu benötigen. Die entsprechende Vorgehensweise ist in Abbildung 6 dargestellt und wurde im zugeordneten Abschnitt erläutert.

Bei *Plug and Play* Geräten wird vom gerade eingefügten Gerät ein *Identifer* zurückgegeben, der vom *Device Manager* genutzt wird, um den richtigen Treiber zu finden. Andernfalls wird eine Erkennungsroutine durchlaufen, welche die nötigen Informationen beschafft, um den richtigen Treiber ausfindig zu machen.

Der Zugriff auf dieses Gerät erfolgt nun analog zur Vorgehensweise in Abbildung 6, jedoch wird in der folgenden Abbildung 12 dieser Vorgang erweitert.

Die Applikation greift auf ein Gerät über Dateisystemfunktionen des *Kernel*s zu. Das *Kernel* leitet diese Kommandos an den *Stream Treiber* weiter. Dieser greift entweder direkt, über ins RAM abgebildete Portadressen, auf das Gerät zu (*Memory mapped I/O*) oder benutzt einen *Native Treiber*, der Zugriff auf diese Adressen hat.

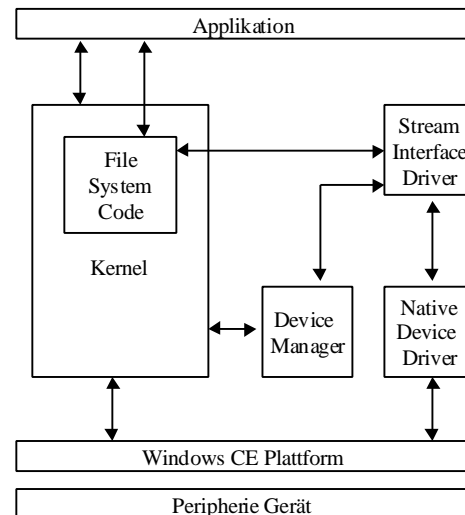


Abbildung 12: Interaktion von Applikation, Kernel, Device Manager und Gerätetreibern

Im folgenden Abschnitt wird der *Native Treiber* näher betrachtet. Er greift direkt auf die Hardware zu und stellt somit eine Abgrenzung zum Aufgabengebiet des *Embedded System Developers* und des *Orginal Equipment Manufacturers* dar. *Native Treiber* werden typischerweise vom *Orginal Equipment Manufacturers* oder Microsoft entwickelt. In Ausnahmefällen wird der *Native Treiber* vom *Embedded System Developer* entwickelt.[17]

B.1 Native Treiber

Es kann zwischen Ein-Schicht- und Zwei-Schicht *Native Treibern* unterschieden werden. Der Ein-Schicht-Treiber ist besser für zeitkritische Aufgaben wie z.B. Echtzeitbedingungen geeignet. Der Zwei-Schicht-Treiber

besteht aus einer oberen Schicht dem *Model Device Driver* (MDD) und einer unteren Schicht dem *Platform-Dependent Driver* (PDD).

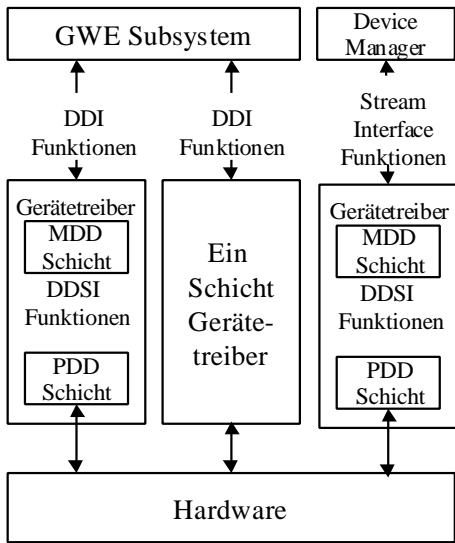


Abbildung 13: Ein- und Zwei-Schicht-Treiber kontrolliert durch Device Manager oder GWES

Die PDD Schicht greift direkt auf die Hardware zu. Sie bietet der MDD eine einheitliche Schnittstelle, die sogenannte *Device Driver Service Provider Interface* (DDSI) an. Die MDD kann dadurch auf beliebige Plattformen übertragen werden, ohne daß Veränderungen an ihr vorgenommen werden müssen. Diese stellt darauf aufbauend eine *Device Driver Interface* (DDI) zur Verfügung. Auf die MDD Schicht wird vom *Kernel*, dem *GWES* oder von einem *Stream Treiber* zugegriffen (Abbildung 13). Dabei wird an die in Abbildung 11 und Abbildung 12 dargestellten Sachverhalte angeknüpft. Der in Abbildung 6 dargestellte *TrueFFS Treiber* ist ein *Stream Treiber*. Er benötigt einen *Native Treiber*, um den Zugriff auf den entsprechenden Baustein zu erhalten.

Wenn ein *Stream Treiber* für den Zugriff auf ein Gerät verwendet werden soll, muß keine Verbindung des *Native Treibers* zum *GWES* vorhanden sein. Der *Stream Treiber* stellt der Applikation eine Dateischnittstelle zur Verfügung. Der *Stream Treiber* nutzt selber eine DDI Schnittstelle (vergleiche Abbildung 13).

Ein Zugriff auf Geräte kann durch die Applikation veranlaßt werden. Es kann aber auch vorkommen, daß Geräte der Applikation signalisieren, mit ihnen in Kontakt zu treten. Dies ist zum Beispiel nötig, wenn ein Gerät zusätzlich angeschlossen wird (vergleiche Abbildung 11 und Abschnitt 0 zweiter Absatz). Der dafür benötigte Mechanismus wird *Interrupt* genannt und im nächsten Abschnitt behandelt. [13]

B.2 Interrupt Handling mit Native Gerätetreibern

Unterbrechungsanforderungen sind Anfragen der Peripherie an die CPU. Diese sind notwendig, um der CPU zu signalisieren, daß Vorgänge seitens der CPU notwendig werden, um beispielsweise die Weiterverarbeitung von eingegangenen Daten zu veranlassen. Da Win CE in der Versionen 2.xx keine geschachtelten *Interrupts* zuläßt, ist die Abfertigung einer *Interruptanforderung* notwendig, damit die nächste bearbeitet werden kann. Durch diesen Sachverhalt wird die Echtzeitfähigkeit von Win CE begrenzt.

Die Interrupt Bearbeitung wurde bei Win CE in zwei Teile aufgespalten. Der eine Teil wird im *Kernelmodus* durchgeführt, der andere Teil im Benutzermodus. Die *Interrupt Service Routine* (ISR) wird im *Kernelmodus* ausgeführt und der *Interrupt Service Thread* (IST) im Benutzermodus. Auf beide wird weiter unten noch ausführlich eingegangen. Während die ISR in Ausführung ist, sind alle Unterbrechungen (*Interrupts*) maskiert. Daher muß die ISR kurz und schnell sein (Abbildung 14). Die ISR ist Teil des *OEM Adaption Layer* und hat direkten Zugriff auf Hardwareregister. Aufgabe der ISR ist es, einen Interrupt Identifikator zu ermitteln und diesen an das *Kernel* zurückzugeben. Der *Interrupt Handler* verwendet diesen *Identifizier* dann, um den zugeordneten IST aufzurufen, welcher dann die eigentliche Interruptbearbeitung abwickelt. Die ISR löst dadurch die Zuordnung eines physischen Interrupts auf, den Interrupteingang der CPU auf einen logischen Interrupt den IST. Der IST bleibt solange inaktiv, bis er ein entsprechendes Signal erhält.

Bei Erhalt des Signals ruft dieser dann Subroutinen auf, die beim *Native-Zwei-Schicht-Treiber* in der PDD Schicht aufzufinden sind. Handelt es sich um einen Ein-Schicht-Treiber, muß dieser die entsprechenden Funktionen in seiner Schicht bereitstellen. Der IST kann als eine zweite Art *Thread* angesehen werden, der sich ausschließlich dadurch auszeichnet, daß er ein externes Gerät bedient. Er erfährt vom Betriebssystem keinerlei Sonderbehandlung und kann daher wie alle anderen *Threads* verdrängt werden (vergleiche Abschnitt 6). ISTs bekommen typischerweise einen der zwei höchsten Prioritäten zugeordnet, diese kann jeder andere *Thread* aber ebenso besitzen.

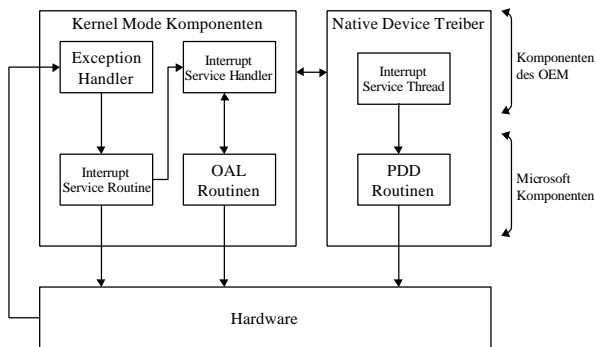


Abbildung 14: Schema einer Interruptanforderung und deren Bearbeitung

Wenn ein *Interrupt* auftritt, wird im Kernel direkt zum *Exception Handler* verzweigt (Abbildung 14). Er ruft die dort registrierte ISR auf, um den benötigten Identifikator zu erhalten. Die einzelne ISR wird während des Bootvorgangs beim *Exception Handler* registriert. Die Kommunikation zwischen der ISR, dem IST und den Unterrouinen des OAL wird durch den *Interrupt Service Handler* (ISH) gesteuert. Er übernimmt Details der Registrierung und Abmeldung eines Treibers für einen bestimmten Interrupt.

Nachdem nun der grundsätzliche Ablauf einer *Interruptbearbeitung* geklärt wurde, soll im folgenden Abschnitt auf die besonderen Erfordernisse des Echtzeitbetriebs eingegangen werden. Dabei wird einerseits durch Formeln der Ablauf einer *Interruptbearbeitung* beschrieben und anschließend werden Testwerte für die in den Formeln vorkommenden Parameter angegeben.[13]

C Prozeß-, Threadmodell und Echtzeitbedingungen

In Echtzeitsystemen hängt die Brauchbarkeit einer Berechnung nicht nur von der mathematischen Korrektheit ab, sondern auch von der Zeit, in der die Berechnung durchgeführt wird. Ein Echtzeitbetriebssystem (EZBS) ist Bestandteil eines Echtzeitsystems (EZS). Das EZBS muß auf die ihm umgebene Hard- und Software abgestimmt werden. Ein Betriebssystem muß bestimmten Anforderungen genügen, bevor es als EZBS in Betracht gezogen werden kann. Es muß *multithreaded* und *preemptive* sein. *Threadprioritäten* müssen unterstützt werden und ein Vererbungssystem für diese Prioritäten muß vorhanden sein. Die zur Verfügung gestellten Synchronisationsmechanismen müssen abschätzbare Verzögerungen zur Folge haben.

Den Entwicklern von EZS müssen darüber hinaus detaillierte Informationen über die *Systeminterruptstufen*, die Systemanrufe und die entsprechende Zeitinformation zur Verfügung gestellt werden. Diese Information muß im einzelnen beinhalten:

- Die maximale Zeit, die ein *Interrupt* vom Betriebssystem oder den Gerätetreibern maskiert

werden kann

- Die maximale Zeit, die Gerätetreiber benötigen, um einen bestimmten *Interrupt* zu bearbeiten, sowie die entsprechende Information über die eingegangene Unterbrechungsanforderung
- Die *Interruptverzögerung*, d.h. die Zeit von dem Eintreffen der Unterbrechungsanforderung bis zu deren Bearbeitung muß bekannt und berechenbar sein und den Anforderungen der Applikation gerecht werden.
- Die Zeit für jeden Systemaufruf muß vorhersagbar und unabhängig von der Anzahl von Objekten im System sein.

Jetzt soll dargestellt werden, wie diese Anforderungen in Win CE umgesetzt wurden. Wesentlich ist, daß eine Obergrenze für die Zeit angegeben werden kann, die benötigt wird, um einen *Echtzeitthread* zu starten, nachdem eine entsprechende Unterbrechungsanforderung eingetroffen ist.[9]

C.1 Threadsynchronisation

Threads, welche Synchronisationsmechanismen wie *Mutexe*, *Events* und *Critical Sections* anfragen und deren Anforderungen nicht unmittelbar erfüllt werden können, werden in eine *FIFO-by priority* Schlange gestellt. Für jede der acht Prioritätsstufen existiert eine solche Schlange. Eine Anfrage eines *Threads* wird in der jeweiligen Schlange ans Ende gestellt. Die einzelnen Anfragen werden am Schlangenkopf entnommen und der Reihe nach bedient. Der *Dispatcher* muß diese Schlangen anpassen, wenn *priority inversion* auftritt und bei den davon betroffenen *Threads* die entsprechenden Prioritäten anpassen. Zusätzlich gibt es noch Funktionen, die Zeitinformationen bereitstellen. Sie werden benötigt, um Echtzeitbedingungen zu verwirklichen.[9]

C.2 Zusätzliche Überlegungen zum Echtzeitbetrieb

Das Speichermanagement hat Einfluß auf die Zeit, in der die einzelnen Prozesse der CPU zugeteilt werden. Das *Paging*, d.h. das Ein- und Auslagern von Seiten, ist nichtdeterministisch. Damit wäre eine genaue Angabe der *Interruptverzögerung* ohne weitere Annahmen nicht möglich.

Der Seitenauslagerungsalgorithmus ist auf eine niedrigere Priorität gesetzt, als das für *Threads* mit Echtzeitbedingungen der Fall ist, damit diese nicht unterbrochen werden. Zudem werden die betroffenen *Threads* *locked down* gesetzt, wodurch sie permanent im physischen Speicher bleiben nicht am *Paging* teilnehmen.

Durch *Memory Mapping* kann die Leistung des Systems optimiert werden. Durch Datenstrukturen wird verhindert, daß zeitkritische Prozesse innerhalb desselben physikalischen Speichers laufen. Dies kann besonders bei

kooperierenden Prozessen oder zwischen Treibern und den zugeordneten Applikationen eine erhebliche Leistungssteigerung zur Folge haben.

Nachdem in diesem Abschnitt die Vorbereitungen für den Echtzeitbetrieb geschaffen wurden, wird im nächsten Abschnitt die Interruptverzögerung formal definiert und diese Definition mit Testwerten veranschaulicht. [9]

C.3 Interrupt Handling: IRQs, ISRs und ISTs

Verschachtelten *Interrupts* werden in Windows CE erstmals in der noch nicht erhältlichen Version 3.0 implementiert. Dies hat zur Folge, daß wenn sich eine ISR *Interrupt* erst einmal in Ausführung befindet, keine neue Unterbrechungsanforderung angenommen wird. Erst wenn die derzeit in der Ausführung befindliche ISR ihre *Interruptbearbeitung* vollständig abgeschlossen hat, kann es zum Aufruf einer entsprechenden ISR kommen, welche die gerade eingegangene Anforderung bedient.

Durch diesen Sachverhalt ist die *Interrupt Latency* definiert. Das ist die Zeit, die zwischen dem Eintreffen eines *Hardwareinterrupts* und dem Start der zugeordneten ISR verstreicht.

Die allgemeine Formel zur Bestimmung der *ISR Latency* ist wie folgt definiert:

$$\text{Start Of ISR} = \text{Value1} + \text{D_ISR_Current} + \text{SUM(D_ISR_Higher)}$$

Value1	Verzögerungswert, der durch die Bearbeitung im <i>Kernel</i> entsteht
D_ISR_Current	Das Zeitintervall einer sich gerade in Ausführung befindlichen ISR vom Eintreffen des Interrupts an bis zum Ende dieser ISR. Dieser Wert liegt irgendwo im Intervall [0, längste ISR im System].
Sum(D_ISR_Higher)	Summe aller Zeitintervalle aller höher priorisierten ISRs, die eintreffen, bevor diese ISR zur Ausführung kommt.

Durch die oben dargestellte Formel kann eine obere Grenze für die *Interruptverzögerung* angegeben werden. Die zugeordneten *Threads* sind permanent im Hauptspeicher verfügbar. Durch Setzen eines sogenannten *locked down Bits* werden sie vom *Paging* ausgeschlossen.

Beispiel: In einem System existiert eine ISR mit höchster Priorität und keine andere mit derselben Priorität. Dann nimmt D_IST_Higher den Wert Null an. Im günstigsten Fall ist keine andere ISR in Bearbeitung. Die ISR wird dann mit einer Verzögerung von Value1 gestartet. Im ungünstigsten Fall hat die längste ISR im System ihre Ausführung gerade gestartet, wenn die Unterbrechungsanforderung eingeht. Für die Verzögerung gilt dann Value1 + Dauer der längsten ISR im System.

Die Allgemeine Formel für die IST Verzögerung kann wie folgt angegeben werden:

$$\text{Start Of IST} = \text{Value2} + \text{SUM(D_IST)} + \text{SUM(D_ISR)}$$

Value2	Verzögerung, die durch Ausführung im <i>Kernel</i> entsteht
SUM(D_IST)	Summe aller Zeitintervalle aller höher priorisierten ISTs und aller höher priorisierten Threadwechsel, die zwischen dieser ISR und dem Start des zugeordneten ISTs aufgetreten sind.
SUM(D_ISR)	Summe aller Zeitintervalle aller anderen ISRs, die zwischen dieser ISR und dem zugeordneten IST aufgetreten sind.

Beispiel: Es befindet sich eine kritische ISR im System. Dieser ISR ist ein IST zugeordnet, welcher dadurch ebenfalls zeitkritisch ist. Beide bekommen die höchste Prioritätsstufe Null. Es gibt keine anderen ISTs, die zwischen dieser ISR und dem zugeordneten IST Unterbrechungen verursachen können. Es ist allerdings möglich, daß andere ISRs vor dem Start des zeitkritischen IST zur Ausführung kommen. Dadurch sind Fälle denkbar, in denen ein IST in seinem Start immer durch eine andere ISR gehindert wird. In der Praxis sind diese Fälle weniger von Bedeutung, da die Anzahl der eingehenden Interruptverzögerungen absehbar ist.

Der *Embedded System Developer* hat Einfluß auf *Interruptprioritäten*, *Threadprioritäten* und ISR und IST Ausführungszeiten, um das System hinsichtlich seiner Vorstellungen anzupassen. Die Werte Value1 und Value2 sind Werte, die er hinnehmen muß. Abschließend werden im nächsten Abschnitt zu diesem Themengebiet noch konkrete Testwerte angegeben und kommentiert.[9]

C.4 Interruptverzögerung

Folgende Testwerte ergaben sich bei einer Untersuchung, die 1000 *Interrupts* auf einem Hitachi D9000 mit SH3 Prozessor mit einer internen Frequenz von 58,98 MHz und externen von 14.745 MHz. Auf dem *Handheld PC* wurden lediglich die notwendigen Module zur Durchführung des Tests installiert (Nk.exe, Filesys.exe, Gwes.exe, Device.exe und Explorer.exe). Es wurden keine Interrupts durch Benutzer und andere Applikationen verursacht. Die Messungen wurden von dem Utility Intrtime.exe ermittelt.

Verzögerung	Werte des Tests
Start of ISR	[1.3-7.5] Mikrosekunden
Start of IST	[93-275] Mikrosekunden

Die Werte 1.3 und 1.6 des ersten Intervalls sind 293 und 549 mal aufgetreten, dies entspricht 84 Prozent aller Testmessungen. Ungefähr 90 Prozent (923 von 1000 Messungen) der Werte des zweiten Intervalls betragen 102 Mikrosekunden und kleiner.

Weitere Testmessungen belegten, daß der Wert Start of ISR von der Anzahl der Objekte im System weitgehend unabhängig ist. Vielmehr ist der Zeitpunkt, wann die Unterbrechungsanforderung vorliegt, dafür verantwortlich, wie groß dieser Wert sein wird, d.h., wann eine *Interruptbearbeitung* beginnt. Im folgenden soll der Wert Value2 definiert und durch Testmessungen in der Größe spezifiziert werden.[9]

Value2 = D_K_CALL + D_Next_Thread

D_K_Call Dauer des *Kernelanrufs*, genauer gesagt: die Zeitdauer, die im *Kernel* verblieben wird während der *Interrupts* maskiert sind.
D_Next_Thre Zeit, die das *Kernel* benötigt, um die ead Vorbereitungen für den Start des IST zu treffen.

Abschätzung einer Obergrenze mit Testwerten für Value2.

Gemessene Werte für	Maximal Werte in Mikrosekunden
D_K_Call	266
D_Next_Thread	237
Summe	503

D Schlußwort

Der Erfolg von Win CE ist bislang mäßig. 1999 wurden ungefähr 800.000 Win CE Geräte verkauft. Bei den Windows NT- und 9x Varianten wurden im selben Zeitraum Stückzahlen von über 100 Millionen realisiert. Bislang existieren ungefähr 500 kommerzielle Applikationen und trotzdem konnte kein breites Spektrum an Abnehmern gefunden werden. Das PalmOS belegte 1998 (1999) den ersten Platz mit 73% (80%) vor Win CE mit 14% (13%) bei *Palmtop* Rechnern. Das PalmOS ist kleiner und die Lebenserwartung von Batterien ist deutlicher länger als bei Win CE Geräten. Im „*verticale devices*“ Markt, das ist der Markt für kommerzielle Anwendungen wie *Pen Notepads*, die beispielsweise für die Lagerabrechnung eingesetzt werden, beträgt der Marktanteil von Win CE 2.1 % in 1999. Dieser soll allerdings bis 2002 auf 9% ausgebaut werden. Bei Handhelds mit Tastatur beträgt der Marktanteil sogar 91 %. Allerdings handelt es sich hier um einen sehr kleinen Markt, und die Funktionalität von Win CE läßt für diese Anwendungen noch zu wünschen übrig. Eine besonders starke Konkurrenz stellt NT Embedded dar. Bei Microsoft glaubt man allerdings weiter an den Erfolg von Win CE.

Schließlich wurde Win NT für leistungsfähigere Geräte konzipiert, und es gibt kein Betriebssystem auf dem Markt, welches für jedes Gerät geeignet ist. Insbesondere beim kommerziellen Einsatz bei kleineren Geräten, die als *Client* eingesetzt werden, sieht Microsoft ausbaufähige Chancen für Win CE. NT Embedded wurde hingegen mit Serverfunktionalitäten ausgestattet, wodurch eine Integration mittels Win32 Schnittstelle einfach wäre. Auch Microsofts Konkurrenz hält CE für einen wesentlichen Bestandteil des Marktes. Oft wird die Unterstützung von Visual Basic, ActiveX und die Übertragbarkeit des Systems auf unterschiedlichste Plattformen als wesentlicher Vorteil von Win CE angesehen.

Win CE wird von Anwendern, Konkurrenten und Microsoft als kompetentes Produkt angesehen. Die Marktausrichtung Microsofts war bislang wenig durchdacht, um in den Kernmärkten von Win CE zusätzliche Marktanteile zu sichern, da hier die Anforderungen, die gestellt werden, noch nicht erfüllt werden. Die Version 3 sorgt derzeit für Aufsehen. Einige Marktspezialisten versprechen sich viel von dieser dritten Version, in der Microsofts Betriebssysteme bereits ihre entscheidende Würze erhielten.[7]

E Glossar

API	Application Programmers Interface
ATL	Active Template Library
CD	Compact Disk
COM	Component Object Model
CPU	Central Processing Unit
DDI	Device Driver Interface
DDSI	Device Driver Service Provider Interface
DLL	Dynamic-Link Library
Exe	Execute
EZBS	Echtzeitbetriebssystem
EZS	Echtzeitsystems
FAT	File Allocation Table
FIFO	First In First Out
FTL	Flash Translation Layer
GB	Giga Byte
GDI	Graphics Device Interfaces
GWES	Graphics, Window und Event Manager Subsystem
IP	Internet Protokoll
IRDA	Infrared Data Association
ISH	Interrupt Service Handler
ISR	Interrupt Service Routine
IST	Interrupt Service Thread
IuK	Informations- und Kommunikationstechnologie
KB	Kilo Byte
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LFM	Linear Flash Memory

MB	Mega Byte
MDD	Model Device Driver
MFC	Microsoft Foundation Classes
MHz	Mega Hertz
NDIS	Network Driver Interface Specification
NT	New Technology
OAL	OEM Adaption Layer
OEM	Orginal Equipment Manufacturer
OS	Operating System
OSI	Open System Interconnection
PC	Personal Computer
PDD	Platform-Dependent Driver
PPP	Point to Point Protokoll
RAM	Random Access Memory
ROM	Read Only Memory
TCP	Transmission Control Protokoll
TrueFFS	True Flash File System
USB	Universal Serial Bus
Win CE	Windows Consumer Electronics
XIP	Execute In Place

F Literaturverzeichnis

Artikel

- [1] o.V.
Microsoft Windows CE and Windows NT for Embedded Systems, 10/1998
<http://www.microsoft.com/windowsce/embedded/resources/cente.asp>
- [2] o.V.
Working With Processes and Threads, 7/1997
<http://www.microsoft.com/windowsce/embedded/resources/threads.asp>
- [3] o.V.
Understanding Modularity in Microsoft Windows CE, 6/1998
<http://www.microsoft.com/windowsce/embedded/resources/modularity.asp>
- [4] o.V.
Writing Device Drivers for Windows CE, 12/1998
<http://www.microsoft.com/windowsce/embedded/resources/devicedr.asp>
- [5] Jason Black, Sridhar Mandyam, 2/1997
Universal Serial Bus Support for Windows CE
<http://www.microsoft.com/windowsce/embedded/resources/usb.asp>
- [6] Jason Black, Sridhar Mandyam, 10/1997
Linear Flash Memory Devices on Windows CE
<http://www.microsoft.com/windowsce/embedded/resources/ftl.asp>

[es/ftl.asp](http://www.microsoft.com/windowsce/embedded/resources/ftl.asp)

- [7] James Geoffery, 10/1999
Windows CE: Problem child or late blommer?
<http://cnn.com/TECH/computing/9910/19/wince/index.html>
- [8] Sridhar Mandyam, 1/1997
Network Driver Interface Specification (NDIS) in Windows CE
<http://www.microsoft.com/windowsce/embedded/resources/ndis.asp>
- [9] John Murray, 10/1998
Real-Time Systems with Microsoft Windows CE
http://www.msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/backgrnd/html/msdn_rtdraft6.htm
- [10] David Pellerin, 4/1997
The Microsoft Win32 Programming Model: A Primer for Embedded Software Developers
http://www.msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/backgrnd/html/msdn_w32model.htm
- [11] Guy M. Smith, Kelly Horn, 3/1997
The Windows CE Communications Model
<http://www.microsoft.com/windowsce/embedded/resources/comm.asp>
- Microsoft Online Seminare
Microsoft Professional Developers Conference 1998
<http://www.msdn.microsoft.com/training/seminars/WinDev.asp#WinCE>
- [12] Douglas Boling, David Solomon
Windows CE Programming
- [13] Douglas Boling, David Solomon
Kernal Programming
- [14] Sridhar Mandyam, Raz Dan., Bill Grundmann
Flash Memory Support
<http://www.intel.com/>
<http://www.m-systems.com/>
- [15] Mike Thomson
Inside Windows CE
- [16] James Y. Wilson
Windows CE Device Driver Development
- [17]
Windows CE 2.12 Developer Documentation, 10/1997
<http://www.microsoft.com/windowsce/embedded/download/212doc.asp>