EuroDesign

EuroDesign
embedded technologies GmbH

SolidCard-Familie

HyperBoot-Lader

HBL-03072001-A

Ausgabe I
Stand Juli 2001

Copyrights und Warenzeichen

Inhalt

Das Konzept des HyperBoot-Laders

Ein Standard-PC nutzt nach dem Einschalten das sogenannte BIOS (Basic In Output System). Dieses hat die Aufgabe, die verfügbaren Resourcen wie CPU-Taktgeschwindigkeit, Speichertyp und -menge und die angeschlossenen Peripheriegeräte zu erkennen und zu initialisieren. Dieser Vorgang ist für Standard-PCs durchaus sinnvoll, soll doch ein solcher Rechner auch bei Änderung der Konfiguration weiterhin starten und benutzbar bleiben. Da jedoch die möglichen Konfigurationen nahezu beliebig sein können, muß das BIOS alle erdenklichen Varianten erkennen und prüfen, um die Initialisierung erfolgreich durchführen zu können. Dieser Vorgang benötigt daher eine geraume Zeit (10 bis 20 Sekunden), unabhängig von der Geschwindigkeit der vorgefundenen CPU. Der Zeitraum, bis zu dem das Gesamtsystem benutzbar ist, ist entprechend lang.

In einigen Fällen schlägt die automatische Initialisierung aber auch fehl und der Anwender muß manuell eingreifen. Zu diesem Zweck besitzt das Standard-BIOS das sog. Setup, mit dem einige Parameter verändert werden können, die die Initialisierung beeinflussen bzw. in manchen Fällen erst ermöglichen.

In einem Embedded PC ist die Konfiguration der Hardware dagegen immer gleich: Weder die CPU wechselt, noch der Speichertyp bzw. die -menge. Auch wechselt die angeschlossene sonstige Peripherie beispielsweise nur im Reperaturfall. Die Erkennung der Konstellation ist also überflüssig und benötigt nur unnötige Zeit.

Hier setzt unser HyperBoot-Lader an: Das System wird für die vorgesehene Konfiguration initialisiert. Dies benötigt nur einen Bruchteil der Zeit, den die automatische Erkennung erfordern würde. In Zahlen ausgedrückt: Die reine Initialisierung dauert nur noch zwischen 0,030 und 0,1 Sekunden. Dann kann das Betriebssystem bereits geladen und gestartet werden.

Da die Konfiguration immer gleich ist, stellt der HyperBoot-Lader auch kein Menü bereit, welches einem Setup vergleichbar wäre.

Dennoch kann es für Einwicklungs-, Wartungs- oder Reperaturzwecke sinnvoll sein, in den Bootvorgang einzugreifen. Dafür stellt der HyperBoot-Lader einen Wartungs-Modus bereit, in dem der normale Boot-Vorgang verzögert, bzw. verändert werden kann.

Da oftmals in einem Embedded PC kein Bildschirm benötigt wird, ist der HyperBoot-Lader auch über die serielle Schnittstelle bedienbar. Wenn der HyperBoot-Lader keine Grafikkarte findet, nutzt er automatisch die serielle Schittstelle für alle Ein- und Ausgaben. Wird eine Grafikkarte, aber keine Tastatur gefunden, erfolgen die Ausgaben auf den Bildschirm, die Eingabe erfolgt über die serielle Schnittstelle.

Der HyperBoot-Lader kann primär das freie Betriebssystem Linux booten. Es ist aber auch möglich andere Software zu booten, beispielsweise Programme die ohne ein Betriebsystem auskommen. Nicht möglich dagegen ist das Booten von Betriebssystemen, die ein Standard-BIOS voraussetzen. Die Funktionalität eines Standard-BIOS wird vom HyperBoot-Lader nicht unterstützt.

Im Gegensatz zu einem Standard-BIOS, arbeitet der HyperBoot-Lader bereits in der x86-Betriebsart Protected-Mode.

Anwendung des HyperBoot-Laders

Sie werden den HyperBoot-Lader in der Regel kaum bemerken. Er läuft nur sehr kurzzeitig und übergibt dann die Kontrolle an das geladene Betriebssystem.
Um diesen schnellen Boot-Vorgang zu realisieren, muß jedoch zuvor das Boot-Medium festgelegt, vorbereitet und initialisiert sein.

Wenn ein Bildschirm angeschlossen ist, können Sie den Ablauf direkt verfolgen. Findet der Hyperboot-Lader keinen Grafikcontroller, verwendet er automatisch die serielle Schnittstelle COM1 mit den folgenden Parametern:

- 19200 Baud
- 8 Datenbits
- 1 Stopbit
- kein Paritätsbit
- kein Protokoll

Wird ein Grafikcontroller gefunden, aber keine Tastatur, werden die Ausgaben auf den Grafikcontroller ausgegeben, die Eingaben werden jedoch von der seriellen Schnittstelle erwartet.

Der HyperBoot-Lader ist in der Lage ein Standard-Linux-Binärabbild zu laden oder auch eine HyperBoot-Binärdatei.
Letztere ermöglicht es, neben dem Kernel auch eine Kommandozeile und eine RAM-Disk mitladen zu können. Die HyperBoot-Binärdatei ist nicht auf Linux eingeschränkt, sondern kann beliebige Programme laden.

Wenn eine Grafikkarte und ein Bildschirm vorhanden ist, zeigt der HyperBoot-Lader nach dem Einschalten des Systems das folgende Bild:

```
                EuroDesign
          embedded technologies GmbH
        www.eurodsn.de info@eurodsn.de
          Tel.: +49 (0)8166/99495-80
_____________________________________________
Warten auf IDE-Festplatte....
Festplattentyp: DC108
Kapazitaet: 540 MByte (LBA)
497kB Abbildgroesse
.................................
```

In diesem Fall war eine IDE-Festplatte zum Booten vorgesehen. Die Punkte zeigen den Fortschritt des Ladevorgangs an.
Zu beachten ist, daß ein Boot-Vorgang von einer Festplatte ggf. durch das Hochlaufen der Festplatte selbst verzögert werden kann.
Nach dem Laden des Betriebssystems beginnt sofort dessen Start, die Arbeit des HyperBoot-Laders ist damit beendet.

Einrichten der Boot-Medien

Damit der HyperBoot-Lader ein Betriebssystem laden kann, muß das vorgesehene Medium dafür vorbereitet werden.
Die folgenden Medien sind als Quellen für einen Boot-Vorgang geeignet:

- 1,44 MB Disketten
- IDE Festplatten oder kompatible FlashDisk-Laufwerke
- DiscOnChip-Flash
- Flash-ROM

Bei allen Boot-Medien wird erwartet, daß das Binärabbild linear geschrieben wird. D.h. es darf dort kein Dateisystem vorhanden sein. Bis auf die Diskette und das Flash-ROM ist es also erforderlich, das jeweilige Medium zu partitionieren, um auch ein normales Dateisystem benutzen zu können.
Dabei werden mindestens 2 Partitionen angelegt:

- eine für das Binärabbild des Betriebssystems (in der Regel um die 1 MB groß)
- eine weitere für das Dateisystem

Das Binärabbild wird linear in die erste Partition geschrieben, die zweite Partition wird wie gewohnt mit dem gewünschten Dateisystem formatiert.

Hier ein Beispiel anhand einer IDE-Festplatte:

Aus Sicht von Linux ist diese IDE-Festplatte das Gerät „hda".
Mit

```
fdisk /dev/hda
```

können Sie die Festplatte nun partitionieren.

Legen Sie eine erste primäre Partition an, wählen Sie als Größe etwa 1 MB. Legen Sie dann eine zweite primäre Partition an, die beispielsweise den Rest der Platte belegt.
Die erste Partition ist von Linux aus als „hda1", die zweite Partition als „hda2" ansprechbar.
Das Betriebssystem-Binärabbild erhält als Root-Device das Gerät „hda2" eingetragen.
Es kann dann mit

```
dd bs=8192 if=vmlinux of=/dev/hda1
```

linear auf die erste Partition geschrieben werden.
Mit

```
mkfs.ext2 /dev/hda2
```

erstellen Sie auf der zweiten Partition ein Dateisystem (in diesem Fall das „Extended 2"-Format). Jetzt können Sie alle sonstigen vom Betriebssystem benötigten Dateien auf dieses Dateisystem kopieren.

Die prinzipiell gleiche Vorgehensweise ist bei der Verwendung einer DiskOnChip notwendig. In diesem Fall handelt es sich aber aus Sicht von Linux um das Gerät „nftla", bzw. die beiden Partitionen „nftla1" und „nftla2".

Abweichende Nutzung bei 1,44 MB Disketten und Flash-ROM

1,44 MB Disketten werden nicht partitioniert. Sie enthalten das Standard-Linux- oder das HyperBoot-Binärabbild direkt linear.
Mit

```
dd bs=8192 if=vmlinux of=/dev/fd0
```

kann jederzeit ein neues System auf Diskette geschrieben werden und anschließend vom HyperBoot-Lader geladen werden.

Das Flash-ROM ist anders organisiert, so daß sein Inhalt nicht von Linux aus verändert werden kann. Um hier Änderungen am
Inhalt vornehmen zu können, müssen Sie auf die Funktionalität des integrierten Debuggers im HyperBoot-Lader zurückgreifen.
Lesen Sie hierzu die Angaben im Kapitel „HyperBoot aus dem Flash-ROM".

Auswahl des Bootmediums

Nachdem Sie ein oder mehr Medien für den Boot-Vorgang vorbereitet haben, müssen Sie es nun auswählen, so daß der HyperBoot-Lader darauf zugreift. Dazu bringen Sie den HyperBoot-Lader zunächst in den Wartungsmodus.
Wenn Sie unser Evaluation-Kit verwenden, müssen sie den Microschalter für die Wartungs-Funktion jeweils in eine bestimmte Stellung bringen. Schlagen Sie diese Information im zugehörigen Handbuch zu Ihrem Evaluation-Kit nach.

In dieser Betriebsart ändert sich der Inhalt im Startbildschirm. Das System bootet jetzt nicht sofort, sondern zeigt Ihnen eine Auswahl an gefundenen Binärabbildern an und wartet darauf, daß Sie eines davon zum Booten auswählen.

Wurde kein bootbares Binärabbild gefunden, ergibt sich der nachfolgende Bildschirm:

```
                EuroDesign
          embedded technologies GmbH
        www.eurodsn.de info@eurodsn.de
          Tel.: +49 (0)8166/99495-80
_____________________________________________

Es steht kein ladbares Abbild zur Verfuegung!
Abbild voreinstellen: d <Eintrag>
Debugger starten: q
Quellen freischalten: e
Einstellungen sichern: s
>
```

In diesem Fall wurden entweder noch keine Binärabbilder auf die Medien aufgespielt, oder das jeweilige Medium ist noch nicht freigeschaltet worden.

Wenn Binärabbilder gefunden wurden, ändert sich der Inhalt des Startbildschirms:

```
                EuroDesign
          embedded technologies GmbH
        www.eurodsn.de info@eurodsn.de
          Tel.: +49 (0)8166/99495-80
_____________________________________________

Abbild: 0   Standard-Image -> Floppy
Abbild: 1   Demo-System -> IDE

Abbild laden: r <Abbild>
Abbild voreinstellen: d <Eintrag>
Debugger starten: q
Quellen freischalten: e
Einstellungen sichern: s
>
```

Hier wurde auf der eingelegten Diskette und auf der Festplatte ein ladbares Abbild gefunden. Mit der Eingabe

```
r <Abbildnummer>
```

kann dieses nun geladen werden.

```
r 0
```

lädt also den Kernel von Diskette,

```
r 1
```

von Festplatte aus Partition 1.

Wollen Sie nun das Abbild auf der angeschlossene IDE-Festplatte zukünftig als Standard sofort laden lassen, wählen Sie den Menüpunkt

```
d <Abbildnummer>
```

Damit diese Einstellung gesichert wird, müssen Sie anschließend noch

```
s
```

eingeben. Machen Sie jetzt einen Funktionstest, indem Sie die Reset-Taste drücken. Sie erhalten nun einen Bildschirm mit dem gleichen Inhalt wie zuvor, jedoch hat der IDE-Eintrag jetzt ein vorangestelltes „*"-Symbol. Damit zeigt Ihnen der HyperBoot-Lader an, daß dieses Binärabbild sofort geladen würde, wenn er nicht mehr im Wartungs-Modus ist.

Wird Ihr Binärabbild nicht gefunden, kann es auch sein, daß dieses Medium nicht freigeschaltet ist. D.h. der HyperBoot-Lader durchsucht dieses Medium nicht nach Binärabbildern und listet die dort befindlichen Binärabbilder auch folglich nicht auf.

Geben Sie in diesem Fall das Kommando

```
e
```

ein. Jetzt erhalten Sie eine Liste der möglichen Medien:

```
                    EuroDesign
              embedded technologies GmbH
            www.eurodsn.de info@eurodsn.de
              Tel.: +49 (0)8166/99495-80
_____________________________________________________

Abbild: 0   Standard-Image -> Floppy
Abbild: 1   Partition 0 -> IDE

Abbild laden: r <Abbild>
Abbild voreinstellen: d <Eintrag>
Debugger starten: q
Quellen freischalten: e
Einstellungen sichern: s
>e
1=Floppy sperren
2=IDE sperren
3=DOC freischalten
4=Flash-ROM freischalten
```

Sie können nun die Quellen durch Eingabe einer der angegebenen Ziffern entweder sperren, wenn sie freigeschaltet oder freischalten, wenn sie gesperrt war. Eine hier vorgenommene Änderung wird erst beim nächsten Systemstart oder RESET berücksichtigt. Dazu muß sie aber über die Eingabe

```
s
```

gesichert werden, sonst geht die Änderung verloren.
Haben Sie jetzt Ihr gewünschtes Medium freigeschaltet und das Binärabbild dort korrekt abgelegt, wird es zukünftig gefunden und kann von dort aus geladen werden.

Beenden Sie den Wartungs-Modus, damit der HyperBoot-Lader zukünftig den schnellen Boot-Vorgang durchführt.

Zusammenfassung der Kurzkommandos

r <Nr>:
lädt eines der darüber aufgelisteten Abbilder

d <Nr>:
wählt eines der darüber aufgelisteten Abbilder als standardmäßig zu ladendes. Dieses wird dann zukünftig sofort geladen.

e:
erzeugt eine Liste mit den möglichen Medien, von denen ein Abbild geladen werden kann. Das Medium wird per Ziffer umgeschaltet (sperren oder freigeben).

s:
Mit e durchgeführte Änderungen dauerhaft sichern.

q:
Integrierten Debugger starten.

Sonstige Funktionalität des HyperBoot-Laders

Befindet sich das System im Wartungs-Modus, können Sie von der reinen Auswahl eines zu bootenden Binärabbildes zusätzlich über die Eingabe

```
q
```

in den integrierten Debugger wechseln. Dieser stellt ihnen einige grundlegende Funktionen bereit, um auf die Hardware des Systems zugreifen, bzw. Informationen abfragen zu können, ohne daß ein Betriebssystem laufen muß.

Aus diesem Debugger können Sie nur mittels Neustart des Systems wieder zurück in den HyperBoot-Lader.

Wenn Sie

```
?
```

eingeben, erhalten Sie eine Liste der möglichen Befehle. Je nach SolidCard erhalten Sie unterschiedliche Angaben. Die hier dargestellte Liste stammt von einer SolidCard II.

```
Anzeigen              D [Startadresse] [Endadresse]
Eingeben              E  Startadresse Wert [Wert]
E/A Einlesen          I  Adresse
E/A Schreiben         O  Adresse Wert
Füllen                F  Adresse Wert Anzahl
Speichertest          T  <Startadresse> <Laenge>
                         <Anzahl Versuche>
Lesen                 R  Bereich Adresse
                      5  <bwd> SC520 MMCR
Schreiben             W  Bereich Adresse Wert
                      5  <bwd> SC520 MMCR
PAR-Inhalt            PA  [l]
PCI-Config anzeigen   PI
Block laden IDE       LI <Sektornummer>
Block laden FDC       LF <Sektornummer>
Linux-Abbild:
Von Disk in Flash     LS
Liste anzeigen        LL
Im Flash löschen      LE <Nummer>
Flash komprimieren LC
Alle Adressen sind linear und hexadezimal anzugeben!
```

Parameter in Spitzen Klammern ('<>') sind erforderlich, in eckigen Klammern ('[ ]') optional.

Anzeigen eines Speicherbereichs

Kommando: `D [Startadresse [Endadresse]]`

Zeigt den Speicherbereich beginnend von Startadresse bis Endadresse in Tabellenform an. Wird die Endadresse weggelassen,
werden 256 Bytes ab der Startadresse angezeigt. Werden beide Parameter weggelassen, werden die nächsten 256 Bytes angezeigt,
beginnend ab der Endadresse des vorhergehenden Aufrufs.

Eintragen von Datenwerten in einen Speicherbereich

Kommando: `E <Startadresse> <Wert> [Wert2] ...`

Legt Wert ab der Startadresse im Speicher ab. Es kann mehr als ein Wert angegeben werden. Mit jedem Wert wird die Startadresse um eins erhöht.

Einlesen eines Bytes aus dem I/O-Adressraum
Kommando: `I <Adresse>`
Liest aus Adresse ein Byte und zeigt diesen Wert an. Adresse darf im Bereich 0...FFFF liegen.

Schreiben eines Bytes in den I/O-Adressraum
Kommando: `O <Adresse> <Wert>`
Schreibt das Byte Wert in Adresse. Adresse darf im Bereich 0...FFFF und Wert im Breich 0...FF liegen.

Füllen eines Speicherbereichs
Kommando: `F <Adresse> <Wert> <Anzahl>`
Füllt den Speicher ab Adresse mit Wert. Gefüllt wird bis Adresse+Anzahl-1.

Testen eines Speicherbereichs
Kommando: `T <Startadresse> <Länge> <Anzahl Wiederholungen>`

Testet einen Speicherbereich, ob sich darin beliebige Bitmuster ablegen lassen. Der Test wird ab Startadresse begonnen, wird auf die Länge durchgeführt und dies mit der Anzahl der Wiederholungen. Wird die Anzahl Wiederholungen mit 0 angegeben, wird der Test unendlich oft wiederholt.

Funktionsprinzip des Tests: Der Speicherbereich wird in 32 Bit Langwörter eingeteilt. Zunächst wird in jedes Langwort ein aufsteigender Wert beginnend mit 0 geschrieben. Danach wird dieser Speicherbereich auf diese Werte geprüft. Sind Wiederholungen vorgesehen, wird der Speicherbereich wiederum mit aufsteigenden Langwörtern beschrieben, diesmal jedoch mit 1 beginnend. Bei jeder Wiederholung wird der Startwert um eins erhöht. Werden unendliche Wiederholungen gefordert, wird jede Speicherstelle also mit jedem möglichen Muster beschrieben und geprüft. Adressbitfehler (Lötfehler etc.) werden auf diese Art sicher erkannt.

Der Test wird bei einem Fehler abgebrochen und die fehlerhafte Speicheradresse ausgegeben. Beachten Sie bitte, daß die heutigen Prozessoren einen Cache besitzen. Damit wirklich der externe Speicher geschrieben und gelesen wird, muß die Länge die Größe des Caches übersteigen. Andernfalls würde nur der Cache, nicht aber der externe Speicher geprüft.

Zusammengefaßtes Lesen (SolidCard 1)
Kommando: R <Area> <Offset>

Um einige Konfigurations-Register lesen zu können, müssen bestimmte Operationen ohne Unterbrechung durchgeführt werden.
Dies stellt der Befehl R bereit. Er liest einen 8 Bit Wert aus dem Offset des angegeben Bereichs. Area teilt dem Debugger
mit, welcher Bereich ausgelesen werden soll:

4	SolidCard 1 Chipsatz-Register
C	SolidCard 1 Grafikregister der CGA-Emulation
M	SolidCard 1 Grafikregister der MDA-Emulation

Beispiel: R 4 0 liest das Bank 0 DRAM-Konfigurationsregister aus
R C E liest die höherwertige Adresse des CGA-Cursors

Zusammengefaßtes Lesen (SolidCard 2)
Kommando: R 5 <b | w | d> <Offset>
Liest ein Byte (8 Bit), Wort (16 Bit) oder Doppelwort (32 Bit) aus dem „Memory Mapped Configuration Register"-Bereich des SC520-Chipssatz.
Da die Register in diesem Bereich keine einheitliche Datenbreite aufweisen,

muß mit einem Parameter diese Breite vor dem eigentlichen Offset angegeben werden. Der Offset ist der MMCR-Offset des Datenblatts. Die echte Zugriffsadresse (ab FFFE000016) wird automatisch ergänzt.

Beispiel: R5 b 2 liest ein Byte aus MMCR-Offset 2 (Cache-Einstellung)
R5 w 0 liest ein Wort aus MMCR-Offset 0 (Chiprevision)
R5 d 8C liest ein Doppelwort aus MMCR-Offset 8C (PAR1)

Zusammengefaßtes Schreiben (SolidCard 1)
Kommando: W <Area> <Offset> <Wert>

Um einige Konfigurations-Register schreiben zu können, müssen bestimmte Operationen ohne Unterbrechung durchgeführt werden. Dies stellt der Befehl W bereit. Er schreibt einen 8 Bit Wert in Offset des angegeben Bereichs. Area teilt dem Debugger mit, welcher Bereich geschrieben werden soll:

4	SolidCard 1 Chipsatz-Register
C	SolidCard 1 Grafikregister der CGA-Emulation
M	SolidCard 1 Grafikregister der MDA-Emulation

Beispeil: W 4 80 schreibt das Bank 0 DRAM-Konfigurationsregister aus
W C E schreibt die höherwertige Adresse des CGA-Cursors

Zusammengefaßtes Schreiben (SolidCard 2)
Kommando: W5 <b | w | d> <Offset> <Wert>

Schreibt ein Byte (8 Bit), Wort (16 Bit) oder Doppelwort (32 Bit) in den „Memory Mapped Configuration Register"-Bereich des SC520-Chipssatz. Die Parameter sind wie bei „Zusammengefaßtes Lesen" zu werten, nur daß hier noch ein Wert hinzukommt.
Unabhängig von der Breite von Wert, wird die im ersten Parameter angegebene Breite beim Schreiben verwendet. D.h. beispielsweise FF als Wert wird bei d im ersten Parameter als Doppelwort geschrieben (000000FF).

Aktuellen Zustand der PAR-Register anzeigen (SolidCard 2)
Kommando: PA [l]

Zeigt die aktuellen Werte der SC520-PAR-Register an. Diese werden nicht in binärer, sondern in lesbarer Form ausgegeben.
Ohne den Parameter l wird eine kurze Tabelle angezeigt, mit l eine ausführliche Liste mit allen Einstellungen.

Geräte am PCI-Bus anzeigen (SolidCard 2)
Kommando: PI

Listet die am PCI-Bus angeschlossenen Geräte und zeigt deren Resourcen wie Speicher- und E/A-Bereiche und Interruptkanäle.

Einen Sektor von einer IDE-Festplatte lesen und anzeigen
Kommando: LI <Sektornummer>

Liest von einer IDE-Festplatte (Master) den angegebene Sektor (im LBA-Format) und zeigt dessen Inhalt am Bildschirm an.

Einen Sektor von einer 1,44MB Diskette lesen und anzeigen
Kommando: LF <Sektornummer>

Liest von einer Diskette den angegebene Sektor (im LBA-Format) und zeigt dessen Inhalt am Bildschirm an.

Die meisten Eigenschaften sollten selbsterklärend sein. Wenden Sie sich an unseren Support (info@eurodsn.de), wenn Sie Fragen zu diesen Kommandos haben.

HyperBoot aus dem Flash-ROM

Neben dem Boot-Vorgang aus einer IDE-Flashdisk oder einer DiscOnChip, ist der Systemstart aus dem Flash-ROM eine der schnellsten Methoden, ein Betriebssystem zu starten. Es handelt sich dabei um ein 2 MB NOR-Flash, welches linear in den Adressraum der CPU eingeblendet ist und damit einen sehr schnellen Zugriff ermöglicht. Dieses Bauteil ist eine Option.
Wenn Sie es nutzen möchten, teilen Sie uns das bitte vor dem Kauf der SolidCard 2 mit.

Um das Flash-ROM für den Bootvorgang nutzen zu können, müssen Sie zunächst den Wartungs-Modus aktivieren. Dann wechseln sie mit

```
q
```

in den integrierten Debugger. Um den Inhalt des Flash-ROM zu manipulieren sind die Befehle

- `LS`
- `LL`
- `LE` und
- `LC`

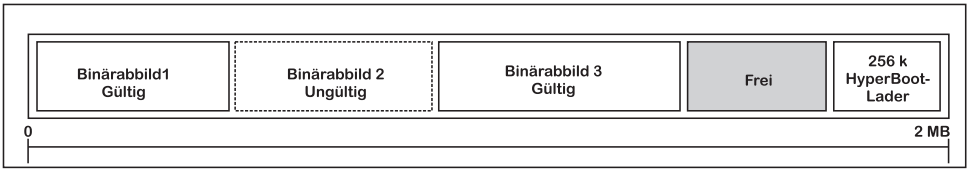
vorgesehen.

LL: Listet alle vorhandenen Binärabbilder auf und gibt den verbleibenden Platz im Flash-ROM aus.

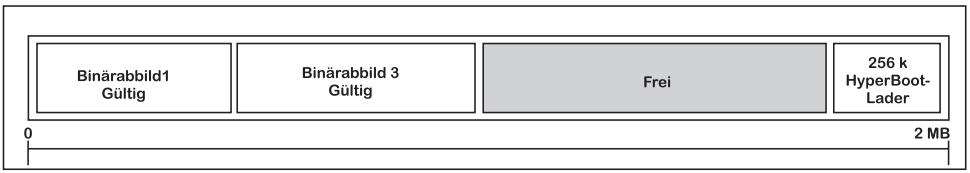
LS: Liest ein Binärabbild von einer 1,44 MB Diskette und schreibt es an den freien Platz im Flash-ROM. Reicht der Platz im Flash-ROM nicht mehr aus, erfolgt eine Fehlermeldung.

LE: Löscht ein vorhandens Binärabbild im Flash-ROM. Dabei wird das Binärabbild nur als ungültig definiert, noch nicht gelöscht. Es kann jedoch nicht wieder gültig erklärt werden.

LC: Faßt gültige Binärabilder und freie Bereiche im Flash-ROM zusammen. Dies ist erforderlich, wenn Sie beispielsweise aus 3 vorhandenen Binärabbildern das Zweite in der Liste löschen. Da neue Abbilder nur an das Ende der Liste angehängt werden, kann es trotz Löschung sein, daß nicht genug Platz für das neue Binärabbild vorhanden ist. Die folgende Grafik soll dies verdeutlichen: Binärabbild 2 ist ungültig, zusammen mit dem freien Bereich wäre genug Speicher für ein weiteres Binärabbild vorhanden.



Nach Durchführung des Befehls `LC` ergibt sich folgendes Bild:



Jetzt ist genug freier zusammenhängender Speicher für ein weiteres Binärabbild vorhanden.

Erstellung bootfähiger Binärabbilder

Der HyperBoot-Lader kann mit zwei Arten von binären Abbildern umgehen:

- Standard-Linux-Binärabbild
- HyperBoot-Binärformat

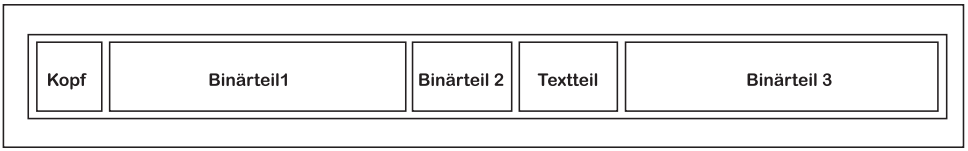
Beide Arten können von Disketten, DiscOnChip und Festplatten gelesen werden, von Flash-ROM nur das HyperBoot-Binärformat.

Das Standard-Linux-Binärabbild erstellen Sie wie gewohnt. =

```
make zImage oder make bzImage
```

erstellt die notwendige und nutzbare Datei. Nachteil dieser Methode ist es, daß nur der eigentliche Kernel geladen wird und auch keine Kernel-Kommandos übergeben werden können. Das immer notwendige Root-Dateisystem muß in diesem Fall bereits vorhanden sein.

Das HyperBoot-Binärformat ermöglicht neben dem Laden des Kernels auch das Laden einer Kommandozeileninformation und einer RAM-Disk. Zu diesem Zweck hat eine



solche Binärdatei den folgenden Aufbau:
Die einzelnen Elemente (bei einem Linux-System):

- Kopf: enthält Erkennungs- und verwaltungstechnische Informationen
- Binärteil 1: Enthält den reinen Kernel (ohne Setup etc.)
- Binärteil 2: Enthält den Setupteil des bootbaren Linux-Kernels
- Textteil: Hier liegt der Text, welcher als Kommandozeile an den startenden Kernel übergeben wird
- Binärteil 3: Enthält ein RAM-Disk-Abbild

Den Aufbau des Kopfes erläutert die folgenden Struktur, verfaßt in der Programmiersprache C:

```
struct HYPERBOOT
{       unsigned char mark[20];
        unsigned long ver;
        unsigned char name[160];
        unsigned long totalBlocks;
        unsigned long headerSize;

        unsigned long kernelBlocks;
        unsigned long kernelSize;
        unsigned long kernelTarget;
        unsigned long kernelStart;
        unsigned long kernelFlags;

        unsigned long paramBlocks;
        unsigned long paramSize;
        unsigned long paramTarget;

        unsigned long commandBlocks;
        unsigned long commandSize;
        unsigned long commandTarget;

        unsigned long rootBlocks;
        unsigned long rootSize;
        unsigned long rootTarget;
};
```

Die Inhalte der einzelnen Einträge:

- **mark:** Enthält eine Kennung, die der HyperBoot-Lader auswertet, um die vorgefundene Datei zu klassifizieren. Ein HyperBoot-Binärabbild enthält hier die Zeichenkette „EURODESIGN_HYPERBOOT", jedoch ohne eine abschließende 0!
- **ver:** Enthält eine Versionsnummer, um den Aufbau dieser Struktur zukünftig erweitern zu können. Der Wert 0x00010000 steht für die Version 1.0, 0x00010001 für die Version 1.1
- **name:** Hierüber kann dem Binärabbild ein Titel vergeben werden. Dieser Text wird bei der Auswahl im Wartungsmodus angezeigt.
- **totalBlocks:** Anzahl der von dieser Datei belegten Sektoren à 512 Byte.
- **headerSize:** Anzahl Sektoren, die durch diesen Kopf belegt sind. Derzeit 1.
- **kernelBlocks:** Anzahl Sektoren, die der Kernel belegt.
- **KernelSize:** Größe des Kernels in Byte kernelTarget: Zieladresse des Kernels im Speicher. Lineare 32 Bit-Zahl.
- **kernelStart:** Startadresse, an der der Kernel gestartet werden soll. Lineare 32 Bit-Zahl.
- **kernelFlags:** Informationen über den Kernel. Derzeit ungenutzt, d.h. alle Bit 0.
- **paramBlocks:** Anzahl Sektoren, die der Setup-Teil belegt.
- **ParamSize:** Größe des Setup-Teil in Byte paramTarget: Zieladresse des Setup-Teils im Speicher. Lineare 32 Bit-Zahl.
- **commandBlocks:** Anzahl Sektoren, die der Text der Kommandozeile belegt.
- **commandSize:** Größe des Textes der Kommandozeile in Byte.
- **commandTarget:** Zieladresse der Kommandozeile. Lineare 32 Bit-Zahl.
- **rootBlocks:** Anzahl Sektoren, die die RAM-Disk belegt.
- **rootSize:** Größe der RAM-Disk in Byte
- **rootTarget:** Zieladresse der RAM-Disk im Speicher. Lineare 32 Bit-Zahl.

Der Aufbau ist an die Erfordernisse des Linux-Betriebssystems angepaßt. Jedoch kann auch ein beliebiges anderes Programm damit geladen werden.

Um beispielsweise ein Rettungssystem bestehend aus einer Kommadozeile, einem minimalen Kernel und einer RAM-Disk mit notwendigen Tools in ein HyperBoot-Binäarabbild umzuwandlen, benötigen Sie die folgenden Dateien:

- mImage
  erzeugt aus den nachfolgenden Dateien das Binärabbild
- arch/i386/boot/setup
- arch/i386/boot/bootsect
- arch/i386/boot/compressed/vmlinux.out
- eine Textdatei mit dem Inhalt der Kommandozeile das Abbild einer RAM-Disk

Wenn die Textdatei den Namen „sc2command.txt" und die RAM-Disk den Namen „image.gz" trägt, kann mImage wie folgt aufgerufen werden:

```
mImage -k linuxRAM.out 0x1000 0x1000
       -p bootsect setup /dev/ram0 0x90000
       -c sc2command.txt 0x90800
       -r image.gz 16777216
       -n „Rettungssysten mit RAM-Disk"
       -o sc2rescue.bin
```

Dieser Aufruf erzeugt ein Linux mit einer RAM-Disk als Root-Device, benennt dieses Abbild mit „Rettungssysten mit RAM-Disk" und schreibt das Ergebnis in die Datei „sc2rescue.bin".

Wobei:

`-k <Kernelname> <Zieladresse> <Startadresse>`

Kernelname: Beispielsweise /usr/src/arch/i386/boot/compressed/vmlinux.out

Benutzen Sie nicht „vmlinux", dort ist bereits der Boot-Sektor und der Setup-Teil integriert!
Zieladresse: Bei einem zImage 0x1000, bei einem bzImage 0x100000
Startadresse: Bei einem zImage 0x1000, bei einem bzImage 0x100000

`-p <Boot-Sektor> <Setup> <Root-Device> <Zieladresse>`
Boot-Sektor: Beispielsweise /usr/src/arch/i386/boot/bootsect
Setup: Beispielsweise /usr/src/arch/i386/boot/setup
Root-Device: Name der Gerätedatei, die der Kernel als Root-Device benutzen soll
Zieladresse: Derzeit immer 0x90000

`-c <Kommandodateiname> <Zieladresse>`
Kommandodateiname: Name einer Datei mit ASCII-Inhalt
Zieladresse: Derzeit immer 0x90800

`-r <RAM-Disk-Name> <Ende des Speicherbereichs>`
RAM-Disk-Name: Name der Datei mit dem RAM-Disk-Abbild
Ende des Speicherbereichs: Da das RAM-Disk-Abbild am Ende des verfügbaren Speichers abgelegt werden soll, ist hier die Menge des auf dem Zielsystem verfügbaren Hauptspeichers anzugeben. Beachten Sie, daß es nicht sinnvoll ist, eine solche RAM-Disk bei einem Speicherausbau von nur 2 MB zu betreiben.

`-n <Bezeichnung dieses Binärabbildes>`
Hier können Sie eine Bezeichnung angeben (max. 160 Zeichen). Diese wird beim Laden oder der Auswahl angezeigt

`-o <Dateiname dieses Binärabbildes>`
Über diesen Schalter bestimmen Sie den Namen des zu erzeugenden HyperBoot-Binärabbildes.

Beachten Sie, das zwischen alles Schaltern und deren Parametern ein Leerzeichen stehen muß.

Systemzustand beim Übergang vom HyperBoot-Lader zum Betriebssystem

Der HyperBoot-Lader ist nicht unbedingt allein auf Linux ausgelegt, wobei er jedoch für dieses Betriebssystem konzipiert wurde. Über das HyperBoot-Binärabbild können Sie auch ein anderes Betriebssystem oder auch ein einfaches Programm laden und zur Ausführung bringen. Dieses Kapitel beschreibt, in welchem Zustand der HyperBoot-Lader den Prozessor und die sonstige Hardware an zu startende Betriebssystem/Programm übergibt.

Peripherie:	LPT: Standard. COM1...4: Standard DMA-Controller: Standard Tastatur-Controller: Standard Floppy-Controller: Standard Timer: Standard
Echtzeituhr:	Diese ist aktiv, jedoch enthält dessen CMOS-RAM keine Daten. Der HyperBoot-Lader verwendet das CMOS-RAM nicht.
Interrupt-Controller:	Diese sind aktiv, jedoch so initialisiert, wie Linux sie verwendet.
Prozessor:	Der Betriebsmodus des HyperBoot-Laders ist der Protected-Mode. Die IDT beginnt an der physikalischen Adresse 0x00032 mit der Länge 1024 Byte, die GDT an Adresse 0x0000C mit der Länge 32 Byte
Speicher:	Der gesamte Speicher steht einer Anwendung zur Verfügung. Sofern der Chipsatz es unterstützt, existiert die klassische Lücke im Bereich 0xA0000....0xFFFFF nicht mehr, auch in diesem Bereich liegt dann normaler Arbeitsspeicher. Sämtliche speicheradressierbaren Peripheriegeräte liegen oberhalb des Arbeitsspeichers.

Glossar

Boot-Medien: Gemeint sind Disketten, Festplatten, ROM-Speicher, von denen ein Programm oder Betriebssystem geladen werden kann.

Boot-Vorgang: Laden und Starten einer Arbeitsumgebung, beispielsweise eines Betriebssystems.

Dateisystem: Durch das Prinzip von Sektoren müssen Verfahren vorhanden sein, Dateien, die mehr als einen Sektor benötigen logisch zu verwalten. D.h. es werden Strukturen benötigt, die das Wiederherstellen einer Datei aus der Vielzahl von Sektoren sicherstellen. Es existieren eine ganze Reihe von Dateisystemen, jede mit speziellen Eigenschaften.

DiscOnChip: Eine NAND-Flash-Technologie der Firma MSystems - ermöglicht die Verwendung eines Dateisystems ohne mechanische Teile. Festplattenersatz.

HyperBoot: Besondere Art der Initialisierung und Starten eines PC-Systems - verzichtet auf die Verwendung eines üblichen BIOS.

HyperBoot-Binärdatei: Ist eine aus mehreren Teilen zusammengesetzte Binärdatei, die alle notwendigen Teile des Betriebssystems enthält, für den Bootvorgang selbst aber keinen aktiven Bootlader enthalten muß (siehe Standard-Linux-Binärabbild). Das Kopieren vom Bootmedium in den Arbeitsspeicher erfolgt vollständig durch den HyperBoot-Lader.

Kernel: Dies ist die Bezeichnung eines Programms und seiner Daten, welches das eigentliche Betriebssystem darstellt.

Kommandozeile: Wortweise Kommandos, die ein Linux-Kernel verarbeiten kann. Dient der Konfiguration des Kernels.

Linear: Disketten und Festplatten legen ihre Daten in in nummerierten Sektoren auf ihrem Speichermedium ab. Dadurch müssen Dateien die mehr Daten enthalten als in einem Sektor Platz ist auf mehrere Sektoren verteilt werden. Linear bedeutet, daß die Daten aufsteigend auch in aufsteigend nummerierten Sektoren abgelegt werden. Um die Datei zurückzugewinnen, genügt es, die Sektoren von Sektor 0 an aufsteigend zu lesen. Bei Verwendung eines Dateisystems existiert dieser lineare Zusammenhang nicht.

NAND-Flash: Blockorientierter ROM- Speicher. Geeignet als Medium zum Sichern von Daten. Ein Prozessor kann ein darin befindliches Programm nicht direkt ausführen, sondern nur blockweise in den Arbeitsspeicher auslesen und dort ausführen.

NOR-Flash: Kann von einem Prozessor linear angesteuert und daher als Programmspeicher verwendet werden. Hat die Eigenschaft eines ROM (verliert seinen Inhalt auch bei Spannungsverlust nicht), kann jedoch neu beschrieben werden.

Partitionen: Wenn eine Festplatte in mehrere unabhängige Bereiche aufgeteilt wird, werden die einzelnen Bereiche Partitionen genannt. Jede Partition kann anders formatiert sein oder auch ein anderes Betriebssystem enthalten. Jedes Medium kann bis zu 4 primäre Partitionen enthalten. Daneben kann eine erweiterte Partition angelegt werden, die dann wiederum 4 Partitionen enthalten kann. In der Regel kann man nur von primären Partitionen booten, erweiterte Partitionen dienen meistens der reinen Datenaufnahme.

Protected-Mode: Eine der Betriebsarten, in der ein x86 Prozessor Programme verabeiten kann. In dieser Betriebsart steht einem Programm das gesamte Leistungsspektrum des Prozessors und der gesamte Speicher zur Verfügung.

RAM-Disk: Enthält wie andere Medien auch ein Dateisystem und Daten, befindet sich jedoch vollständig im Arbeistspeicher. Dies ist für Rettungssysteme oder die erste Inbetriebnahme einer Karte sinnvoll, wenn auf den angeschlossenen Peripheriegeräten noch keine Daten vorhanden sind.

Real-Mode: Eine der Betriebsarten, in der ein x86 Prozessor Programme verabeiten kann. In dieser Betriebsart verhält sich der Prozessor wie eine 16 Bit CPU und kann auf die ersten 1 MB des Speichers zugreifen.

Sektor: Kleinste Verwaltungseinheit einer Diskette oder Festplatte, in der Daten abgelegt werden können. Meist 512 Byte Fassungsvermögen.

Standard-Linux-Binärabbild: Enthält einen Real-Mode Bootlader und das Betriebssystemabbild. Der Bootlader kann das Betriebssystem unter einem normalen PC-BIOS laden und starten.

x86: Bezeichnung für eine Prozessorfamilie unter der die Prozessoren 80386, 80486, Pentium, Pentium II...IV fallen. Ursprünglich von der Firma Intel entwickelt.