ARM&EVA / Tutorial

r1.3

Peer Georgi
Conitec Datensysteme GmbH

27. Februar 2006

Die Entwicklung von Software für embedded Linux (wie für die meisten anderen Betriebssysteme auch) findet auf verschiedenen Ebenen statt. Für eine gesicherte Entwicklung und nicht zuletzt für die Produktion müssen alle Schritte, welche letztendlich zur Endanwendung führen reproduzierbar und sicher sein.

Die mitgelieferte Software ist entwickelt worden, damit alle Prozesse, die im Zusammenhang mit Ihrer Softwareentwicklung stehen, sicher beherrscht werden.

Dieses Dokument soll Ihnen ein Leitfaden für den Umgang mit den Werkzeugen bieten.

Es wird vorausgesetzt, dass die mitgelieferte CD-ROM erfolgreich gestartet wurde oder die „carmeva-xxx" Pakete in eine bereits bestehende Linux-Distribution installiert wurden.

CONITEC DATENSYSTEME

Inhaltsverzeichnis

I Grundlagen **5**

1 Eigene Programme ausführen **5**
1.1 Eigene Linux-Programme 5
1.1.1 Remote Shell via Telnet 5
1.1.2 Verbindung per NFS einrichten 6
1.2 Eigene Betriebssystemunabhängige Programme übertragen 7

2 Programmieren des integrierten Flash-Speichers **8**
2.1 Programmierung des Flash direkt unter Linux 8
2.2 Verwendung der "*ebTools*" 9
2.2.1 Der Bootloader 11
2.2.2 Den Bootloader entfernen 11
2.2.3 Widerherstellen des Auslieferungszustandes 12

3 Anpassungen vornehmen **13**
3.1 Programme zum Start 14
3.2 Konfigurationsdaten zur Startzeit 14
3.3 Benutzerdaten 15
3.4 Linux anpassen 16
3.4.1 IP-Adresse und MAC-Adresse anpassen... 16
3.4.2 Wurzelverzeichnis anpassen... 16

II Beispiele **17**

4 Customized Linux „Schritt für Schritt” **17**
4.1 Bootloader entfernen . . . 17
4.1.1 Terminalprogramm starten... . . . 17
4.1.2 Bootloader interaktiv entfernen... . . . 18
4.2 Anpassungen vornehmen... . . . 18
4.2.1 Änderung der IP-Adresse . . . 19
4.2.2 Anpassung des Root-Dateisystems . . . 19
4.3 Veränderte Distribution testen... . . . 21
4.4 Veränderte Distribution in den Flash schreiben... . . . 22

5 Customized RootFs „Schritt für Schritt” **23**
5.1 Grundlagen . . . 23
5.2 Erstellung eines Rootdateisystems . . . 24
5.2.1 Vorbereitung . . . 24
5.2.2 Erstellen eines Dateisystems . . . 25
5.2.3 Einbinden des Dateisystems („Loopback-mount”) . . . 26
5.2.4 Erzeugen der Basisdistribution . . . 26
5.2.5 Profil Ihrer Distribution . . . 27
5.2.6 Aushängen und komprimieren . . . 27
5.2.7 Test des neuen Rootdateisystems . . . 28

6 Customized BusyBox „Schritt für Schritt” **29**

7 Customized Linux-Kernel „Schritt für Schritt” **30**
7.1 Konfiguration . . . 30
7.2 Test des neuen Kernels . . . 31

8 Betriebssystemunabhängige Programme **32**
8.1 Voraussetzungen . . . 32
8.2 Programme starten... . . . 32
8.2.1 Ein Beispiel . . . 32
8.2.2 Noch ein Beispiel . . . 32
8.3 Programme in den Flash programmieren... . . . 34
8.4 Programme entwickeln... . . . 34
8.4.1 Beispiele . . . 35
8.4.2 Kompfortabeles Entwickeln mit kdevelop3 . . . 35

Dieses Dokument

Im Folgenden soll kurz die Nomiklatur vorgestellt werden. Es wird Gebrauch von Bildschirmbildern (Screenshot's) gemacht, welche wiefolgt aussehen:

Hier werden **Befehle** hervorgehoben. Besondere *Parameter* sind ebenfalls gekennzeichnet. Das Zeichen ↰ kennzeichnet die Return Taste.

Hinweise, welche der Leser möglichst zur Kenntnis nehmen sollte, werden in folgender Form dargestellt:

Das sollten Sie lesen... **Hinweis**

Für den Leser sinnvolle Informationsquellen für das eine oder andere Thema werden mit **„Weiterführende Informationen zu diesem Thema"** angedeutet.

Dieses Tutorial ist in zwei Kapitel gegliedert. Der Grundlagenteil soll dem Leser einen groben Überblick über die Thematik vermitteln. Das zweite Kapitel bietet Beispiele, die unmittelbar mit der ausgelieferten Live-CDRom durchführbar sind. Sollten Sie nicht im Besitz der CD-Rom sein oder möchten Sie an ihrer Eigenen Linux-Distributionen arbeiten, so können Sie die notwendigen Pakete von unserer Download-Seite im Internet unter „***www.conitec.com***" beziehen.

Viel Erfolg und gutes gelingen!

Teil I
Grundlagen

1 Eigene Programme ausführen

Eigene Programme können sowohl Linux-Programme, als auch betriebssystem**unabhängige** Programme sein, die **anstelle** des Linuxkernels ausgeführt werden. Betriebssystemunabhängige Programme bieten die Möglichkeit die MCU ohne Betriebssystem zu betreiben. (Siehe später)
Für die unmittelbare Ausführung von Linux Programmen ist **keine** Neuprogrammierung des Flashs notwendig, da die vorinstallierte Distribution darauf vorbereitet ist, Anwendungen direkt über das lokale Netzwerk (Ethernet) auszuführen.

Soll jedoch der Inhalt des Flashs geändert- oder betriebssystemunabhängige Programme ausgeführt werden, muss zunächst der Bootloader entfernt werden. Dieser Vorgang ist in 2.2.2 ausführlich beschrieben.

1.1 Eigene Linux-Programme

Um eigene Linux-Programme zu starten, bietet sich ein NFS (**N**etwork **F**ile **S**ystem) - Share an. Hierbei wird ein beliebiges Verzeichnis des Entwicklungs-PCs (vorzugsweise auch ein Linux-System), in die Verzeichnisstruktur des Zielsystems eingeblendet (**gemounted**).
Diese Methode hat den Vorteil, dass schnelle Entwicklungszyklen erreicht werden können und die Übertragung des Programms vollständig transparent erfolgt.

Zur persistenten Speicherung im Flash ist dessen Neuprogrammierung notwendig, was Thema eines der folgenden Kapitel ist. (Siehe 3.4).

1.1.1 Remote Shell via Telnet

Die vorinstallierte Linux-Distribution enthält einen Telnetserver. Somit kann mit jedem beliebigen Telnet-Client auf das MCU-Modul zugegriffen werden. Für das Login als „root” wird **kein** Passwort benötigt. Unter einer beliebigen Shell führt folgende Eingabe zum Ziel:

```
telnet 192.168.1.12
Escape character is '^ ]'.

BusyBox v1.00-rc3 (2005.05.24-16:15+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.



/ #
```

Nach erfolgreich aufgebauter Telnet-Verbindung können alle installierten Kommandos verwendet werden.

1.1.2 Verbindung per NFS einrichten

Verbinden Sie sich, wie im vorherigen Abschnitt beschrieben, mit dem Zielsystem. Damit Sie eine NFS-Verbindung aufbauen können, benötigen Sie die Information, welche IP-Adresse Ihr Entwicklungs-PC verwendet. Sollte Ihr Entwicklungs-PC dynamisch (per DHCP) konfiguriert worden sein, können Sie das Program **ifconfig** in einr Root-Shell ausführen. Im Folgenden wird davon ausgegangen, dass Ihr PC die IP-Adresse **192.168.1.2** verwendet.
Die Eingabe von:

```
nfs-connect 192.168.1.2
/ #
```

Baut eine Verbindung per NFS auf. Alle Dateien, welche Sie in Ihr Freigabeverzeichnis (per Default „**/mnt/target-transfer**") kopieren, sind für das Zielsystem unter **/mnt/nfs** sichtbar.

1.2 Eigene Betriebssystemunabhängige Programme übertragen

Betriebssystemunabhängige Programme werden mit Hilfe der MCU-Firmware, unmittelbar nach dem MCU-Reset übertragen. Hierfür muss jedoch verhindert werden, dass die MCU aus dem Flash startet. Befindet sich das MCU-Modul in einem startfähigen Zustand, muss also der Bootloader zunächst entfernt werden (Siehe 2.2.2).
Für die Übertragung selbst bieten sich verschiedene Verfahren an.

- Übertragung per V24
  Hier kommt das XModem-Protokoll zum Einsatz
- Übertragung per USB
  Hier kommt zunächst das USB-DFU Protokoll, danach das LDBA-Protokoll zum Einsatz.

Die MCU unterstützt beide Verfahren, um Daten bzw. Programme zu empfangen. Auf dem Entwicklungs-PC kann das Programm "*ebStartUp"* verwendet werden, welches wiederum ein beliebiges Programm oder auch Daten in den Arbeitsspeicher des MCU-Moduls überträgt und zur Ausführung bringt. Es unterstützt die schnelle Datenübertragung mit Hilfe der USB-Schnittstelle und der Startvorgang kann vollständig vom Entwicklungs-PC gesteuert werden. Alternativ kann das XModem-Protokoll verwendet werden um bis zu 16kByte grosse Programme zu übertragen und auszuführen. Für genauere Informationen sei auf das Handbuch von "*ebStartUp"* und das Datenblatt der MCU verwiesen.

Weiterführende Informationen zu diesem Thema: Das „ebStartUp” - Handbuch.

2 Programmieren des integrierten Flash-Speichers

Auf dem MCU-Modul ist ein programmierbarer Flash-Baustein vorhanden. Dieser kann verwendet werden, um Programme automatisch zu starten. Die Einrichtung des Flash-Speichers kann wie folgt ablaufen:

1. Einrichtung des Flash-Speichers ohne Linux Unterstützung.
   (Verwendung der "*ebTools"*)
2. Einrichtung des Flash-Speichers unter Linux

2.1 Programmierung des Flash direkt unter Linux

Der Flash kann unter Linux programmiert werden, da der verwendete Flash bereits vom Linuxkern unterstützt wird. Das Programm "*ebFlashSetup "* erzeugt, neben den Daten für "*ebStartUp"* auch ein Abbild des Flashinhaltes als Datei, deren grösse der Flashgrösse entspricht. Diese Datei kann 1:1[1] in den Flash geschrieben werden. Somit steht ein Weg zur Verfügung den Flash auch unter Linux zu programmieren. In diesem Zusammenhang sei auf die „**/dev/mtdX**”-devices verwiesen, welche im Internet dokumentiert sind.

Weiterführende Informationen zu diesem Thema: Quellen des Linux-Kerneles; Datenblatt des AT91RM9200.

[1] Es muss die Kennung für startfähigen Inhalt berücksichtigt werden - Stichwort „Vector 6”. (Siehe Datenblatt der MCU Abschnitt „7.3 Bootloader”)

2.2 Verwendung der "*ebTools*"

Die Programmierung des Flash-Speichers erfolgt durch ein Programm, das auf der MCU ausgeführt wird. Zuvor werden alle Daten, welche in den Flash programmiert werden sollen, in den Arbeitsspeicher der MCU übertragen werden. Dieses Vorgehen ist **betriebssystemunabhängig** und erfolgt analog für Programme, die anstelle des LinuxKernels ausgeführt werden sollen.

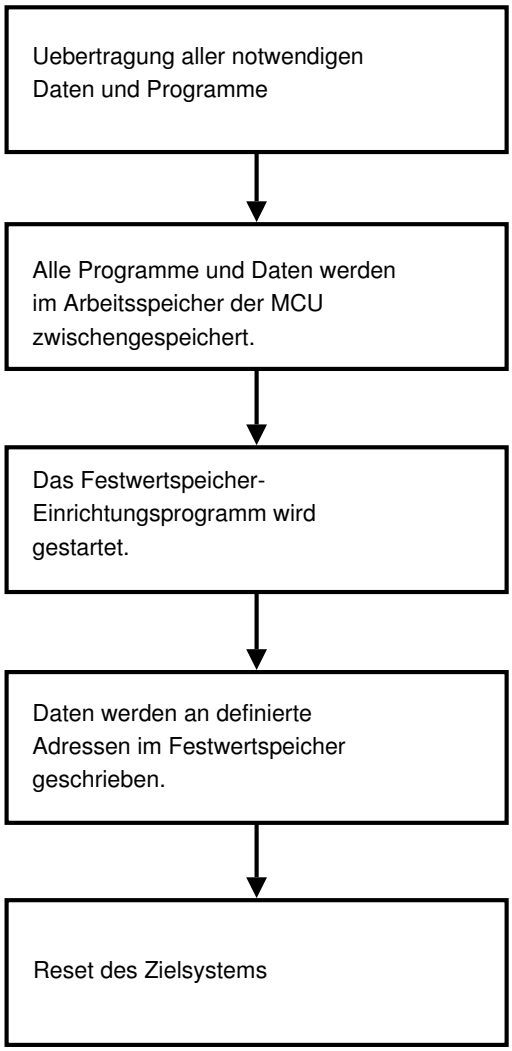

Abbildung 1: Ablauf der Flash-Programmierung (gesteuert durch "*ebStartUp*")

Damit der Programmiervorgang des Flash-Speichers für den Anwender so einfach, wie möglich ist, wurde das Programm "*ebFlashSetup* " entwickelt. Es berechnet die Zieladressen der Daten im Flash und arbeitet mit "*ebStartUp*" zusammen. Auf diese Weise ist ein automatisierbarer Flash-Einrichtungsprozess möglich, der vom Entwicklungs-PC gesteuert wird! Für nähere Informationen sei auf das Handbuch von "*ebFlashSetup* " verwiesen. Wei-

terhin liegen Beispiele, in Form von "*ebFlashSetup "* - Projektdateien bei. Nicht zuletzt die "*ebFlashSetup "* -Projektdatei aus dem „**recovery**"-Verzeichnis, kann als Beispiel dienen.

Weiterführende Informationen zu diesem Thema: Das „ebFlashSetup" - Handbuch.

2.2.1 Der Bootloader

Der Bootloader wird von der MCU-internen Firmware geladen und ausgeführt. Er hat die Aufgabe hardwarenahe Peripherie der MCU zu konfigurieren und hat die Aufgabe Datensätze und Programme aus dem Flash in den Arbeitsspeicher zu laden. Er gibt während dieses Vorgangs Meldungen über die V24 (RS232) aus.

Der mitgelieferte Bootloader wird im Quellkode mitgeliefert und kann der Endanwendung entsprechend angepasst werden. Er wurde in der Programmiersprache „C” verfasst.

2.2.2 Den Bootloader entfernen

Es gibt zwei mögliche Fälle, die das entfernen des Bootloaders erfordern:

1. Es sollen Programme ausgeführt werden, die mit Hilfe der USB-Schnittstelle übertragen werden.
2. Der Flash soll mit Hilfe von "*ebStartUp"* und "*ebFlashSetup "* neu programmiert werden.
   (Im wesentlichen die gleiche Situation, wie Punkt 1).

Hinweis

Haben Sie keine Angst vor dem entfernen des Bootloaders!

Die MCU enthält eine **nichtlöschbare** Firmware, welche in jedem Fall Kommunikationskanäle bereitstellt, die von "*ebStartUp"* unterstützt werden.

Der mitgelieferte Bootloader lässt sich mit Hilfe eines Terminalprogramms, dass die V24-Schnittstelle bedient, steuern. Er lässt sich entfernen, indem sofort nach dem Systemreset die Taste „**r**” im Terminalprogramm gedrückt wird. Nach Beendigung des Ladevorgangs, muss die Aktion durch drücken der Taste „**y**” bestätigt werden.
Nach diesem Vorgang gilt der Flash-Inhalt als **nicht startfähig** und die MCU versucht nach dem Neustart ein Programm zu empfangen. Soll die MCU ein Programm ausführen, muss es zunächst übertragen werden. (Siehe 1.2). Dieser Zustand ist für den Entwicklungsprozess sinnvoll, da sehr schnell neue Programme oder auch Distributionen ausprobiert werden können, **ohne das nach jeder Änderung der Flash erneut programmiert werden muss**.

Weiterführende Informationen zu diesem Thema: Lesen Sie Kapitel 4. Dort befindet sich ein Beispiel.

2.2.3 Widerherstellen des Auslieferungszustandes

Der Auslieferungszustand kann hergestellt werden, indem der Flash mit den „recovery-Daten" erneut programmiert wird. Die Voraussetzung hierfür ist ein nicht startfähiger Flash-Inhalt. Falls sich das MCU-Modul im startfähigen Zustand befindet, muss zunächst der Bootloader entfernt werden (Siehe 2.2.2).

Im Verzeichnis „***recovery***" befindet sich eine "*ebFlashSetup "* -Projektdatei, sowie die dazugehörigen Daten, um das MCU-Modul in den Auslieferungszustand zu bringen. Hierfür wird einfach das Skript „***recovery***" gestartet. Folgende Eingaben veranlassen dies:

```
cd /usr/local/carmeva/sw/recovery

./recovery
```

Beachten Sie, dass in diesem Fall eine Default- MAC-Adresse verwendet wird. Sie lässt sich jedoch anpassen, indem die Datei „*mac-address*" im Unterverzeichnis „ressources" angepasst wird.

Hinweis

Die MCU-Module werden mit eindeutigen MAC-Adressen ausgeliefert. Die auf der CD vorhandene MAC-Adresse ist eine **Default-Adresse**. Aus diesem Grund wird empfohlen die gelieferte MAC-Adresse gut aufzubewahren und zu verwenden.

3 Anpassungen vornehmen

Alle in diesem Kapitel besprochenen Änderungen setzen einen nicht startfähigen Flashinhalt und ggf. die Neuprogrammierung des Flashs voraus. Aus diesem Grund müssen die vorherigen Abhandlungen verstanden worden sein und der Bootloader entsprechend Kapitel 2.2.2 entfernt werden.
Die auf dem Zielsystem vorinstallierte Linux-Distribution besteht im wesentlichen aus dem **Linux-Kernel**, dem „**busybox**"-Projekt, und einigen **Shell-Skripten**. Zusätzlich lassen sich Merkmale konfigurieren, welche vor dem Linux-Start relevant sind. Dazu gehören:

- Bootloader
- MAC-Adresse
- IP-Adresse / Linux Kommandozeile

Alle Komponenten lassen sich auf Ihre Anwendung anpassen, da alle Quellkodes verfügbar sind. Die folgende Abbildung zeigt die verschiedenen Ebenen.
Der Inhalt des Flash-Speichers (Flash-Image) besteht aus einem Paket der genannten Elemente, welches durch "*ebFlashSetup "* erstellt wird. Folgende Abbildung gibt einen Überblick über die verschiedenen Ebenen.

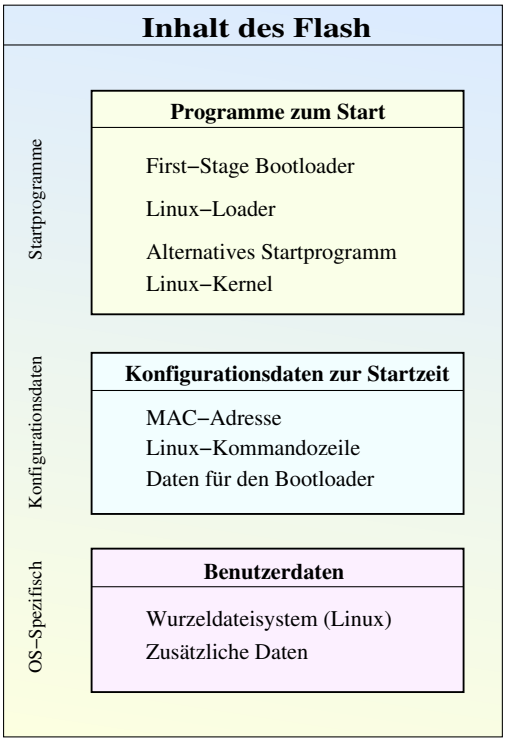

Abbildung 2: Flash-Inhalt

Die verschiedenen Konfigurationsebenen bilden sich auf verschiedene Teilprojekte ab, die auf der Entwicklungs-CD in verschiedenen Unterverzeichnissen abgelegt wurden.

3.1 Programme zum Start

Die Systemprogramme zum Start des MCU-Moduls werden im Quellkode mitgeliefert. Dieser befindet sich
unter „**/usr/local/carmeva/sw/boot/sw.arm9**".
Aus diesem Grund können Anpassungen vorgenommen werden. Zur Konfiguration des vorinstallierten Linux-Systems sind hier jedoch **keine Änderungen nötig**.

3.2 Konfigurationsdaten zur Startzeit

Die Konfigurationsdaten liegen in Form von einfachen Dateien (meist Textdateien) vor, deren Inhalte vom Bootloader oder dem Linux-Kern ausgewertet werden.

3.3 Benutzerdaten

Zu den Benutzerdaten gehört das Wurzelverzeichnis der Linux-Distribution. Es spricht jedoch nichts gegen weitere Datenblöcke, welche unabhängig vom Betriebssystem Informationen enthalten können.

3.4 Linux anpassen

Auf der beigelegten CD-ROM befindet sich
unter „**/usr/local/carmeva/sw/linux/examples/custom-linux**" eine Linuxdistribution, welche als Startpunkt für eigene Änderungen gelten kann. Sie entspricht in ihrer Vorkonfiguration den Werkseinstellungen, kann aber sehr leicht angepasst werden.
Das Verzeichnis enthält folgendes:

- **ReadMe** - zum durchlesen
- **start/** - ein **Makefile** mit der die Distribution getestet werden kann (ohne den Flash zu beschreiben)
- **setup/** - ein **Makefile** mit der die Distribution in den Flash geschrieben werden kann
- **ressources/** - enthält alles, was die Linux-Distribution benötigt

Die genaue Verwendung der Dateien sollte im nächsten Kapitel deutlich werden.

3.4.1 IP-Adresse und MAC-Adresse anpassen...

Bitte lesen Sie hierzu das Kapitel 4.

3.4.2 Wurzelverzeichnis anpassen...

Bitte lesen Sie hierzu das Kapitel 5.

Teil II
Beispiele

4 Customized Linux „Schritt für Schritt”

Dieses Kapitel führt Sie Schritt für Schritt durch die Anpassung der Linuxdistribution und deren Test. Dabei sollte die Verzeichnisstruktur klar werden. Es wird empfohlen, die einzelnen Teilschritte (auch die Quellen der verwendeten Skripte und Makefiles) genauer zu untersuchen, damit das Gesamtsystem verstanden wird.

Dabei stellt sich das Kapitel das hypothetische Ziel, dass sie nach dem Systemstart ein
„**...Diese Zeile habe ich selbst geschrieben...**”
auf dem Display, sowie der seriellen Konsole ausgeben wollen. Weiterhin soll die IP-Adresse auf „**192.168.0.25**” geändert werden. Das Kapitel orientiert sich an der Installation der mitgelieferten Live-CD und geht davon aus, dass sowohl das V24-Kabel, als auch das USB-Client Kabel mit dem Entwicklungs-PC verbunden ist.
Zunächst folgen die groben Schritte:

1. Bootloader entfernen, wenn er noch existiert
2. Anpassungen vornehmen
   (a) Linux-Kommandozeile anpassen
   (b) Wurzeldateisystem („**rootfs**”) anpassen
3. Linux-Distribution testen
4. Linux-Distribution in den Flash speichern

4.1 Bootloader entfernen

Wenn das ARM-Modul noch im ausgelieferten Zustand ist (der Bootloader ist aktiv), so muss zunächst der Bootloader entfernt werden. Sollte dies schon passiert sein, kann dieser Abschnitt übersprungen werden.
Zum entfernen des Bootloaders wird das Terminalprogramm benötigt.

4.1.1 Terminalprogramm starten...

Der Name des Terminalprograms („gtkterm”)[2] wird als Kommando in das Kommandofenster eingegeben, welches bei der Tastenkombination „Alt+F2” erscheint. Nach einem Reset des Evaluationboards sollten nun Meldungen im Terminal erscheinen... .

[2] Alternativ kann „**minicom**” in der Konsole verwendet werden. Auf WinXX Platformen steht das Programm „Hyperterm” zur Verfügung.

4.1.2 Bootloader interaktiv entfernen...

Wie bereits in 2.2.2 beschrieben, lässt sich der Bootloader entfernen, indem sofort nach dem Reset die Taste „**r**” gedückt[3] wird. Folgende Bildschirmausgabe sollte die Folge sein:

```
Press "y" to remove the boot sector...


***! boot sector removed !***
Please restart target...
```

Nachdem die Frage mit „y” beantwortet wurde, ist der Bootloader inaktiv. Nach einem weiteren Reset des Zielsystems meldet sich die Firmware der MCU mit „C”. Dies ist das Startzeichen für das XModem-Protokoll und zeigt, dass die MCU auf ein Programm wartet, welches über die serielle Schnittstelle übertragen werden kann. Parallel dazu ist der USB-Client aktiv und kann als schneller Übertragungsweg verwendet werden. (Siehe später).

4.2 Anpassungen vornehmen...

In diesem Beispiel handelt es sich um zwei Anpassungen, welche auf zwei verschiedenen Ebenen konfiguriert werden:

- ➤ IP-Adresse: Wird mit Hilfe der Kommandozeile konfiguriert.
- ➤ Eine Ausgabe zum Systemstart: Wird im „rootfs” konfiguriert.

Alle Änderungen werden in einer Linux-Shell durchgeführt. Shell-Programme sind beispielsweise „konsole” oder auch „xterm”.
Zunächst wechsele man in das Verzeichnis
„**/usr/local/carmeva/sw/linux/examples/custom-linux**”. Im Unterverzeichnis „**ressources**” stehen alle Datensätze der Distribution.

```
cd /usr/local/carmeva/sw/linux/examples/custom-linux/ressources
ls
LdBigAppUSB LdDataFlash LinuxKernel linux.cmdline rootfs.image.gz use-prebuild-kernel
LdBigAppUSB_Alt LdLinux++ StDataFlash mac-address use-custom-kernel
```

[3] Gross- und Kleinschreibung beachten!

- „linux.cmdline” - Die Kommandozeile des Linux-Kernels
- „mac-address” - Die eingestellte MAC-Adresse der Ethernet-PHY
- „rootfs.image.gz” - Das komprimierte Rootdateisystem
- „LinuxKernel” - Der Linuxkernel selbst, in unveränderter Form, wie er unter „**linux/arch/arm/boot/Image**” nach dem Kompilieren des Kernels zur Verfügung steht
- Der Rest sind Bootloader und Hilfsprogramme, welche vor dem Linux-Kernel ausgeführt werden.

Die beiden Shell-Skripte „use-prebuild-kernel” und „use-custom-kernel” können für spätere Entwicklungen hilfreich sein und werden im Moment nicht benötigt. Wer genau hinschaut erkennt, das „LinuxKernel” ein symbolischer Link ist, welcher per Default auf einen vorkonfigurierten Linuxkernel zeigt. Dieser Link wird mit „**use-prebuild-kernel**” erzeugt. Sollten später die Änderung des Linux-Kernels notwendig sein, kann „**use-custom-kernel**” dazu verwendet werden den Link auf das Kernel-Verzeichnis zeigen zu lassen, wo der selbst kompilierte Linux-Kernel liegt.

Für das aktuelle Beispiel sind folgende Dateien interessant:

1. „linux.cmdline”
2. „rootfs.image.gz”

4.2.1 Änderung der IP-Adresse

Hierfür muss die Datei „linux.cmdline” bearbeitet werden. Es handelt sich um eine gewöhnliche Textdatei, welche mit jedem ASCII-Texteditor bearbeitet werden kann. Der Aufbau der Linux-Kommandozeile ist in der Dokumentation des Linux-Kernels beschrieben und selbsterklärend. Die IP-Adresse, sowie die weiteren Netzwerkeinstellungen können dort editiert werden.

Das Programm "*ebLinuxCmdLine*" wertet diese Textdatei aus und erzeugt daraus die binäre Darstellung, welche der Linux-Kernel erwartet.

4.2.2 Anpassung des Root-Dateisystems

Etwas aufwändiger ist die Änderung des „rootfs”. Hier gibt es zwei prinzipielle Vorgehensweisen:

1. Ändern eines bereits bestehenden Root-Dateisystems
2. Erzeugen eines völlig neuen Root-Dateisystems

Zur Änderung / Erstellung des Root-Dateisystems existiert ein eigenes Projekt unter „**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs**”. Hier befindet sich das vorkonfigurierte Root-Dateisystem, entsprechend des Auslieferungszustandes.

Änderung des bestehenden Root-Dateisystems

Im Folgenden wird von der Existenz der Datei „**rootfs.image.gz**” ausgegangen. Folgende Schritte sind durchzuführen:

- ➤ Rückgängigmachen der Kompression („Auspacken”) der Datei
- ➤ „Loop-Back Mount” der Datei „rootfs.image”
- ➤ Änderungen durchführen
- ➤ „Unmount” des „Loop-Back mounts”
- ➤ Komprimieren der Datei

Zunächst wird die Komprimierte Datei in dem Verzeichnis ausgepakt, wo sie sich befindet. Folgende Eingabe führt zum Ziel:

```
cd /usr/local/carmeva/sw/linux/prepare.rootfs/rootfs
gunzip rootfs.image.gz
ls
Makefile ReadMe Target abc distribution.basic ressources rootfs.image
```

Es befindet sich nun eine Datei mit dem Namen „rootfs.image” im Verzeichnis. Diese Datei enthält das Root-Dateisystem in Form einer „ext2” Partition. Diese Partition kann jetzt gemounted werden.

```
sudo mount -o loop rootfs.image Target
```

Nach diesem Schritt ist das Root-Dateisystem des Zielsystems unter „Target” eingebunden und zeigt folgenden oder ähnlichen Inhalt:

```
ls
bin boot dev etc host lib linuxrc lost+found mnt proc sbin sys tmp usr var
```

Ausführen eines Befehls zum Systemstart

Um unmittelbar nach dem Systemstart eine Meldung auf das Display zu schreiben, bedarf es der Änderung des „Init-Scripts”. Dieses befinded sich unter „**Target/etc/init.d/rcS**”. Es kann

mit einem beliebigen ASCII-Editor editiert werden. Um das beschriebene Ziel zu erreichen, sind jedoch folgende Eingaben ausreichend:

```
cd Target/etc/init.d
echo  "echo ...Diese Zeile habe ich selbst geschrieben..." » rcS
echo  "echo ...Diese Zeile habe ich selbst geschrieben... » /dev/tty1" » rcS
```

Hiermit wird die Datei „rcS” um zwei Zeilen ergänzt. Der erste Zeile gibt die Meldung auf der Boot-Konsole aus (Serielles Terminal). Die zweite Zeile leitetet die Ausgang auf „**/dev/tty1**” um und sorgt so für die Darstellung auf dem Display.

Die Änderungen des Root-Dateisystems sind für dieses Beispiel abgeschlossen. Es kann also aus dem Verzeichnisbaum „ausgehangen” / „unmounted” werden.

```
cd ..
sudo umount Target
# Komprimieren...
gzip -9
ls
Makefile ReadMe Target distribution.basic ressources rootfs.image.gz
```

Die Datei „rootfs.image.gz” hat jetzt den veränderten Inhalt.

Weiterführende Informationen zu diesem Thema: Dokumentation des busybox Projektes. (**http://www.busybox.net**); Shell Programmierung.

Erzeugen eines völlig neuen Root-Dateisystems

Dieses Thema wird im Abschnitt 5 ausführlich beschrieben.

4.3 Veränderte Distribution testen...

Um die Distribution zu testen, muss sie zum Zielsystem übertragen werden. Diese Aufgabe übernimmt das Program "*ebStartUp"* . Es wird, wie alle "*ebTools"* über eine Projektdatei konfiguriert, welche als ASCII-Textdatei vorliegt. Das Programm "*ebStartUp"* ist sehr mächtig und kann nicht nur das Linux-Betriebssystem starten, sondern auch alle eigene betriebssystem unabhängige Programme.

Im Verzeichnis „**/usr/local/carmeva/sw/linux/examples/custom-linux/**” befindet sich eine Projektdatei für "*ebStartUp"* und ein Makefile. Folgende Eingabe startet das Betriebssystem:

```
cd /usr/local/carmeva/sw/linux/examples/custom-linux/start
make start
```

Nach diesen Eingaben kann im Terminalfenster der Startvorgang verfolgt werden.

Weiterführende Informationen zu diesem Thema: Ein Blick in das Makefile; Dokumentation von "*ebStartUp"* . (**/usr/local/carmeva/doc/de/manual/ebSuite**)

4.4 Veränderte Distribution in den Flash schreiben...

Um das System selbst startfähig zu machen kann der Flash programmiert werden. Diese Aufgabe übernimmt "*ebFlashSetup "* im Zusammenhang mit "*ebStartUp"* . Auch hier existiert eine vorkonfigurierte "*ebFlashSetup "* Konfigurationsdatei.

```
cd /usr/local/carmeva/sw/linux/examples/custom-linux/setup
make setup
```

Auch hier können und sollten die Ausgaben auf dem seriellen Terminal verfolgt werden, da das Ende des Programmiervorgangs derzeit nur auf dem seriellen Terminal angezeigt wird.

Hinweis

Das Programm StDataFlash verwendet die LEDs von EVA, um seine Aktivität anzuzeigen.

Nachdem die Übertragung via USB abgeschlossen ist, blinkt eine LED mit langsamer Periode (ca. 1/2 Sekundentakt). Dies zeigt an, dass die Flashprogrammierung gerade vorgenommen wird. Am Ende des Einrichtungsvorgangs werden alle vier LEDs als „Lauflicht” betrieben. In diesem Zustand kann ein Reset durchgeführt werden.

Weiterführende Informationen zu diesem Thema: Ein Blick in das Makefile; Dokumentation von "*ebStartUp"* . (**/usr/local/carmeva/doc/de/manual/ebSuite**)

5 Customized RootFs „Schritt für Schritt”

Das Wurzeldateisystem („Rootdateisystem”, „Wurzeldateisystem”, „rootfs”) enthält alle Programme und Daten, welche das Betriebssystem benötigt. Zusätzlich finden eigene Programme (z.B. Ihre Endanwendung), dort ihren Platz. Das Wurzeldateisystem liegt in Form eines Verzeichnisbaums vor dessen Struktur nicht bzw. nur unwesentlich von der, einer gewöhnlichen Linux-Distribution abweicht. Der Verzeichnisbaum wird wiederum in einem von Linux unterstützten Dateisystem gespeichert. Die folgenden Beispiele beziehen sich auf die Verwendung des Ext2-Dateisystems.

Beim Systemstart wird das Rootdateisystem in Form einer RamDisk verwendet. Programme des Zielsystems können alle Dateien ändern und neue hinzufügen, bis der Speicherplatz der RamDisk erschöpft ist.[4]

Hinweis

Lassen Sie sich durch die Komplexität dieses Abschnitts nicht abschrecken. Ihre CArmEva-Distribution macht es Ihnen so einfach wie möglich!

5.1 Grundlagen

Bei der Erstellung eines Rootdateisystems müssen folgende Parameter berücksichtigt werden:

- Grösse der Ramdisk = grösse des unkomprimierten Dateisystems
- Summe des durch den Inhalt des Rootdateisystems belegten Speichers

Durch die Tatsache, dass das Dateisystem nach Abschluss aller Änderungen komprimiert wird, resultieren vier verschiedene Grössen, die der Entwickler kennen sollte:

1. Grösse der Ramdisk in bytes
2. Grösse des unkomprimierten Rootdateisystems in bytes
3. Freier Platz im unkomprimierten Rootdateisystem in bytes
4. Grösse des komprimierten Rootdateisystems in bytes

[4] Änderungen von Dateien betreffen nur die RamDisk (niemals den Flash).

5.2 Erstellung eines Rootdateisystems

Zunächst der prinzipielle Ablauf:

- ➤ Erstellung eines Dateisystems in eine Datei
  - Erstellung einer leeren Datei, mit der Grösse des unkomprimierten Rootdateisystems
    (z.B. 16MByte = **16384**kByte)
  - Erstellung eines Dateisystems in diese Datei
- ➤ „LoopBack-Mount” der Datei in ein beliebiges Verzeichnis
- ➤ Erzeugen der wichtigsten Daten und Einstellungen
  - Erstellung des leeren Dateibaums
  - Hinzufügen der wichtigsten Komponenten
    (Device-nodes, notwendige Shellbefehle, grundlegende Konfigurationsdateien)
- ➤ Anpassungen
  - Anpassen der Default-Konfiguration und hinzufügen eigener Konfigurationsdateien
    (z.B. Autostart von Prozessen oder Anwendungsprogrammen)
  - Hinzufügen eigener Komponenten
    (Anwendungsprogramme, Daten, Kernel-Treiber, ...)
- ➤ „Unmount” des Dateisystems
- ➤ Komprimieren des Dateisystems

Für die Erstellung eines Rootdateisystems „from scratch” existiert ein Projekt im Verzeichnis „**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs**”. Alle Vorgänge sind mit Hilfe eines **Makefiles** durchführbar und dennoch nicht bodenlos automatisiert. Somit bleibt dem Entwickler die nötige Transparenz, alle Teilschritte zu verstehen und nachvollziehen zu können. Jeder Teilschritt wird durch ein „Target” des Makefiles abgebildet, wie gleich deutlich wird...

5.2.1 Vorbereitung

Zunächst sollten Sie die Datei: „**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/rootfs.image.gz**” unter einem anderen Namen sichern.

```
cd /usr/local/carmeva/sw/linux/prepare.rootfs/rootfs
mv rootfs.image.gz rootfs.image.gz.backup
```

Alle weiteren Abläufe spielen sich in diesem Verzeichnis ab.

5.2.2 Erstellen eines Dateisystems

Das Erstellen einer leeren Datei, in welcher anschliessend ein Dateisystem erstellt wird, ist mit dem make-Ziel „**image-file**” definiert.

```
make image-file
```

Sie werden nun nach einer grösse gefragt, welche Sie in kilo bytes angeben müssen. Für eine 16MByte grosse Datei lautet die Antwort also: 16384. Daraufhin fragt das Programm „mkfs.ext2”, ob es ein Dateisystem erstellen soll und ob es das wirklich soll. Beide Fragen sind mit „y” zu beantworten.

```
Size of Image (kB): 16384
Creating image of size 16384kb - success
Do you wan't to create a file system (ext2) ? (Y/n): Y
Creating file system... rootfs.imagemke2fs 1.38 (30-Jun-2005)
rootfs.image is not a block special device. Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
4096 inodes, 16384 blocks
819 blocks (5.00
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 32 mounts or 180 days, whichever comes first.
Use tune2fs -c or -i to override.
```

Danach befindet sich eine 16MByte grosse Datei mit dem Namen „**rootfs.image**” im aktuellen Verzeichnis, die auf ihren „LoopBack”-Mount wartet...

5.2.3 Einbinden des Dateisystems („Loopback-mount”)

Dieser Schritt erfolgt durch die Eingabe von:

```
make mount
```

Der Mountpoint ist das zuvor leere Verzeichnis „Target”. Nach diesem Schritt darf „Target” nicht mehr leer sein, sondern genau einen Eintrag aufweisen:

```
cd Target
ls
lost+found
cd ..
```

5.2.4 Erzeugen der Basisdistribution

In diesem Schritt werden:

- Der Verzeichnisbaum und Devicenodes erzeugt
- Die wichtigsten Shellkommandos installiert

```
make create
make basic_dist
```

Die Schellkommandos entstammen aus dem freien BusyBox-Projekt, dessen Quellen unter „/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/**ressources/busybox**” zu finden sind. Die Konfiguration der BusyBox ist im Kapitel 6 beschrieben. Das Target „*basic_dist*” installiert grundlegende Konfigurationsdateien.

Nach diesen Schritten ist das Rootdateisystem prinzipiell startfähig. Was fehlt sind einige anwendungsspezifische Anpassungen der Konfigurationsdateien und nicht zuletzt Ihr Anwendungsprogramm.

5.2.5 Profil Ihrer Distribution

Verschiedene Anwendungen des MCU-Moduls wirken sich, neben evtl. zusäzlicher Hardware, vor allem im Rootdateisystem aus. So werden verschiedene Konfigurationsdateien und Programme für verschiedene Projekte benötigt.
Damit Sie den Überblick bewahren können bietet das Makefile verschiedene Targets an, die mit Shell-Skripten zusammenarbeiten. Es handelt sich insbesondere um das Unterverzeichnis „**ressources/carmeva/**”. Hier befindet sich derzeit nur das Unterverzeichnis „factory”„ wo die Änderungen abgelegt wurden, welche Rootdateisystem führen, wie es dem Auslieferungszustand entspricht.

Das Shell-Skript, welches hier abgelegt wurde verändert das Konfigurationsverzeichnis „**Target/etc**”. Für nähere Informationen diesbezüglich, untersuchen Sie bitte den Inhalt des Verzeichnisses. Es wird im Makefile aufgerufen, wenn:

```
maske factory_dist
```

aufgerufen wird.
Nach dem gleichen Stil können Sie eigene Änderungen am default-Rootdateisystem vornehmen und eigene Targets im Makefile hinzufügen.

Weiterführende Informationen zu diesem Thema: Ein Blick in das Makefile („/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/**Makefile**”); Erkundung der Verzeichnisstruktur.

5.2.6 Aushängen und komprimieren

Nach all diesen Schritten ist das Rootdateisystem fertig und kann „eingepackt” werden.

```
cd /usr/local/carmeva/sw/linux/prepare.rootfs/rootfs    # Wenn sie dort nicht schon sind

make umount
make compress
```

Nach diesen Schritten befindet sich eine fertige Datei mit dem Namen „**rootfs.image.gz**” im aktuellen Verzeichnis[5], die auf ihren Test wartet.

[5] Die Dateiendung „.gz” wird durch den Komprimiervorgang automatisch hinzugefügt.

5.2.7 Test des neuen Rootdateisystems

Auch hier bietet das Projekt im
Verzeichnis „**/usr/local/carmeva/sw/linux/examples/custom-linux**” alles was Sie benötigen. Das Rootdateisystem, im Unterverzeichnis „ressources”, ist hier bereits als symbolischer Link auf die Datei ausgelegt, die Sie eben erzeugt haben. Somit kann der Test sofort mit folgender Eingabe durchgeführt werden:

```
cd /usr/local/carmeva/sw/linux/examples/custom-linux/start
make start
```

Nach diesen Eingaben werden alle Übertragungen ausgeführt und Ihr neues Rootdateisystem wird dabei verwendet.

5.2.7.1 In einigen Fällen sind Anpassungen in der Linux-Kommandozeile notwendig. Hier ist die grösse des Rootdateisystems und die maximale grösse der komprimierten Ram-Disk hinterlegt. Diese Parameter müssen entsprechend der aktuellen Situation angepasst werden.

5.2.7.2 Beispiel Haben Sie zu Beginn z.B. ein leeres Dateisystem der grösse **12288**kBytes = 12MByte erzeugt, muss dies in der Linux-Kommandozeile hinterlegt werden. Diese Grösse entspricht der, des unkomprimierten Dateisystems und ist als *„ramdisk_size*” bezeichnet.

Weiterhin mögen Sie feststellen, dass die komprimierte Datei („**rootfs.image.gz**”) grösser ist als **4MByte** (hypotetische Annahme z.B. 5000000 Bytes), somit müssen Sie weiterhin den „*initrd*”-Eintrag in der Linux-Kommandozeile wiefolgt anpassen:

„**initrd=**<PositionDerRamDiskImSpeicher**,**GrösseDesKomprimiertenImages>”

Da sich die Position im Speicher **nicht ändern** muss, muss nur der zweite Parameter angepasst werden. In diesem Beispiel also:

„initrd=0x20400000,**5000000** ramdisk_size=**12288**”

6 Customized BusyBox „Schritt für Schritt”

Das BusyBox ist ein freies Projekt, welches Grundlegende Schellkommandos in einer einzigen Applikation abbildet. („Multicallbinary”). Dies mindert den Ressourcesbedarf, da der Programmoverhead nur einmal anfällt und nicht für jedes Schellkommando.

Die Quellen des Projekts sind
unter „/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/ressources/busybox” zu finden. Die Konfiguration verläuft ähnlich wie die Kernelkonfiguration (nächster Abschnitt). Dabei kann konfiguriert werden, welche Komponenten bzw. welche Schellkommandos erwünscht sind und welche nicht verwendet werden. Diese Auswahl hat unmittelbaren Einfluss auf die grösse der ausführbaren Datei „*busybox*” und somit auf den Speicherbedarf des Rootdateisystems. Folgende Eingabe startet den Konfigurationsdialog.

```
cd /usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/ressources/busybox
make menuconfig
```

Daraufhin erscheint ein Menu, wo sie Änderungen durchführen können. Am Ende müssen die Einstellungen gespeichert werden und der Buildprozess schliesst sich an.

```
make
```

Am Ende des Kompiliervorgangs steht eine „*BusyBox*” mit neuen bzw. geänderter Funktionalität zur Verfügung. Diese Änderungen wirken sich aus, wenn das Rootdateisystem neu erstellt wird (Siehe 5.2).

Weiterführende Informationen zu diesem Thema: Dokumentation des BusyBox-Projekts („**http://www.busybox.net**”); Kapitel 5 dieses Dokuments.

7 Customized Linux-Kernel „Schritt für Schritt”

Der mitgelieferte Linuxkernel liegt als Quellkode im Verzeichnis „**/usr/local/carmeva/sw/linux/prepare.kernel/kernel**”. Es handelt sich um einen Kernel, wie er bei `http://maxim.org.za/AT91RM9200/2.6/` zu beziehen ist, hat aber einige Änderungen bezüglich der ARM&EVA Platform erhalten. Er ist weiterhin so konfiguriert, dass er bereits den notwendigen Cross-Compiler (gcc-3.4.1) verwendet.

7.1 Konfiguration

Der Ablauf der Konfiguration ist identisch mit der Konfiguration eines Linux-Kernels für einen PC und finedet vorzugsweise in einer Shell statt.

```
cd /usr/local/carmeva/sw/linux/prepare.kernel/kernel
make menuconfig
```

Das Menu, welches im daraufhin erscheint, bietet die Möglichkeit Anpassungen vorzunehmen. Die vielen Parameter und Einstellungen sind nicht Thema dieses Dokuments, da sie im Internet recherchiert werden können.

Nachdem notwendige Anpassungen vorgenommen und die Einstellungen gespeichert wurden, kann der Buildprozess beginnen. Auch hier verläuft der Vorgang, wie man es bei der „normalen” heimischen Platform kennt.

```
make
```

Dieser Befehl löst den Buildprozess aus und nachdem alles erfolgreich kompiliert und gelinkt wurde befindet sich unter „**arch/arm/boot**” die Datei „**Image**”. Diese Datei wird **direkt** für den Linuxloader verwendet und bedarf keiner weiteren Modifikation, wie es bei anderen Bootloaderprojekten (wie u-boot), notwendig ist.

7.2 Test des neuen Kernels

Der Test des neuen Linux-Kernels erfolgt beispielhaft im Verzeichnis „**/usr/local/carmeva/sw/linux/examples/custom-linux/”**, was der Leser bereits aus den vorherigen Kapiteln kennt. Zunächst muss dafür gesorgt werden, dass in Zukunft der „Entwicklungskernel” für den Start von Linux verwendet werden soll. Hierfür befindet sich im Unterverzeichnis „**ressources**” ein Skript, welches den symbolischen Link „LinuxKernel” auf den Entwicklungskernel im
Verzeichnis „**/usr/local/carmeva/sw/linux/prepare.kernel/kernel/arch/arm/boot/Image**” verweisen lässt.

```
cd /usr/local/carmeva/sw/linux/examples/custom-linux/ressources
./use-custom-kernel.sh
cd ../start
```

Damit ist bereits alles für einen Neustart des Zielsystems vorbereitet und der Start des neuen

Betriebssystems kann wie gewohnt durch die Eingabe von:

```
make start
```

erfolgen.

8 Betriebssystemunabhängige Programme

Wie zu anfangs bereits angedeutet, kann die MCU dafür verwendet werden Programme auszuführen, welche ihrerseits kein Betriebssystem benötigen (oder selbst eines darstellen). Der Bootloader ist ein gutes Beispiel für ein Betriebssystemunabhängiges Programm.

Im Verzeichnis „**/usr/local/carmeva/sw/boot/sw.arm9**" finden Sie einige Programme, welche direkt nach dem Reset der MCU ausgeführt werden können. Diese Programme können mit dem mitgelieferten GNU-make basierenden, innovativen Build-System (auch direkt von der Live-CDRom) verändert und kompiliert werden.

8.1 Voraussetzungen

Einem betriebssystemunabhängigen Programm unterliegen, neben seiner eigentlichen Aufgabe, zusätzliche Verantwortungen. Die wichtigsten davon ist die Initialisierung der hardwarenahen Peripherie.
So müssen **vor** dem Start der Anwendung der Arbeitsspeicher, der Systemtakt und andere Komponenten initialisiert werden. Auf der beigelegten CDRom sind Beispielprogramme abgelegt, welche einen Einblick in die grundlegende Struktur solcher Programme geben sollen.

8.2 Programme starten...

8.2.1 Ein Beispiel

Das Programm „cArmEvaLEDs" ist ein Programm, welches die LEDs des Evalboards ansteuert. Dabei kann es über das serielle Terminal mit den Tasten „1"-"4" gesteuert werden.

Start des Programms

```
cd /usr/local/carmeva/sw/user-applications/cArmEvaLEDs
./start
```

Nach dem Reset des Zielsystems wird das Programm per USB übertragen und sofort ausgeführt. Auf dem seriellen Terminal können die Ausgaben verfolgt werden.

8.2.2 Noch ein Beispiel

Das Programm „cArmEvaLCD" steuert das LC-Display direkt an.

```
cd /usr/local/carmeva/sw/user-applications/cArmEvaLCD
./start
```

Auch hier muss das Zielsystem mit einem druck auf den Reset-Taster neu gestartet werden, damit die Übertragung und die Ausführung beginnt.

Weiterführende Informationen zu diesem Thema: Ein Blick in das Shell-Skript („start”); Ein Blick in die "*ebStartUp"* -Projektdateien (*.ebs).

8.3 Programme in den Flash programmieren...

Das Programm "*ebFlashSetup "* kann, wie eingangs erwähnt alle Arten von Daten und Programme so aufbereiten, dass ein startfähiger Flash-Inhalt entsteht. Das gilt natürlich auch für Betriebssystem**unabhängige** Programme aller Art.

Für das cArmEva-LCD Projekt ist die Beispielkonfiguration für "*ebFlashSetup "* im Verzeichnis „/usr/local/carmeva/sw/user-applications/cArmEvaLCD_Flash" zu finden. Folgende Zeilen lösen den Programmiervorgang aus:

```
cd /usr/local/carmeva/sw/user-applications/cArmEvaLCD_Flash
./setup
```

Auch hier können und sollten die Ausgaben auf dem seriellen Terminal verfolgt werden, da das Ende des Programmiervorgangs derzeit nur auf dem seriellen Terminal angezeigt wird.

Weiterführende Informationen zu diesem Thema: Ein Blick in das Shell-Skript („setup"); Ein Blick in die "*ebFlashSetup "* -Projektdatei („**presentation.prj**").
Natürlich lohnt sich auch ein Blick in das ebStartUp-Handbuch, sowie das ebFlashSetup-Handbuch.

8.4 Programme entwickeln...

Programme, welche ohne Betriebssystem funktionieren sollen, können ähnlich wie „normale" Programme auch in C bzw. C++ entwickelt werden. Der entsprechende Kompiler liegt Ihrer ARM&EVA Distribution bei. Dennoch gibt es Unterschiede, die bei der Entwicklung beachtet werden müssen. Einige davon sollen hier erwähnt werden:

- kein Betriebssystem –> keine Betriebssystemaufrufe möglich –> Speicherverwaltung, Dateiverwaltung, ... fehlen
- Standartbibliotheken sind meist nicht ohne Anpassung verwendetbar
- Programme werden für eine feste Adresse des Arbeitsspeicher reloziert (Der Linkprozess unterscheidet sich wesentlich)

  ▷ *Sie als Entwickler sind für alle Funktionen selbst verantwortlich.*

Neben den Nachteilen, die offensichtlich scheinen, folgen aus diesen Punkten eine ganze Menge von Vorteilen:

- Direkter Zugriff auf die MCU-Funktionen
- Direkter Zugriff auf Peripherie

➤ Kein Performanceverlust durch Fehlstrukturierungen oder unmögliche Abstraktion

▷ *Sie als Entwickler können endlich alles machen, was Sie schon immer wollten:* ***alles***.

8.4.1 Beispiele

Sie haben mit dieser Distribution einige Beispiele für betriebssystemunabhängige Anwendungen erhalten. Sie befinden sich
unter „**/usr/local/carmeva/sw/boot/sw.arm9/app**". Sie unterliegen einem Buildsystem, welches auf GNU-Make basierend, allen Anforderungen gerecht werden sollte. Im Verzeichnis „**/usr/local/carmeva/sw/user-applications/...**" befinden sich die Startskripte für diese Programme.

Verwiesen sei auch auf das Projekt „**LdLinux++**". Es handelt sich um einen Linux-loader der zeigt, dass systemnahe Programme auch in strukturierter Form in C++ entwickelt werden können. (Diese Aussage teilen viele Entwickler nicht)

8.4.2 Kompfortabeles Entwickeln mit kdevelop3

Auf dem Desktop sollten sich drei Symbole befinden, welche die folgenden Namen tragen:

➤ „Demo-LCD"

➤ „Demo-LEDs"

➤ „Demo-Custom"

Es handelt sich um drei Projekte aus dem gennanten Source-Verzeichnis, für die eine kdevelop3-Konfiguration erstellt wurde.
Im Menu „Build ▷ Build Project F8" können sie die Quellen kompilieren. Wenn sie das Projekt starten (zum Zielsystem übertragen) möchen, können Sie dies im Menu „Build ▷ Execute Program" machen.

8.4.2.1 Demo-LCD Dieses Projekt steuert das LC-Display des Evalboard direkt an.

8.4.2.2 Demo-LEDs Dieses Projekt steuer die LEDs des Evalboards direkt an.

8.4.2.3 Demo-Custom Ist als Template gedacht, mit dem Sie eigene Versuche machen können... . Es steuert zunächt nur eine der vier LEDs an und zeigt so das prinzipielle Vorgehen beim entwickeln eines hardwarenahen Programms.

Weiterführende Informationen zu diesem Thema: Das Datenblatt für das Evaluationboard; Das Datenblatt der MCU AT91RM9200; „ARM Architecture Reference Manual"; Dokunentation des **gcc** Projektes.