ARM&EVA / Tutorial

Peer Georgi
Conitec Datensysteme GmbH

March 1, 2006

Software development for embedded Linux, as for most other operating systems as well, takes place on several levels. For a well organized development structure as well as for the production, every development step on every level must be reliable and reproducible.

The enclosed software was designed for easy control of all processes related to the software development.

This tutorial gives you a step by step introduction in using the tools.

We assume that the enclosed CD-ROM was started successfully or the „carmeva-xxx" tools were installed in an existing Linux distribution.

Contents

This Document

In the following the terminology is introduced. We'll use screen shots looking like this:

```
This way instructions are marked. Special parameters are highlighted also.
The ↰ character indicated the Return key.
```

Hints that you should note are displayed like this:

```
Read this...
```

Note

Further information about certain themes are indicated like this:
„Further reading"

The tutorial is divided into two chapters. The Basics chapter is intended to give an overview. The Examples chapter presents examples that can be immediately tested with the enclosed Live CD-ROM. If you don't have the CD-ROM or are working with your own Linux distribution, you can download the tools from our download page at ***www.conitec.net***.

Have fun!

Part I
Basics

1 Executing your programs

Programs can be Linux applications as well as OS independent programs executed **in stead** of the Linux kernel. OS independent programs run on the MCU without operating system (see below). For executing Linux applications, reprogramming the flash ROM is **not necessary**. The preinstalled Linux distribution is designed to execute applications directly via LAN (Ethernet).

However when the ROM content is to be altered or OS independent programs are to be run, the boot loader must be removed first. This is described in detail under 2.2.2.

1.1 Linux applications

For starting Linux programs a NFS(**N**etwork **F**ile **S**ystem)-Share can be used. For this, an arbitrary directory of the development system (preferably a Linux system) is mapped into the directory structure of the target system. This method offers short development cycles and total transparency of the file transfers.

For permanently storing files in the flash ROM, it has to be reprogrammed, which is described in a further chapter (see 3.4).

1.1.1 Remote Shell via Telnet

The preinstalled Linux distribution comprises a Telnet server. This way any Telnet client can access the MCU module. A password is not required. Under any shell just enter:

```
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.



/ #
```

As soon as the Telnet connection is opened, all installed commands can be used.

1.1.2 Connection via NFS

Connect to the target via telnet like descripted above. To create an NFS connection you will require information about the IP-network configuration of your development pc. In case your development pc has been configured via DHCP you may call „**ifconfig"** in a root shell to find out the IP address.
Assuming your development uses the IP address „**192.168.1.2**" you have to do the following on your device (via telnet):

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

The device should be connected with your development pc now. All files copied to „/mnt/target-transfer" will be visible / readable / writeable and possibly executable at device side.

1.2 OS independent programs

OS independent programs are transferred by the MCU firmware immediately after MCU reset. For this, booting from ROM must be prevented. If the ROM contains bootable content, the boot loader must be removed first (see 2.2.2).
Several transfer channels are available:

- V24, using the XModem protocol.
- USB, using first the USB-DFU protocol, then the LDBA protocol.

The MCU supports both channels for receiving data and programs. On the development system, the "*ebStartUp"* program is used for transferring an arbitrary program or data file to the target RAM, and executing it. "*ebStartUp"* supports fast data transfer via USB. The startup process can be completely controlled by the development system. Alternatively, the XModem protocol can be used for transferring and executing programs up to 16 KByte in size. For details please refer to the "*ebStartUp"* and MCU manuals.

Further reading: „ebStartUp" Manual.

2 Programming the Flash ROM

The MCU module comprises a programmable flash ROM device. It can be used for automatically starting programs, and can be programmed in the following ways:

1. Flash programming without Linux by using the "*ebTools"* .
2. Flash programming under Linux.

2.1 Flash programming under Linux

The flash ROM can be programmed under Linux because the flash device is supported by the Linux kernel. The "*ebFlashSetup "* program generates the data for "*ebStartUp"* as well as a flash image file that has the same size as the flash ROM. This file can be directly [1] written into the flash ROM. In this context please refer to the „**/dev/mtdX**" devices that are documented in the Internet.

Further reading: Linux kernel sources; AT91RM9200 data sheet.

[1] The bootable content ID must be taken into account - see „Vector 6". (MCU data sheet sec. „7.3 Boot loader")

2.2 Using the "*ebTools"*

The flash ROM is programmed by an application running on the MCU. At first all data to be programmed have to be transferred to the target RAM, regardeless of the operating system. It's the same process for programs to be executed in stead of the Linux kernel.

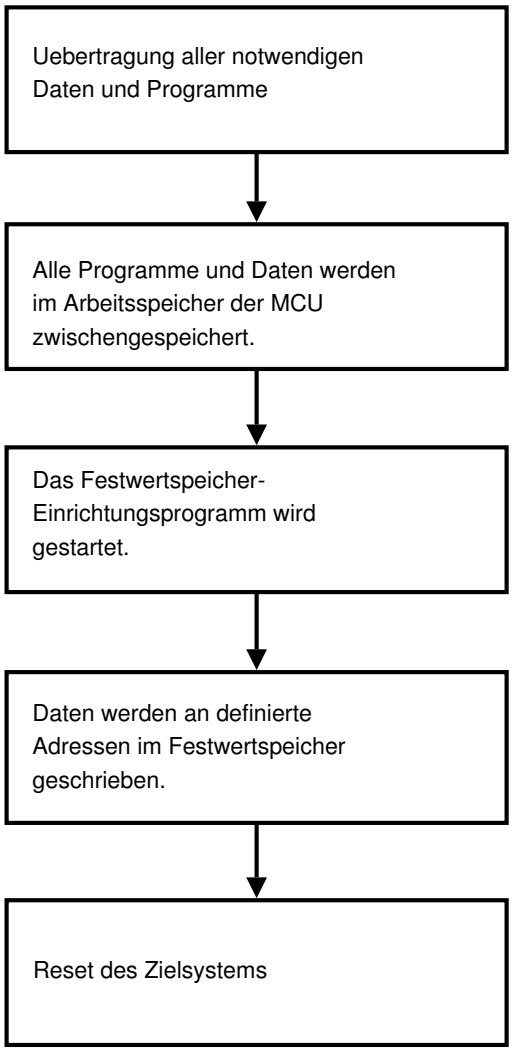

Figure 1: Flash programming process (controlled by "*ebStartUp"*)

"*ebFlashSetup "* was designed to make programming the flash ROM as easy as possible. It calculates the ROM target addresses and works together with "*ebStartUp"* . This way the flash configuration process can be automated and controlled on the development system. For details please refer to the "*ebFlashSetup "* manual. Examples for "*ebFlashSetup "* project files are included. Especially the "*ebFlashSetup "* project file in the „**recovery**" directory may serve as an example.

Further reading: „ebFlashSetup” manual.

2.2.1 The boot loader

The boot loader is executed by the MCU firmware. It's supposed to configure MCU ports and to load data and programs from ROM into RAM. During this process it issues messages via the V24 (RS232) interface.

The boot loader source code in C is included and can be customized.

2.2.2 Removing the boot loader

In two cases removing the boot loader can be required:

1. Programs are to be executed that were transferred via the USB port.
2. The flash ROM has to be reprogrammed using "*ebStartUp"* and "*ebFlashSetup "* (basically the same situation as in 1.).

Note

Do not fear: You can't shoot your foot by removing the boot loader.

The MCU contains a **non removable** firmware that offers communication channels supported by "*ebStartUp"* .

The included boot loader is controlled through a console program via the V24 (RS232) interface. It can be removed by pressing the „**r**" key on the console immediately after reset. This has to be confirmed by pressing „**y**" after the load process.
This renders the flash ROM content **non bootable**. Thus the MCU attempts to receive a program after reset. For executing a program, it must first be transferred to the MCU (see 1.2). This procedure makes sense during development, because programs or distributions can be tested quickly **without reprogramming the ROM after every modification**.

2.2.3 Restoring the delivery configuration

The delivery configuration can be restored by reprogramming the flash ROM with „recovery data". For this the ROM content must be non bootable. Otherwise the boot loader must be removed (see the previous section 2.2.2).

In the „***recovery***" folder a "*ebFlashSetup "* project file can be found, together with the related data for restoring the MCU module to delivery configuration. The „***recovery***" script just has to be started by entering:

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

./recovery
```

Please note that a default MAC address is used in that case. The address is adaptable however by modifying the „*mac-address*" file in the „resources" subfolder.

Note

The MCU modules are delivered with unique MAC addresses. The MAC address on CD is a **default address**. So we recommend to safely store and use that address.

3 Adaptation

All modifications discussed in this chapter require a non bootable ROM content and possibly reprogramming the ROM. For that reason please make sure that you've understood the previous chapters, and have removed the boot loader according to 2.2.2.
The Linux distribution preinstalled on the target consists mainly of the **Linux kernel**, the „**busybox**" project and some **shell scripts**. Additionally some parameters relevant for Linux startup can be configured, especially:

- Boot loader
- MAC adddress
- IP address / Linux command line

All components can be adapted to any application due to the availability of the source code. The following figure displays the various layers.
The ROM content (flash image) consist of a packet of the mentioned components created by "*ebFlashSetup* " .

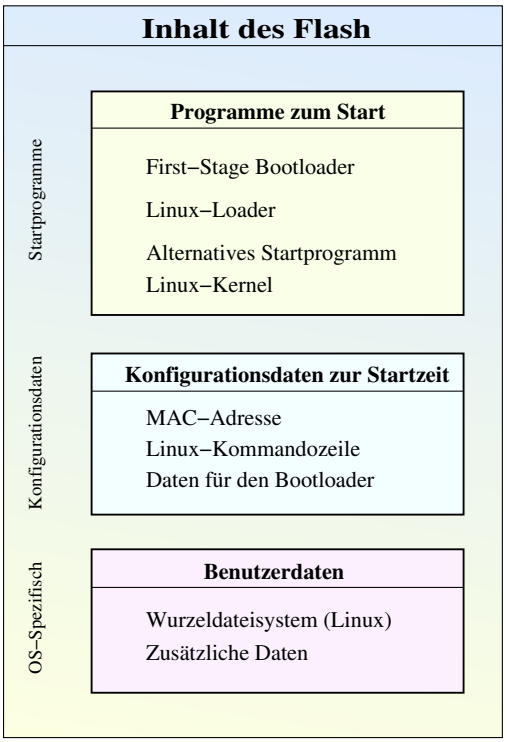

Figure 2: flash ROM content

The various configuration layers are represented by different project parts contained in several subdirectories on the development CD.

3.1 Startup programs

The system programs for starting the MCU module are available in source code, which can be found in „**/usr/local/carmeva/sw/boot/sw.arm9**".
This way all programs can be adapted. For configuring the preinstalled Linux however **no adaption is necessary**.

3.2 Startup configuration

The configuration parameters are available in simple files (mostly text files) that are interpreted by the boot loader or Linux kernel.

3.3 User data

The user data comprise the root directory of the Linux distribution. For further information that is independent of the operating system, further data blocks may be used.

3.4 Adapting Linux

The included CD-ROM contains a Linux distribution in the directory „**/usr/local/carmeva/sw/linux/examples/custom-linux**". It uses a default configuration, but can be adapted easily and can be used as a starting point for customization.
The directory contains the following:

- **ReadMe** - as it says
- **start/** - a **make script** for testing the distribution without overwriting the flash ROM
- **setup/** - a **make script** for writing the distribution into the flash ROM
- **ressources/** - contains everything needed for the Linux distribution

The usage of the files is discussed in the next chapter.

3.4.1 Adapting the IP address and MAC address...

Please refer to 4.

3.4.2 Adapting the root directory...

Please refer to 5.

Part II
Examples

4 Customized Linux „step by step”

This chapter will guide you through adapting and testing the Linux distribution. The directory structure is explained. We recommend to look closely into the different steps (including script and make file sources) for understanding the system.

The goal of this chapter is that the line
„**...This line was written by myself...**”
shall appear on the display and be output on the serial console. Additionally the IP address is to be changed to **92.168.0.25**”. This chapter uses the installation on the included Live CD and requires a V24 cable as well as an USB client cable already connected to the development system.
At first, the overview:

1. Remove the boot loader if it still exists
2. Make adaptations
   (a) Adapt the Linux command line
   (b) Adapt the root file system („**rootfs**”)
3. Test the Linux distribution
4. Program the Linux distribution into flash ROM

4.1 Removing the boot loader

If the ARM module is still in the delivery state (boot loader enabled), at first the boot loader must be removed. If this is already done, this section can be skipped.
For removing the boot loader the terminal program is used.

4.1.1 Starting the terminal program...

The terminal program name („gtkterm”)[2] is entered as command into the command line window that appears on hitting „Alt+F2”. After resetting the evaluation board, messages should be displayed in the console.

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

4.1.2 Interactively removing the boot loader...

As described in 2.2.2 the boot loader is removed by hitting the „**r**” key immediately after reset.[3] The following screen should be displayed:

```
blah
```

After confirming with „y” the boot loader is disabled. After another target reset the MCU firmware displays a „C”. This is the start character for the XModem protocol and indicates that the MCU is waiting for a program to be transferred via the serial interface. In parallel the USB client is active and can be used for fast data transfers (see below).

4.2 Making adaptations...

In this example we have to adapt two parameters on two different layers:

- ➤ IP address: configured through the command line.
- ➤ Message at system start: configured in the „rootfs”.

All modifications are done in a Linux shell. Shell programs are for instance „konsole” or „xterm”.
First, change to the directory „**/usr/local/carmeva/sw/linux/examples/custom-linux**”. In the „**ressources**” subdirectory all data records of the distribution can be found.

```
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
```

- ➤ „linux.cmdline” - the Linux kernel command line
- ➤ „mac-address” - the configured MAC address of the Ethernet PHY
- ➤ „rootfs.image.gz” - the root file system archive
- ➤ „LinuxKernel” - the unmodified Linux kernel as generated in „**linux/arch/arm/boot/Image**” when compiling the kernel source

[3] Command are case sensitive!

➤ The rest are boot loader and auxiliary programs that are executed before the Linux kernel.

The shell scripts „use-prebuild-kernel” and „use-custom-kernel” are useful for future modifications and not used at the moment. At a closer look you’ll notice that „LinuxKernel” is a symbolic link to a preconfigured Linux kernel. This kernel is generated by „**use-prebuild-kernel**”. For future kernel modifications, „**use-custom-kernel**” can be used to change the link to a folder containing the modified kernel.

For the current example the following files are relevant:

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

4.2.1 Changing the IP adddress

The file „linux.cmdline” must be edited for this. It’s a plain text file, so every ASCII editor can be used. The command line structure is described in the Linux kernel documentation and self explaining. IP address and further network settings can be configured there.

The program "*ebLinuxCmdLine*" interprets this text file and creates the binary version reqired by the Linux kernel.

4.2.2 Adapting the root file system

This is a little more expensive. There are two ways to go:

1. Changing an existing file system
2. Creating a new file system

For changing or creating a root file system there’s a dedicated project in „**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs**”. The preconfigured root file system in delivery state can be found here.

Changing an existing root file system

In the following we assume that the archive file „**rootfs.image.gz**” exists. The following steps have to be performed:

➤ Decompressing the archive

➤ „Loop-Back mount” of the file „rootfs.image”

➤ Applying modifications

➤ Unmounting the „Loop-Back mount”

➤ Compressing the files

At first the archive is decompressed into its directory, by entering:

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

Now a file „rootfs.image” exists in the directory. This file contains the root file system in the form of an „ext2” partition. This partition can now be mounted.

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

After this, the root file system is monted under „Target” and displays the following content (or similar):

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

Executing a command at system start

For displaying a message at system start, the „Init script” must be modified. It’s contained in „**Target/etc/init.d/rcS**” and can be edited with any ASCII editor. The following is sufficient for our goal:

```
cd Target/etc/init.d
echo "echo ...This line was written by myself..." » rcS
echo "echo ...This line was written by myself... » /dev/tty1" » rcS
```

This adds two lines to the „rcS” file. The first line displays the message on the boot terminal (serial console), the second line redirects the output to „**/dev/tty1**” and thus causes the message to appear on the display.

The root file system modifications are finished for this example. Now it can be unmounted and compressed.

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

The file „rootfs.image.gz” now contains the modified content.

Further reading: busybox project documentation (**http://www.busybox.net**); shell programming.

Creating a new root file system

This is explained in detail in 5.

4.3 Testing the new distribution...

For testing the distribution it has to be transferred to the target system first. This is "*ebStartUp"* ’s job. Like all "*ebTools"* it is configured through a project file that is a plain text file. The "*ebStartUp"* program is very versatile and can start up not only the Linux OS, but also OS independent programs.

In the „**/usr/local/carmeva/sw/linux/examples/custom-linux/**” folder a project file for "*ebStartUp"* and a make script can be found. Enter the following for starting up the OS:

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

After the entry, the startup process can be watched in the terminal window.

Further reading: the make script; the "*ebStartUp"* documentation (**/usr/local/carmeva/doc/de/manual/ebSuite**).

4.4 Programming the new distribution into flash ROM...

For making the target bootable on its own, the ROM can be reprogrammed. This is done by "*ebFlashSetup "* in combination with "*ebStartUp"* . For this there’s also a prefabricated "*ebFlashSetup "* configuration file.

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

Here also the messages can and should be watched on the serial console because the programming process termination is only indicated there.

Further reading: the make script; the "*ebStartUp"* documentation (**/usr/local/carmeva/doc/de/manual/ebSuite**).

5 Customized rootfs „step by step”

The root file system („rootfs”) comprises all programs and data required for the OS. It’s also the location for your own programs, e.g. your target application. The root file system is a directory tree that differs only slightly from a general Linux distribution. The directory tree is stored in a Linux compliant file system. The following examples use the Ext2 file system.

At system start the root file system is used as a RAM disk. Target applications can access and modify all files and add new files until the RAM space is exhausted.[4]

Note

Do not fear: This section is lengthy, but your Arm+Eva distribution makes things as easy as possible!

5.1 Basics

For creating a root file system, the following parameters have to be considered:

- RAM disk size = size of the uncompressed file system
- Total memory required for the root file system content

Because the file system is compressed after modification, there are four sizes that you should know:

1. RAM disk size in bytes
2. Uncompressed root file system size in bytes
3. Unused space in the uncompressed root file system size in bytes
4. Compressed root file system size in bytes

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

5.2 Creating a root file system

At first an overview:

- ➤ Creating a file for the file sytem
  - Creating an empty file with the same size as the uncompressed root file system (e.g. 16 MByte = **16384** KByte)
  - Creating a file system in this file
- ➤ LoopBack mounting the file into an arbitrary directory
- ➤ Creating the essential data and settings
  - Creating an empty directory tree
  - Adding the essential components (device nodes, required shell commands, basic configuration files)
- ➤ Adaptation
  - Adapting the default configuration and adding individual configuration files (e.g. autostart for processes or applications)
  - Adding individual components (programs, data, kernel drivers, ...)
- ➤ Unmounting the file system
- ➤ Compressing the file system

For creating a new root file system from scratch there's a project in the „**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs**” directory. All processes are based on **make scripts**, but not totally automated. Thus you're allowed to understand and reproduce all steps. Each step is defined by a make script target, as you'll see below.

5.2.1 Prerequisites

At first make a backup of the file „**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/rootfs.image.gz**”.

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

All further processes will use this directory.

5.2.2 Creating a file system

An empty file for the file system is created with the make target „**image-file**”.

make *image-file*

You’ll now be asked for a size in KBytes. Thus for a 16 MByte file the right answer is: 16384. After that, the program „mkfs.ext2” will ask whether a file system is to be created and whether you are really sure about this. If you are, answer „y” two times.

```
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.
```

After that a 16 MByte file „**rootfs.image**” is created in the current directory and waiting for its „LoopBack” mount...

5.2.3 Loopback mounting the file system

To perform this step, enter:

```
make mount
```

The file is mounted in the previously empty directory „Target”. After this step „Target” should contain precisely one entry:

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

5.2.4 Creating the basic distribution

In this step we will:

- ➤ create the directory tree and the device nodes
- ➤ install the basic shell commands

```
make create
make basic_dist
```

The shell commands are taken from the Open Source BusyBox project. Its sources can be found in **„/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/ressources/busybox**”. The BusyBox configuration is described in the chapter 6. The „*basic_dist*” make target installs basic configuration files.

After these steps the root file system is basically ready. The only things missing are your application program and some adaptations of the configuration files.

5.2.5 Distribution profile

Different MCU applications affect the hardware and the root file system. So, different configuration files and programs are required for different projects.
For maintaining some order, the make script offers several targets working together with shell scripts. They are in the directory „**ressources/carmeva/**”. Here the subdirectory „factory” can be found where the differences of the root file system to its delivery state are stored.

The shell script contained herein modifies the configuration directory „**Target/etc**”. Look into the directory for further information. It’s used in the make script when entering:

```
maske factory_dist
```

In the same way you can modify the default root file system and add your own targets to the make script.

Further information: make script („**/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/Makefile**”) and exploring the directory structure.

5.2.6 Unmounting and compressing

After all those steps the root file system is ready and can be compressed.

```
cd /usr/local/carmeva/sw/linux/prepare.rootfs/rootfs    # if they aren’t already in place

make umount
make compress
```

Now the final file „**rootfs.image.gz**” exists in the current directory[5] and waits for its test.

[5] The „.gz” extension is automatically added by the compression process.

5.2.7 Testing the new root file system

The project located in „**/usr/local/carmeva/sw/linux/examples/custom-linux**” contains everything you need. The root file system in the „resources” subfolder is already established as a symbolic link to the file you've just created. So testing only requires entering:

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

After this all data will be transferred and your new root file system is running.

5.2.7.1 In come cases adapting the Linux command line is required. Here the root file system size and the maximum size of the compressed RAM disk is stored, and might have to be adapted to the current situation.

5.2.7.2 Example If an empty file of **12288** KBytes = 12 MByte was created in the beginning, the size has to be stored in the Linux command line. This size is equivalent to the size of the uncompressed file system and named *„ramdisk_size*”.

Additionally you might notice that the compressed file „**rootfs.image.gz**” is bigger than **4MByte** (for instance, 5000000 Bytes). Thus you have to modify the „*initrd*” parameter of the command line:

„**initrd=**<RamDiskAddressInMemory**,**SizeOfCompressedImage>”

The address in memory must **not change**, so only the second parameter has to be adapted:

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

6 Customized BusyBox „step by step”

BusyBox is an Open Source project that puts basic shell commands together in a single program („Multicallbinary”). This reduces the resource requirement because the program overhead only is required once and not for every shell command.

The project sources can be found in **„/usr/local/carmeva/sw/linux/prepare.rootfs/rootfs/ressources/busybox”**. The configuration is similar to a kernel configuration (see next section). The shell commands to be added or omitted can be selected. The selection influences the size of the executable „*busybox*” file and therefore the memory requirement of the root file system. The following entry initates the configuration dialogue:

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

A menu appears where changes can be entered. After that the settings are saved and the executable is built.

```
make
```

After compiling, a new „*BusyBox*” with different commands is available. The changes become effective when the root file system is created (see 5.2).

Further reading: BusyBox project documentation („**http://www.busybox.net**”); chapter 5 of this tutorial.

7 Customized Linux kernel „step by step”

The Linux kernel source is available in the „**/usr/local/carmeva/sw/linux/prepare.kernel/kernel**” directory. It’s a kernel as distributed by `http://maxim.org.za/AT91RM9200/2.6/`, but got some modifications for running on the ARM&EVA platform. It is configured for already using the required cross compiler (gcc-3.4.1).

7.1 Configuration

The configuration process is identical with a Linux kernel configuration on a PC and done in a shell.

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

The menue that appears offers several possibilities for kernel modifications. The many parameters and settings are not described here as they can be found in the Internet.

After finalizing all modifications and storing the settings, the build process can be started. The process is the same as on the PC.

```
make
```

This command starts the build process. After compiling and linking, the **„Image”** file can be found int the „**arch/arm/boot**” directory. This file is used directly for the Linux loader and does not require further modification.

7.2 Testing the new kernel

For testing the new Linux kernel we use the „**/usr/local/carmeva/sw/linux/examples/custom-linux/”** directory that is already known from previous chapters. At first, we have to make sure that the new kernel will be used for the Linux start in the future. The subdirectory „**resources**” contains a script for this, which changes the symbolic link „LinuxKernel” to the new kernel in „**/usr/local/carmeva/sw/linux/prepare.kernel/kernel/arch/arm/boot/Image**”.

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

After that, everything is ready for starting the target system. This is done as usual by entering:

```
make start
```

.

8 OS independent programs

As mentioned above, the MCU can be used to run programs that don't require an operating system (or are an operating system). The boot loader is an example for an OS indepentend program.

The folder „**/usr/local/carmeva/sw/boot/sw.arm9**" contains several programs that can be directly executed after MCU reset. Those programs can be modified and generated with the included GNU-make based build system (even directly from the Live CD-ROM).

8.1 Starting programs...

8.1.1 Example

The program „cArmEvaLEDs" accesses the 4 LEDs of the evaluation board. It is controlled via serial console with the „1"-"4" keys.

Starting the program

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

After resetting the target system, the program is transferred via USB and executed immediately. The output can be watched on the serial console.

8.1.2 Another Example

The program „cArmEvaLCD" controls the LCD directly.

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

After resetting the target system, the program is transferred via USB and executed immediately.

For **further information**, look into the shell script („start"), and in the "*ebStartUp"* project files (*.ebs).

8.2 Programming applications into ROM...

The program "*ebFlashSetup "* can generate executable ROM content from data and programs, especially OS independent programs.

For the cArmEvaLCD project, the configuration sample for "*ebFlashSetup "* can be found in the folder „/usr/local/carmeva/sw/user-applications/cArmEvaLCD_Flash". For programming it into ROM, just enter:

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

The program output can and should be watched on the serial console, as the end of the flash programming process is only indicated there.

For **further information**, look into the shell script („setup"), and into the "*ebFlashSetup "* project file („**presentation.prj**"). Also consulting the ebStartUp and ebFlashSetup manuals might pay off.

8.3 Developing applications...

Programs that run without operating system can be devloped in C / C++ just like normal applications. The compiler ist included in the ARM&EVA distribution. However, there are differences to be taken care of during development:

- ➤ No OS –> no OS functions –> no memory or file management.
- ➤ Standard libraries can not be used, at least not without modification.
- ➤ Programs must be linked to a fixed memory address - the link process is different.

  ▷ *The developer - that means you - is responsible for every function of the program.*

Those obvious drawbacks however are compensated by many advantages:

- ➤ Direct access to MCU functions.
- ➤ Direct access to all ports.
- ➤ No performance loss through abstraction layers.

  ▷ *The developer - that means you - can finally do what he/she wants:* ***everything***.

8.3.1 Examples

Some samples for OS independent applications can be found in the „**/usr/local/carmeva/sw/boot/sw.arm9/app**" folder. They are generated by a GNU MAKE based build system that offers sufficient flexibility for all purposes. The setup scripts for the samples can be found in the „**/usr/local/carmeva/sw/user-applications/...**" folder.

Please also have a look into the „**LdLinux++**" project. It's a Linux loader that proves the possibility to create OS addendums in a structured way in C++ - a claim that is disputed by many developers.

8.3.2 Easy development with kdevelop3

On the desktop three symbols can be found with the following names:

- „Demo-LCD"
- „Demo-LEDs"
- „Demo-Custom"

They are three projects from the source folder for whom a kdevelop3 configuration was created. The menu function „Build ▷ Build Project F8" compiles the sources. For starting a project (including transfer to the target system) please use „Build ▷ Execute Program".

8.3.2.1 Demo-LCD This project directly controls the LCD on the evaluation board.

8.3.2.2 Demo-LEDs This project directly controls the LEDs on the evaluation board.

8.3.2.3 Demo-Custom A template for your own experiments. It controls one of the four LEDs and thus demonstrates the development of a program accessing the hardware.

Further reading: ARM&EVA data sheet; MCU AT91RM9200 data sheet; „ARM Architecture Reference Manual"; **gcc** project documentation.