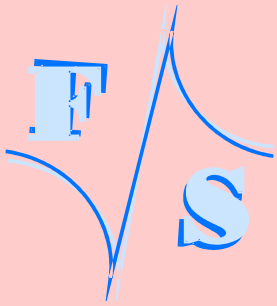
F+S Multi-Platform Linux

First Steps

Version 1.0
(2012-11-22)

fss5pv210



Linux

About This Document

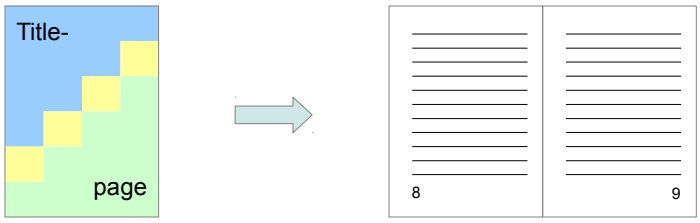
This document shows how to bring up F&S boards and modules under Linux, how to update firmware and how to use the system and the devices. It covers also compiling bootloader, linux kernel image and root filesystem as well as how to build your own applications for the device. The latest version of this document can be found at http://www.fs-net.de.

Remark

The version number on the title page of this document is the version of the document. It is not directly related to the version number of the driver software described herein.

How To Print This Document

This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.



Typographical Conventions

We use different fonts to emphasize the context of special terms:

`File names`

Menu entries

`Program code`

```
Listings
```

```
Board input/output
```


F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Germany

Phone: +49(0)711-123722-0
Fax: +49(0)711-123722-99

For file names we use placeholders for:

<Platform> that you have to replace with the name of your board like *armStoneA8*.

<Architecture> that you have to replace with the name of the target architecture like *fss5pv210*.

V<x>.<y> that you have to replace with the version *major* and *minor* numbers.

For binary package files we use format:

<Package>-<Board | Architecture | f+s>-V<x>.<y>.<extension>

For source package files we use format:

<Package>-<Packageversion>-<Board | Architecture | f+s>-V<x>.<y>.<extension>

History

Date	V	Platform	A,M,R	Chapter	Description	Au

V	Version
A,M,R	Added, Modified, Removed
Au	Author: CZ, DK, HF, HK, MK

Table of Contents

1 Introduction

F&S offers a whole variety of Systems on Module (SOM) and Single Board Computers (SBC). There are different board families that are named NetDCU, PicoMOD, PicoCOM, armStone, QBliss and nanoRISC. Linux is available for all of these platforms. However in the past, there was a separate Linux release for each platform.

Starting with V2.0, we are releasing combined versions of Linux that are suited for a whole group of platforms. These releases are therefore called Multi-Platform Linux Releases. Currently the boards armStoneA8, NetDCU14 and PicoMOD7A are supported by our Multi-Platform Linux, but more platforms will be added in the future.

The main purpose of our Multi-Platform Linux is that all supported boards share the same software versions and provide the same features, as far as the hardware allows. This makes switching from one platform to another rather straightforward and smooth and allows the customer to always choose the board that is suited best for a specific application without being confronted with a new development environment all the time.

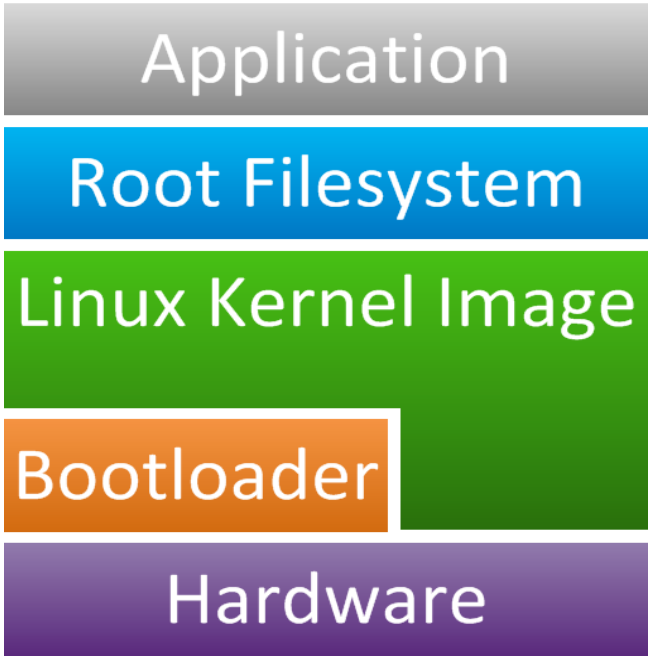


Figure 1: Components of a Linux system

If you look at Figure 1, you will see that a typical Linux system has several layers. At the bottom, there is the hardware of the platform. A bootloader initializes and configures the hardware and then starts the Linux Kernel that contains all the device drivers, controls the memory and data storage and handles process execution. All system and userspace programs, tools and data files are located in a filesystem that is usually called Root Filesystem. And finally the customer application is executing the function that the device is dedicated for.

So the Linux infrastructure is far more than just the kernel. The Linux distribution from F&S covers the following parts:

Hardware	The platform manufactured by F&S
Bootloader	The bootloader is split into two parts: a small stepping stone loader called NBoot that simlpy calls the main bootloader and the main bootloader itself, called U-Boot.
Linux Kernel	This is a modified mainline Linux kernel
Root Filesystem	We use a BuildRoot based root filesystem

In this document, we will show you how you get started with such a platform and how you compile your first programs.

2 Development Environment

To seriously work with F&S boards and modules, you need a Linux based PC for the software development, a terminal program to enter commands on the command line, a TFTP server to download files to the board and an NFS server to provide files and directories over the network.

We at F&S use:

- Linux PCs based on Fedora Linux as development machines. They are running as virtual machines in VirtualBox (Oracle) under a Windows host.
- Putty (terminal program) connected to the serial console of the board to enter commands for U-Boot and Linux.
- Tftp server and nfs server provided by the Linux distribution.

In addition we sometimes use the following Windows software

- DCUTerm (terminal program for Windows) to download U-Boot by serial line.
- TFTPD (by Philippe Jounin) as TFTP Server.

Note:

You will find the document `AdvicesForLinuxOnPC.pdf` in the software release archive that explains how to setup a linux based development machine. Or you can download it from the documents download section of our web server at:

http://fs-net.de

2.1 Starterkit

2.1.1 armStoneA8

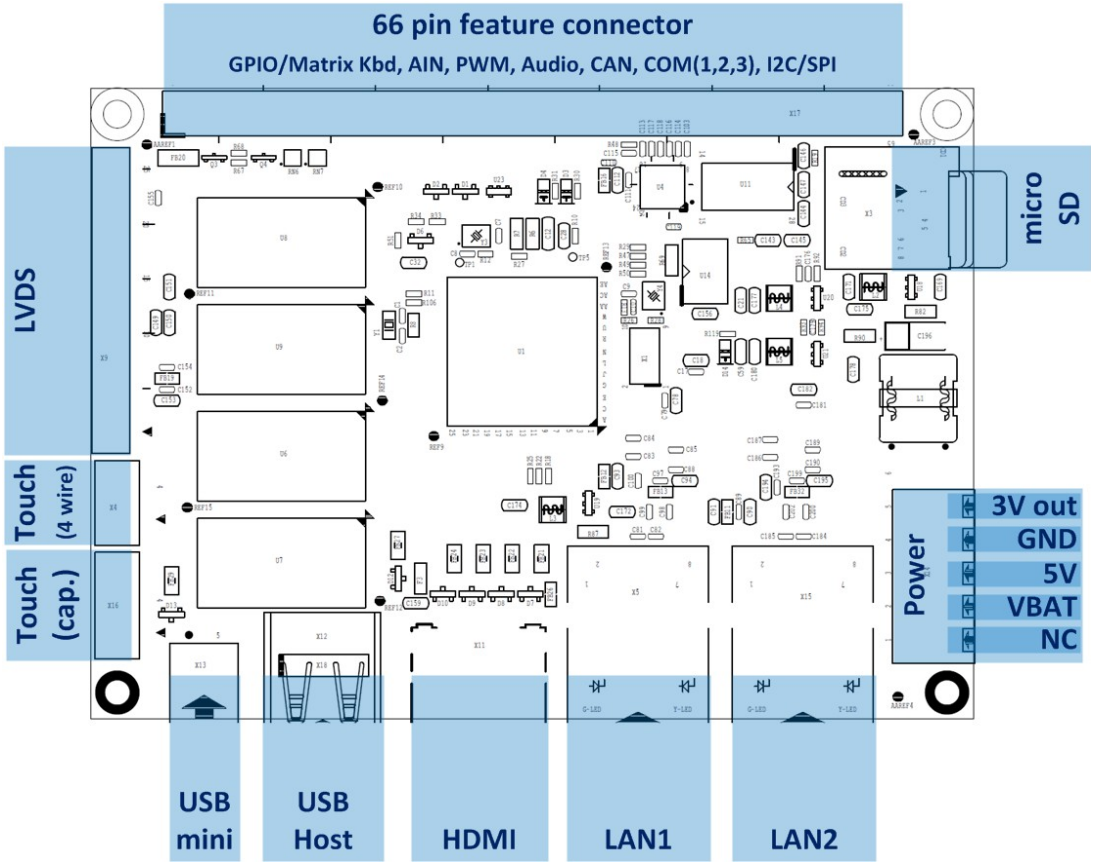


Figure 2: armStoneA8 Starterkit

2.1.2 PicoMOD7A

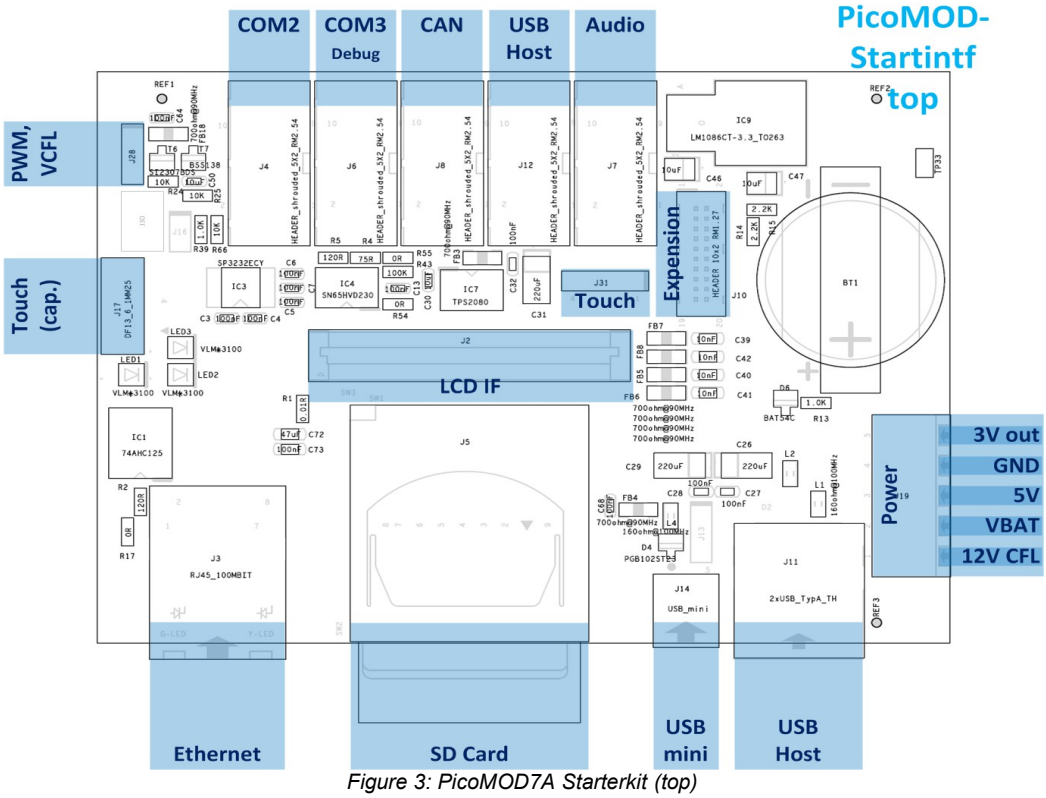


Figure 3: PicoMOD7A Starterkit (top)

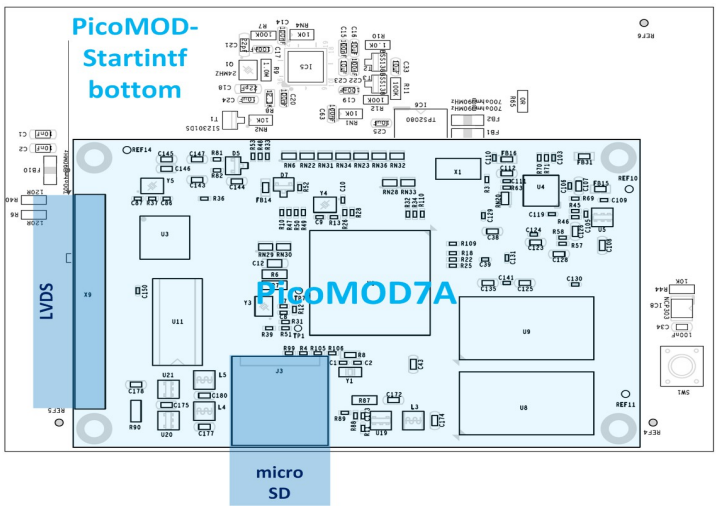


Figure 4: PicoMOD7A Starterkit (bottom)

2.1.3 NetDCU14

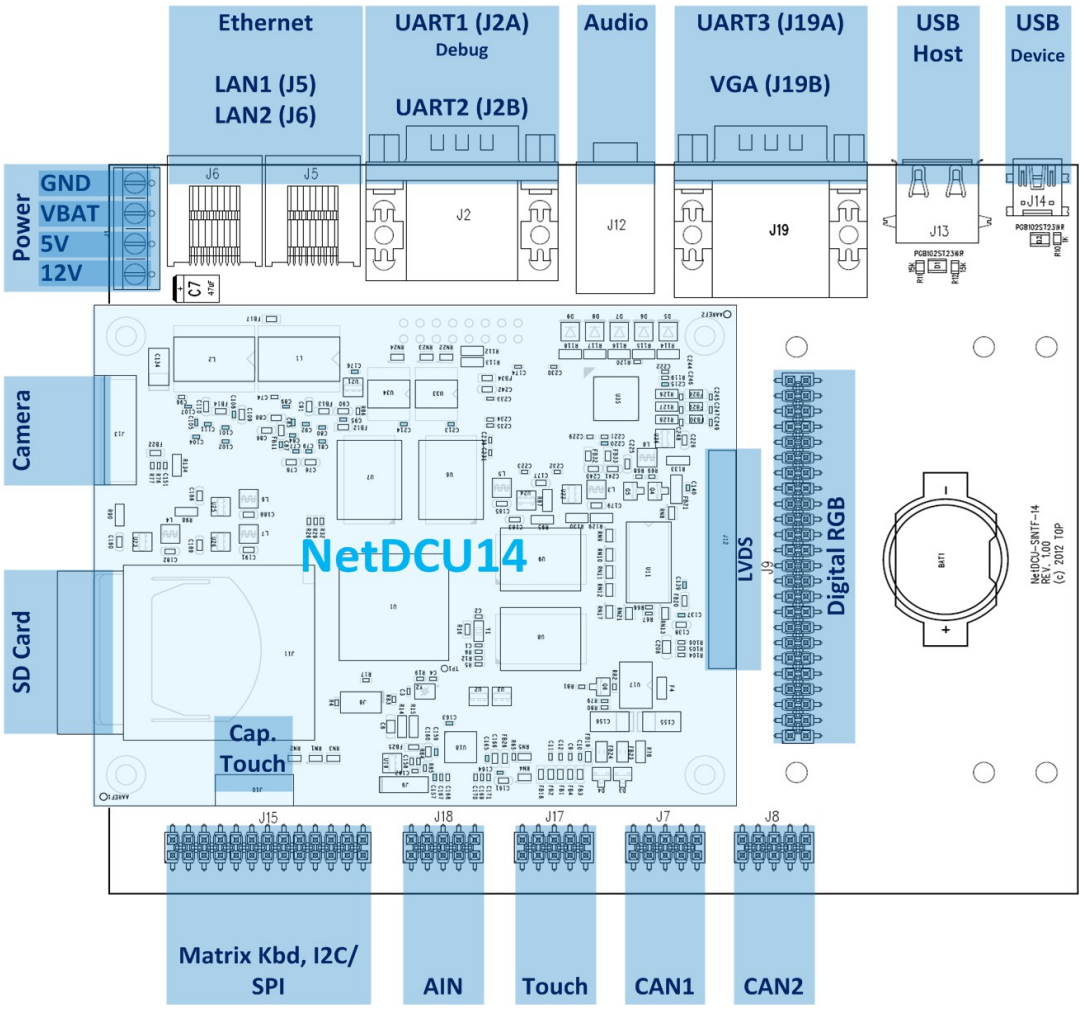


Figure 5: NetDCU14 Starterkit

3 Download Area

There exist two download areas on the F&S web server. The first is the document download area directly on

http://www.fs-net.de

where you have to select “Downloads” and the board family from the menu on the left side. And there is the software download area

http://download.fs-net.de

where you can always find our newest software releases. Some parts of this area are password protected, but if you purchase a Starterkit, you will also get the necessary information to log in to this area.

When you look at our new Linux releases, you will find two tar archives.

`multiplatform-linux-f+s-V<x>.<y>.tar.bz2`
: This is the main release itself containing all sources, the binary images, the documentation and the toolchain.

`sd-card-V<x>.<y>.tar.bz2`...
: If you copy the contents of this archive to an SD card, you can install our precompiled standard system in a very straightforward and comfortable way on the board.

The SD card archive is meant for people who just want to try a release first without having to download the quite large main archive. Its content is also contained in the main release archive, so if you want to download the main archive anyway, you don't need to bother with the SD card archive.

These tar archives are compressed with bzip2. So to see the files, you first have to unpack the archives

```
tar jxvf multiplatform-linux-f+s-V<x>.<y>.tar.bz2
```

This will create a directory `multiplatform-linux-f+s-V<x>.<y>` that contains all the files of the release. The files of the release often use a common naming scheme:

`<package>-<platform>-V<x>.<y>.<extension>`

With the following meaning:

`<package>`.....The name of the package (e.g. uboot, linux, rootfs)

`<platform>`.....The name of a board, if the package is only valid on one board (e.g. armStoneA8); or the name of an architecture, if the package is valid on different boards of the same architecture (e.g. fss5pv210), or the string „f+s“ if the package is architecture independent. For a list of possible platforms and architectures see Table 1 and Table 2. Names that will be available in the future are shown in grey.

`V<x>.<y>`.....The major and minor number of the release (e.g. V2.0)

`<extension>`.....The extension of the package (e.g. .bin, .tar.bz2, etc.)

Download Area

<Platform>	Description
armStoneA8	SBC, Cortex-A8 @ 800MHz with NEON 100x72mm (PicoITX)
PicoMOD7A	COM, Cortex-A8 @ 1GHz with NEON 80x50mm
NetDCU14	SBC, Cortex-A8 @ 1GHz with NEON 100x80mm
PicoMOD6	COM, ARM1176 @ 533MHz with FPU 80x50mm
PicoCOM3	COM, ARM1176 @ 533MHz with FPU 40x50mm
PicoCOM4	COM, ARM926 @ 400MHz 40x50mm

Table 1: F+S Platforms

<Architecture>	Platform
fss5pv210	S5PV210: armStoneA8, PicoMOD7A and NetDCU14
fss3c64xx	S3C64xx: PicoMOD6 and PicoCOM3
fss3c2416	S3C2416: PicoCOM4

Table 2: F+S Architectures

The following table lists the files that you get after unpacking the release archive.

Directory/File	Description
binaries/	**Images to be used with the platform directly**
nbootv210_18.bin	Architecture specific NBoot
uboot-<Architecture>-V<x>.<y>.nb0	U-Boot (bootloader) image
zImage-<Architecture>-V<x>.<y>	Compressed kernel image (can be used as is with U-Boot)
rootfs_std-<Platform>-V<x>.<y>.ubifs	Standard root file system (UBIFS format) to be stored in nand flash memory
rootfs_std-<Platform>-V<x>.<y>.ext2	Standard root file system (EXT2 format) to be used via NFS
rootfs_min-<Platform>-V<x>.<y>.ubifs	Minimal root file system (UBIFS format) to be stored in nand flash memory
rootfs_min-<Platform>-V<x>.<y>.ext2	Minimal root file system (EXT2 format) to be used via NFS
install.scr	Install script (U-Boot autoscript image)
sources/	**Configurations and sources**
u-boot-<Packageversion>-f+s-V<x>.<y>.tar.bz2	U-Boot source with modifications
linux-<Packageversion>-f+s-V<x>.<y>.tar.bz2	Linux kernel source with modifications
buildroot-<Packageversion>-f+s-V<x>.<y>.tar.bz2	Buildroot package with modifications
toolchain/	**Cross-Compilations toolchain**
fs-toolchain-4.6.3-armv6-vfp.tar.bz2	F&S toolchain to use with fss5pv210 and fss3c6410
mkimage	Program needed for autoscript images
examples/	**Short example programs**
can.tar.bz2	CAN tools to test CAN
gpio.tar.bz2	GPIO usage from userspace

spi_xxx	SPI example...
doc/	**Documentation**
FirstSteps.pdf	First Steps document
<Platformname>-Hardware_eng.pdf	Hardware description
AdvicesForLinuxOnPC.pdf	Document describing how to install servers and tools on a Linux PC to be used with F&S Linux boards.
sd-card/	**Files to copy to SD card**
nbotv210.bin	Architecture specific NBoot
ubotv210.nb0	Architecture specific Uboot
zImage-<Architecture>	Architecture specific Linux Kernel image
rootfs-<Platform>.ubifs	Board specific Root File System
install.scr	Same as install-fss5pv210-V1.0.uboot
/	
Readme.txt	Release notes
buildbin.bash	Script file to build all binary packages (used by F&S to build the release packages)
mktar.bash	Script to generate install.scr and sd-card directory (used by F&S when packaging the release)

Table 3: Content of the created release directory

Remark

The files in the subdirectory `sd-card` are actually only symbolic links to the according files in the `binaries` directory. However if you copy the content to an actual SD card, the referenced files will be copied and you get fully normal SD card content.

If you have problems unpacking the `sd-card` directory, for example on a Windows based system that does not know about symbolic links, you can also unpack the separate SD card archive, that just contains this directory.

4 Bringing up the system

When you get a Starterkit from F&S, the Linux system is usually pre-installed. In this case you can skip this chapter. But if you are switching over from an other operating system, if you are upgrading from a previous release or if your board is empty for some other reason, the following section describes how to install the provided standard images of the release on your platform.

Installing the Linux system involves four steps:

1. Check the NBoot version and erase flash

   NBoot is a small stepstone bootloader that is running before the main bootloader. It is the same for Linux and Windows CE and always remains on the board even if the whole flash memory is erased. By default NBoot is rather invisible and just loads and starts the main bootloader when the board is switched on. But it can also be used to download and store a new bootloader.

2. Install the Linux bootloader U-Boot

   U-Boot is the main Linux bootloader. Usually it simply activates the hardware, loads the Linux kernel from flash to RAM and executes it. But it is also used to download and install the Linux kernel and the Linux root filesystem. It can also boot the board from different devices, for example from a server across the network.

3. Install the Linux kernel image

   The Linux kernel image is the operating system of the device. It provides the device drivers, filesystems, multitasking and all I/O features that the board supports.

4. Install the root file system

   The root filesystem is the filesystem that you see after the kernel has booted. It contains the userspace programs, libraries and configuration files required to run the Linux system and applications. The default root filesystem supplied with the board has a Busybox for starting the system and to provide all standard command line tools, some ALSA tools for sound, gstreamer for audio and video processing, DirectFB with a few examples, a rudimentary X-Server to show some graphical user interface after startup and of course all the shared libraries like glibc.

There are many different ways how these four steps can be achieved, but the easiest way is to use an SD card.

By the way for these first tests you don't necessarily need a Linux PC. A Windows PC will do fine. However later, when actually developing applications, you'll definitely need a PC with a Linux OS. But this can also be a virtual machine. For example we are doing all Linux development on Fedora Linux in virtual machines as guests on Windows 7 and Windows XP hosts.

Copy all the files from directory *sd-card* to your SD card. Then insert the card into the slot of the board. You'll also need a terminal program (e.g. putty) to enter your commands and send them to the board.

Before you start, please power off your board. This is important to clear the RAM. Otherwise U-Boot may find an old kernel binary in RAM and execute it instead of installing the new system.

4.1 Check NBoot version

Connect the serial debug port (please refer chapter 3.1) to your PC. Use 38400 baud, 1 start, 1 stop bit, no flow control. Then in your terminal program, open the serial connection. Press and hold key 's' (lower case S). While holding this key, switch on power of the board (or press the reset button). This should bring you into the stepstone bootloader NBoot. You should see something like this (output is taken from PicoMOD7A):

```
F&S Nand Loader VN15 built Jul 24 2012 19:54:54
PicoMOD7A Rev. 1.30
256 MB RAM (2 chips) 128 MB FLASH 1000 MHz

Please select action
'd' -> Serial download of bootloader
'c' -> Load bootloader from SD card
'l' -> Load installed bootloader from flash
'E' -> Erase flash
'B' -> Show bad blocks
Use NetDCUUsbLoader for USB download
```

Listing 1: Nboot menu

Please verify the version of Nboot, here VN15. To be able to use the MultiPlatform Linux Release, you need at least version VN18 on armStoneA8, NetDCU14 and PicoMOD7A. So the above version would be too old and must be updated.

To update NBoot, just press the key 'N' (upper case n). This loads the new version from the SD-Card. Then press 'f' (lower case F) to save the new version to flash memory. Now restart the board with the sequence above and check again if the new version is running now.

4.2 Load and Save U-Boot

If the NBoot version is OK, then erase the flash by pressing 'E'. This removes everything that was on the board before. Don't be afraid, this won't erase NBoot itself.

Then press 'c' (lower case C). This will show some messages similar to this:

```
Card Info:
  Type: SD Card (Byte Mode)
  Specification Version: 0
  Relative Card Address: 1
```

```
  Operating Frequency: 12826923Hz (Normal Speed)
  Bus Width: 4 bits
  Manufacturer ID: 24
  OEM/Application ID: IN
  Product Name: 12221
  Product Revision: 0.3
  Serial Number: 1081883002
  Manufacturing Date: 09/2005
  C_SIZE: 1917
  C_SIZE_MULT: 5
  Block size: 512 bytes
  Total card size: 245504 blocks (119MB)
  Partial reads: yes
FAT Info:
  Start Sector = 99
  Sectors Per Cluster = 4
  Reserved Sectors = 8
  Number of Sectors = 245405
  Sectors Per FAT = 240
Trying 'ebotv210.nb0'...Failed
Trying 'eboot.nb0'...Failed
Trying 'ubotv210.nb0'...Success


>>> U-Boot image loaded (393216 bytes) <<<
```

Listing 2: Updating U-Boot via SD-Card

Now save U-Boot by pressing 'f' (lower case F). This should show

```
Saving U-Boot...Success
```

4.3 Load and Save Linux Kernel Image

As the U-Boot image is still loaded in RAM from the previous step, you can directly start U-Boot by pressing 'x' (lower case X). This will show something like this:

```
U-Boot 2011.12 for F&S

CPU:   S5PV210@1000MHz
Board: PicoMOD7A Rev 1.30 (2x DRAM, 1x LAN, 1000 MHz)
DRAM:  256 MiB
WARNING: Caches not enabled
NAND:  128 MiB
MMC:   SAMSUNG SD/MMC: 0
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   AX88796-0
```

```
Hit any key to stop autoboot:  0
```

Now the board is looking for an update script. This will fail, as we have no file name "update.scr" on our SD card.

```
---- Trying autoload from mmc0:1 ----
reading update.scr
Failed!
---- Trying autoload from usb0:1 ----
USB:   Register 1111 NbrPorts 1
USB EHCI 1.00
scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 0 Storage Device(s)
found
Failed!
---- No autoload script found ----
```

Then the board tries to boot an existing Linux system. This will also fail as we have valid image stored.

```
NAND read: mtdparts variable not set, see 'help mtdparts'
incorrect device type in Kernel
Wrong Image Format for bootm command
ERROR: can't get kernel image!
```

Now U-Boot tries to install a new system by starting the script "install.scr" from the SD card. This is actually found on the card. So this script is loaded and executed. From now on the installation procedure goes automatically.

```
---- Trying autoload from mmc0:1 ----
reading install.scr
Success!
```

Listing 3: U-Boot running installation script from sd card

We don't see the commands that are executed, just the output of them. The Linux kernel image is loaded and stored to the Kernel partition.

```
reading zimage

2757552 bytes read

NAND erase.part: device 0 offset 0x500000, size 0x300000
Erasing at 0x7e0000 -- 100% complete.
OK

NAND write: device 0 offset 0x500000, size 0x2a13b0
 2757552 bytes written: OK
```

Listing 4: U-Boot flashing Linux kernel image from sd card

4.4 Load and Save Root File System

The installation script automatically continues to create a UBI volume on the TargetFS partition and to load the root filesystem and store it there.

```
NAND erase.part: device 0 offset 0x800000, size 0x7800000
Erasing at 0x7fe0000 -- 100% complete.
OK
Creating 1 MTD partitions on "nand0":
0x000000800000-0x000008000000 : "mtd=5"
UBI: attaching mtd1 to ubi0
UBI: physical eraseblock size:   131072 bytes (128 KiB)
UBI: logical eraseblock size:    129024 bytes
UBI: smallest flash I/O unit:    2048
UBI: sub-page size:              512
UBI: VID header offset:          512 (aligned 512)
UBI: data offset:                2048
UBI: empty MTD device detected
UBI: create volume table (copy #1)
UBI: create volume table (copy #2)
UBI: attached mtd1 to ubi0
UBI: MTD device name:            "mtd=5"
UBI: MTD device size:            120 MiB
UBI: number of good PEBs:        960
UBI: number of bad PEBs:         0
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     0
UBI: available PEBs:             947
UBI: total number of reserved PEBs: 13
UBI: number of PEBs reserved for bad PEB handling: 9
UBI: max/mean erase counter: 1/0
No size specified -> Using max size (122185728)
Creating dynamic volume rootfs of size 122185728
reading rootfs.ubi

6451200 bytes read
6451200 bytes written to volume rootfs
```

Listing 5: U-Boot flashing root file system from sd card

By the way if you get some messages about bad blocks, like in the output above, don't worry. NAND flash memory often has some bad bits. This is taken care of by the software and these blocks are automatically skipped. A few bad blocks are completely normal and no reason for reclamation.

Finally the environment is saved, i.e. all the settings done in U-Boot, and the installation script stops with a message.

Bringing up the system

```
Saving Environment to NAND...
Erasing Nand...
Erasing at 0xe0000 -- 100% complete.
Writing to Nand... done
Installation complete

Please set/verify ethernet address(es) now and call saveenv
#
```

Now we only have to set the MAC address of the ethernet chip.

```
# setenv ethaddr 00:05:51:xx:yy:zz
```

Please replace xx, yy and zz with the numbers of the sticker on your board. This is a unique address that is only valid for your board. You must set this Ethernet MAC address!!! The default address that is used otherwise is the same for all boards and will definitely lead to problems in real networking scenarios.

You should also set other networking parameters here. They will be required if installing via Ethernet and of course later when working with the board. You should set the following four entries:

Environment Variable	Meaning
ipaddr	The IP address of your board
serverip	The IP address of your TFTP and/or NFS server (usually your development PC)
gatewayip	The IP address of your gateway; this is the device in your network that knows how to access the internet (usually your router)
netmask	The network mask used for your network (usually 255.0.0.0 for local network 10.x.x.x or 255.255.255.0 for local network 192.168.n.x)

Table 4: U-Boot network settings

Note:

Do not forget to save your modifications!

After you are done, save everything with:

```
saveenv
```

Now you're done. You can restart the board with command

```
reset
```

Summary

After starting into NBoot, you actually only have to press four keys: 'E' for erasing the memory, 'c' for loading U-Boot, 'f' for saving U-Boot, 'x' for starting U-Boot and the installation script. After the installation is done, set the MAC address(es) and save the environment.

That's it.

5 Updating

Updating of one of the system images (U-Boot, Linux kernel, root filesystem) later happens within U-Boot. Now even the U-Boot image can be replaced from within U-Boot.

Just load the file. Either from Micro SD card (fatload), USB stick (fatload), TFTP (tftp), or NFS (nfs). Then save the file to the appropriate MTD partition. If writing to UBoot or Kernel partition, you have to erase the partition first.

```
nand erase.part <partname>
```

Then you can write the new image by

```
nand write $(loadaddr) <partname> $(filesize)
```

If you write to a UBI volume, you don't need to erase anything first. Just write the new data with

```
ubi write $(loadaddr) <volumename> $(filesize)
```

5.1 USB Stick

Installation using a USB memory stick works rather similar to using the Micro SD card. Please also format the USB stick as FAT or FAT32 and store the following files from the sd-card directory into the root directory of the stick.

- zImage-<Architecture>
- rootfs-<Platform>.ubifs

The USB stick should be inserted in the USB slot on the board. Start the USB system with command

```
usb start
```

This will activate the USB port and scan for USB storage devices. It should show something like this:

```
   (Re)start USB...
  USB:   Register 1111 NbrPorts 1
  USB EHCI 1.00
  scanning bus for devices... 2 USB Device(s) found
         scanning bus for storage devices... 1 Storage Device(s)
found
```

then start download to RAM with:

```
fatload usb 0 $(loadaddr) zImage-<Platform>
  reading zImage-<Platform>

  2757552 bytes read
```

Now write the image from RAM to NAND Flash like described at the beginning of this chapter.

6 Switching Boot Strategies

U-Boot is capable of loading the kernel in different modes. During development it may be useful to have the (often changing) root filesystem on the PC itself and export it via NFS. Then U-Boot can prepare the Linux kernel to load the root filesystem via NFS. Among other things, this requires giving the network settings (ipaddr, serverip, gatewayip, netmask), root device (/dev/nfs) and exported directory (rootfs) on the NFS server on the command line. We have prepared an environment variable bootnfs that easily allows switching to NFS booting. It assumes that the exported directory is called rootfs in this case.

```
bootnfs=setenv bootargs console=ttySAC0,38400 $(mtdparts) \
  root=/dev/nfs nfsroot=/rootfs ip=$(ipaddr):$(serverip): \
  $(gatewayip):$(netmask) ro init=linuxrc
```

Listing 6: Linux kernel image bootargs for nfsroot

In the same spirit it is common usage to have a system running stand-alone with a root filesystem in the UBI volume rootfs on MTD partition TargetFS. Then among other things we have to give the type of root filesystem (UBIFS), the MTD partition (TargetFS) and the UBI volume name (rootfs) on the Linux command line. Again there is an environment variable to support this setting.

```
bootubi=setenv bootargs console=ttySAC0,38400 $(mtdparts) \
  rootfstype=ubifs ubi.mtd=TargetFS root=ubi0:rootfs ro \
  init=linuxrc
```

Listing 7: Linux kernel image bootargs for RFS as UBIFS

To activate one of these boot strategies, just run the appropriate environment variable and save the environment to make the change permanent. For example to boot via NFS the next time, call:

```
run bootnfs
saveenv
```

Please replace *ttySAC0* for your board by:

Board	Serial debug port
armStoneA8	ttySAC0
PicoMOD7A	ttySAC2
NetDCU14	ttySAC1

Table 5: Serial debug ports

7 Using the Standard System and Devices

The standard root filesystem is mounted read-only. Therefore you can't create files unless you go to a directory like /tmp. This is to make the system as stable as possible. If the root filesystem is mounted read-only, it is usually no problem to just switch off the power.

If you want to mount the filesystem read-write, just say

```
mount -o remount,rw /
```

But then it is saver to actually shut down the system with

```
halt
```

or restart with

```
reboot
```

Or you can remount the root filesystem back to read-only after applying the changes

```
mount -o remount,ro /
```

7.1 The Sysfs

Sysfs is a virtual file system provided by Linux. Sysfs exports information about devices and drivers from the kernel device model to user space, and is also used for configuration. You find configuration and device information under the class/ tree:

```
# ls /sys/class/
backlight     graphics      mem           rtc           spidev
vtconsole     bdi           i2c-adapter   misc
scsi_device   tty           block         i2c-dev       mmc_host
scsi_disk     ubi           dma           input         mtd
scsi_host     vc            firmware      lcd           net
sound         video4linux   gpio          mdio_bus      regulator
spi_master    video_output
```

You can examine wich major and minor number each device has, so you can create the corresponding device nodes

```
# cat /sys/class/i2c-dev/i2c-0/dev
89:0
```

Listing 8: Using Sysfs to examine devices

Or for example access the RTC subsystem

```
# cat /sys/class/rtc/rtc0/date
2012-08-09
```

Or you can see what devices are attached to different busses

```
# cat /sys/class/spi_master/spi1/spi1.0/modalias
mcp2515
```

On the next pages you will learn how to use Sysfs to access PWM or GPIOs.

7.2 GUI

The default GUI just shows a rudimentary X window desktop under a matchbox window manager. It contains a few icons, a starter menu and a clock. You can start a terminal program and a system load monitor from the starter menu. You can also click on the desktop icons and open them, but there are no further X applications installed. You can connect a USB mouse and/or USB keyboard (e.g. by using a USB hub) and then move the mouse cursor and type commands to the terminal window.

The whole system is not very functional and just demonstrates how a GUI could be implemented. We don't want to get a too large default root filesystem and we don't want to influence your decision of what type of GUI to use (QT, GTK, DirectFB, etc).

You can start some X applications on the command line:

```
export DISPLAY=:0
xclock &
xeyes &
xcalc &
fbv /usr/share/directfb-examples/*.png &
```

The X server is started with script file /etc/init.d/S35x11. So if you don't want this GUI started at every boot, just rename this script to something that does not start with S and two digits. For example rename it to X35x11. Then you can rename it back any time you want.

7.2.1 HDMICFG

Only on fss5pv210

With the hdmicfg configuration program a linux framebuffer can be output to HDMI. With the F+S Buildroot version this program is started on boot time with the init script *S17hdmicfg* located in */etc/init.d*. To start/stop this script run:

```
# /etc/init.d/S17hdmicfg stop
Stopping hdmicfg...
# /etc/init.d/S17hdmicfg start
Starting hdmicfg...
```

You can output linux framebuffer on a second output device at the same time by:

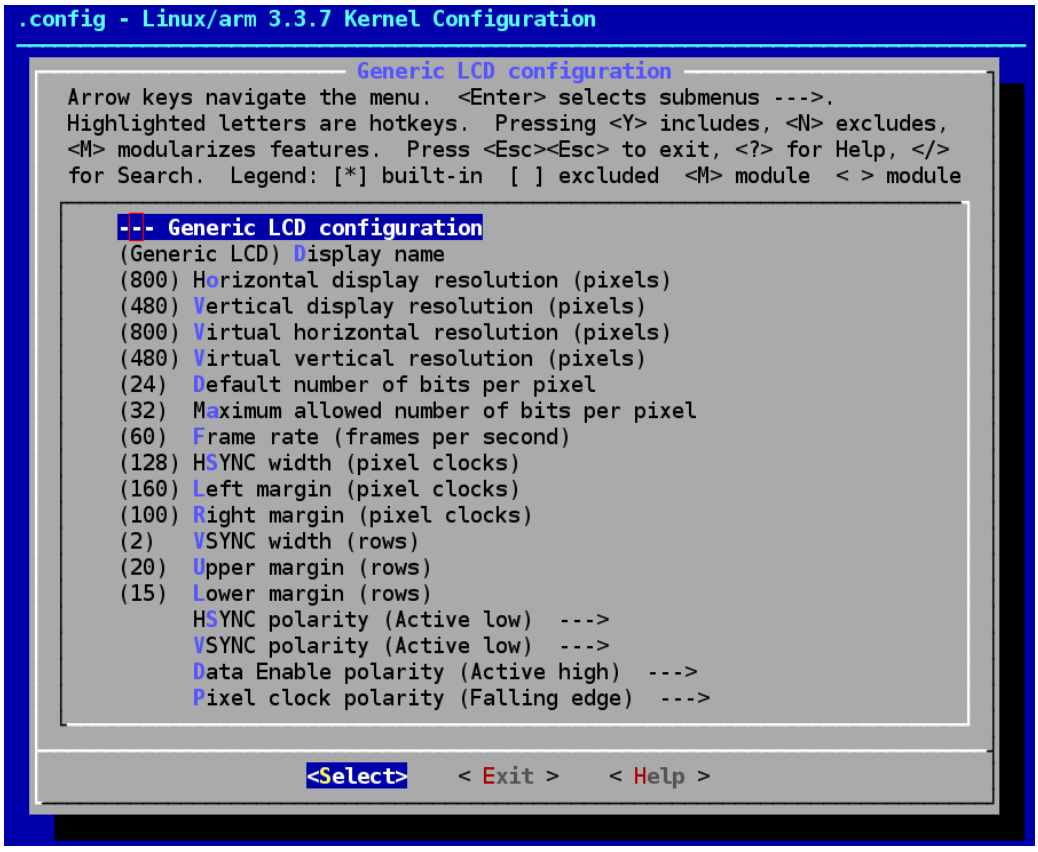
```
# hdmicfg -x 150 -y 100 -o /dev/video1
```

See the options for the hdmicfg program with:

```
# hdmicfg --help
```

7.2.2 Setup display

With the latest release setting up your display is done inside the Linux kernel image. So you need to recompile the linux kernel. Please install the F+S toolchain and extract the Linux kernel source. Then follow steps in chapter 11. From the kernel configuration menu setup your display parameters in the *Generic LCD configuration* menu.



7.3 Backlight

You can adjust the backlight brightness with (values from 0-255):

```
echo 100 > /sys/class/backlight/pwm-backlight.0/brightness
```

7.4 Ethernet

To activate the ethernet port in Linux, you have to configure the network device first. For example to use IP-Address 10.0.0.242, you can use the command

```
ifconfig eth0 10.0.0.242 netmask 255.0.0.0 up
```

Then you can use network commands, e.g.

```
ping 10.0.0.121
```

7.5 Telnet

If you want to use telnet to login from another PC, you have to start the telnet daemon

```
telnetd
```

However this service does not allow to log in as root from such an unsecure line. So you have to add a second user.

```
adduser -D telnet
passwd -d telnet
```

The first commands adds a user called "telnet". The second command sets an empty password for this user.

Now you can log in from another PC with

```
telnet <ipaddr>
```

under username "telnet".

7.6 DirectFB

There are a few DirectFB examples included in the standard root filesystem:

Application	Description

df_andi	Show some walking penguins; you can add or remove some
df_dok	benchmark 2D graphics (currently not accelerated!)
df_fire	Show some fire (may use the wrong color map)
df_knuckles	Show and rotate a skull
df_neo	Show spinning objects
df_particles	Show a particle stream
df_porter	Show different transparency
df_window	Show transparent windows, move with mouse

Table 6: DirectFB examples

Some of these programs must be stopped by pressing Ctl-C.

7.7 Serial

The serial ports are using special devices:

```
mknod /dev/ttySAC0 c 204 64
mknod /dev/ttySAC1 c 204 65
mknod /dev/ttySAC2 c 204 66
```

Listing 9: Creating UART device nodes

If you use the standard BuildRoot configuration, the devices are automatically created.

The default speed is 9600 bit/s with the exception of the default debug port (also used as linux console) which is configured for 38400 bit/s in U-Boot.

Example:

```
echo Hello > /dev/ttySAC1
```

7.8 SPI

Currently there is one SPI device supported on SPI0 bus:

```
mknod /dev/spidev0.0 c 153 0
```

Listing 10: Creating SPI device node

In the standard configuration, these devices are automatically created. This device can be used with the usermode spi driver (also called spidev).

There is also an SPI1 bus, but this bus is dedicated to the CAN controller and can't be used for other purposes.

An example to access the SPI bus is located in the examples subdirectory (spi_adc0831.c).

7.9 CAN

The CAN driver uses Socket CAN, i.e. the CAN bus is accessed as a network device, similar to an ethernet card. If the driver is available, you can find a can0 device when issuing the command

```
ifconfig -a
```

But better use the newer ip program as the older ifconfig does not know anything more detailed about CAN controllers.

```
ip link
```

Before you can activate this device, you have to set the baud rate. This is not done via the /sys filesystem anymore. But instead it requires the ip program. For example to set 125000 bit/s for CAN and activate, you would issue the command:

```
ip link set can0 up type can bitrate 125000
```

Listing 11: Activate CAN device

Now you can create sockets that access the CAN device. Some examples are provided in the examples subdirectory (can_tx.c, can_rx.c, candump.c, cansend.c).

7.10 SD-Card

if a Micro SD card is inserted in the slot on the board, it is detected automatically. Usually the card has a partition on it. Therefore to mount the device you need the following devices:

```
mknod /dev/mmcblk0 b 179 0
mknod /dev/mmcblk0p1 b 179 1
mknod /dev/mmcblk1 b 179 8
mknod /dev/mmcblk1p1 b 179 9
```

Listing 12: Creating mmc device nodes

In the default configuration, these devices are created automatically after you insert the card.

Then you can mount and un-mount the device:

```
mount /dev/mmcblk0p1 /mnt
umount /mnt
```

7.11 USB-Stick (storage)

If a USB memory stick is inserted, it is available like a standard hard disk. As there is usually no hard disk on the PicoMOD7A, it is found as /dev/sda. If you have partitions on your USB stick, you have to access them as /dev/sda1, /dev/sda2 and so on. Otherwise you can use /dev/sda directly. For example

```
mount /dev/sda /mnt
```

mounts the stick under the /mnt directory.

7.12 Touch

You can connect a 4-wire resistive touch or a capacitive touch (Atmel mxt224) to your device. If the system registered the capacitive touch controller then this is used as the default tslib device. From the boot messages you see:

```
Configure input for: default-event0  S3C24XX TouchScreen
```

When starting Linux for the first time you need to calibrate your touch with:

```
mount -o remount,rw /
ts_calibrate
```

after that you can test with:

```
ts_test
```

To use the touch with Xserver you need to reboot after the calibration or restart with:

```
/etc/init.d/S35x11 stop
/etc/init.d/S35x11 start
```

7.13 RTC

You need following device node:

```
mknod /dev/misc/rtc c 254 0
```

Listing 13: Creating rtc device node

Setting date:

```
date -s '2012-01-25 10:18:00' +'%Y/%M/%d %T
```

Save current date to RTC:

```
hwclock --systohc
```

Access RTC:

```
/sys/class/rtc/rtc0 and /proc/driver/rtc
```

Note:

connect 3.3V to VBAT on your PicoMOD7A base board.

7.14 PWM

You can access PWM with the Sysfs system. Set full backlight with:

```
echo 254 > /sys/class/backlight/pwm-backlight/brightness
```

Set backlight off:

```
echo 0 > /sys/class/backlight/pwm-backlight/brightness
```

7.15 GPIO

You can setup and use GPIOs with the Sysfs system.

```
ls /sys/class/gpio
export       gpiochip137  gpiochip181  gpiochip226  gpiochip89
gpiochip0    gpiochip14   gpiochip188  gpiochip23   gpiochip47
…
unexport
```

Please refer the „PicoMOD7A GPIO Reference Card“ document to know how the pins of the PicoMOD7A correspondent with the Sysfs-GPIO system.

Example:

Configure GPIO8 (PicoMOD7A: J10#18 / GPH3_0) as output pin:

```
echo 155 > /sys/class/gpio/export
```

This creates a new directory in /sys/class/gpio:

```
ls /sys/class/gpio/gpio155/
active_low  direction   edge        power       subsystem
uevent      value
```

Set pin as output:

```
echo out > /sys/class/gpio/gpio155/direction
```

Set pin to high level:

```
echo 1 > /sys/class/gpio/gpio155/value
```

7.16 Sound

You need following device nodes:

```
mknod /dev/snd/controlC0 c 116 0
mknod /dev/snd/pcmC0D0c  c 116 24
mknod /dev/snd/pcmC0D0p  c 116 16
mknod /dev/snd/timer     c 116 33
```

You can use standard Alsa tools to play and record sound. There is a tool to test the sound output.

```
speaker-test -c 2 -t wav
```

This will say "Front left" and "Front right" on the appropriate line out channel. If you have a WAV file to play, you can use this command:

```
aplay <file.wav>
```

To record a file from microphone in (mono), just call

```
arecord -c 1 -r 8000 -f s16_le -d <duration> <file.wav>
```

To record a file from line in, you first have to switch recording from microphone to line in. This can be done with

```
amixer sset 'Capture Mux' LINE_IN
```

Then record the stereo file with high quality with

```
arecord -c 2 -r 48000 -f s16_le -d <duration> <file.wav>
```

To see what other controls are available, call amixer without arguments:

```
amixer
```

You can also use gstreamer to test sound.

```
gst-launch audiotestsrc ! Alsasink
```

And to play a WAV file with gstreamer, you can use the following command:

```
gst-launch filesrc location=<file.wav> ! wavparse ! alsasink
```

7.17 Video

A quick video test can be done with:

```
gst-launch videotestsrc ! fbdevsink
```

7.18 Pictures

There is a small image viewer program included called fbv. Just call it with the list of images to show. This will show a new image every second.

```
fbv -s 10 /usr/share/directfb-examples/*.png
```

To show possible program options use:

```
fbv --help
```

7.19 TFTP

There is a short program to download a file from a TFTP server. This can be rather useful to get some files to the board without having to use a memory device like SD card or USB stick. For example to load a file song3.wav from the TFTP server with IP address 192.168.1.35, just call

```
tftp -g -r song3.wav 192.168.1.35
```

7.20 SSH

You can connect to your device by SSH. Therefore you need to set a password for your root user

```
passwd
```

or create a new user by:

```
adduser <username>
```

Now you can connect via SSH by any host in the network with:

```
ssh <username>@<device-ip>
```

7.21 VNC

If you have no display attached or you want to connect by remote you can start the pre-installed (with the standard buildroot root filesystem) x11vnc program on your device by:

```
x11vnc
```

On any host in the network install a vnc-viewer program and connect to your board with:

```
vncviewer <device-ip>:0
```

8 Cross-Compile Toolchain

The cross-compile toolchain is needed to compile U-Boot, the Linux Kernel and the Buildroot package. We would recommend to install it globally, for example in a directory /usr/local/arm. First create the directory and unpack the file from the toolchain subdirectory.

```
su
mkdir /usr/local/arm
cd /usr/local/arm
tar jxvf fs-toolchain-<version>.tar.bz2
```

Listing 14: Installing Cross-Compile Toolchain

This will create a subdirectory fs-toolchain-<version> with the cross-toolchain. Then exit again from superuser:

```
exit
```

Now add the this directory to your global PATH variable and set an environment variable ARCH for compiling the Linux kernel:

```
export PATH=$PATH\:/usr/local/arm/fs-toolchain-<version>/bin
export ARCH=arm
```

(You probably have to edit some global or local bash profile to make these two environment changes permanent.)

9 Compiling U-Boot

The U-Boot source can be unpacked to your local home directory:

```
tar jxvf u-boot-<Packageversion>-f+s-V<x>.<y>.tar.bz2
```

This will create directory u-boot-<Packageversion>-f+s. It alread contains a predefined configuration for F&S platforms. Simply run

```
cd u-boot-<Packageversion>-f+s-V<x>.<y>
make <Architecturename>_config
```

Listing 15: Setup default u-boot board configuration

This will activate the correct header files. Now you can build the bootloader with

```
make
```

This will create the file ***uboot.nb0***, that can be downloaded to your board (either in NBoot or the currently installed U-Boot). The file is exactly 384KB in size (393216 bytes). The build process will also create a slightly smaller uboot.fs. This file can be used for serial download in NBoot only, but results in a slightly faster download there (due to the smaller size).

10 Compiling the Linux Kernel

The kernel source can be unpacked to your local home directory:

```
tar jxvf linux-<Packageversion>-f+s-V<x>.<y>.tar.bz2
```

This will create directory *linux-<Packageversion>-f+s-V<x>.<y>* with the Linux source code. It already contains a predefined configuration for F&S platforms. Please verify that you have set environment variable ARCH to the value "arm" or the make tool will search the wrong directory for the default configuration (see installation of the toolchain above). Now simply run

```
cd linux-<Packageversion>-f+s-V<x>.<y>
make <Architecture>_defconfig
```

Listing 16: Setup default linux kernel board configuration

This will create the .config file. Then just run

```
make menuconfig
```

to modify your settings and then run

```
make
```

The final kernel image is in

```
arch/arm/boot/zImage
```

11 Compiling Buildroot

The basic idea of the F&S' boards root filesystem is the so-called Buildroot environment. Buildroot is an open source project. It allows to build a root filesystem for an embedded system simply by using the original sources of the individual packages. The benefit of Buildroot is:

- Knows all the web sites where to download the packages
- Knows how to modify the packages to be cross-compiled
- Knows how to combine the packages in a single root filesystem (i.e. it knows about all the dependencies between the packages)
- Has a powerful menu system to select which packages should be included in the root filesystem

To compile an own root filesystem, you can unpack the provided Buildroot package to your home directory.

```
tar jxvf buildroot-<Packageversion>-f+s-V<x>.<y>.tar.bz2
```

This will create the directory with the source code. Switch to it:

```
cd buildroot-<Packageversion>-f+s-V<x>.<y>
```

It already contains two predefined configurations for each board: the standard configuration that we use for our default root filesystem and a minimal configuration.

Boardname	Configuration	Description
PicoCOM4	picocom4_min_defconfig	Minimal configuration with busybox and iproute2. Toolchain: fs-toolchain-4.6.3-armv4t
PicoCOM4	picocom4_std_defconfig	Standard configuration with busybox, iproute2, directfb, Alsa-Utils, Gstreamer, Xorg7 and Matchbox. Toolchain: fs-toolchain-4.6.3-armv4t
PicoMOD7A	picomod7a_min_defconfig	Minimal configuration with busybox and iproute2. Toolchain: fs-toolchain-4.6.3-armv6-vfp
PicoMOD7A	picomod7a_std_defconfig	Standard configuration with busybox, iproute2, directfb, Alsa-Utils, Gstreamer, Xorg7 and Matchbox. Toolchain: fs-toolchain-4.6.3-armv6-vfp
armStoneA8	armstonea8_min_defconfig	Minimal configuration with busybox and iproute2. Toolchain: fs-toolchain-4.6.3-armv6-vfp
armStoneA8	armstonea8_std_defconfig	Standard configuration with busybox, iproute2, directfb, Alsa-Utils, Gstreamer, Xorg7 and Matchbox.

		Toolchain: fs-toolchain-4.6.3-armv6-vfp
NetDCU14	netdcu14_min_defconfig	Minimal configuration with busybox and iproute2. Toolchain: fs-toolchain-4.6.3-armv6-vfp
NetDCU14	netdcu14_std_defconfig	Standard configuration with busybox, iproute2, directfb, Alsa-Utils, Gstreamer, Xorg7 and Matchbox. Toolchain: fs-toolchain-4.6.3-armv6-vfp

Simply run

```
make <Platformname>_std_defconfig
```

Listing 17: Setup default buildroot board configuration

or

```
make <Platformname>_min_defconfig
```

Note: <Platformname> has to be lower case letters.

This will create the .config file. Then just run

```
make menuconfig
```

to modify your settings. There it is especially important to set the path to the toolchain. If you installed it in /usr/local/arm, then everything will work automatically. If not, then go to item "Toolchain", select "Toolchain path" and enter the path to your toolchain (including the directory fs-toolchain-<version>). Then you can exit make menuconfig. Don't forget to save your new configuration by saying "yes" when asked. Then run

```
make
```

to build the whole system.

When Buildroot starts to work, it downloads all the required packages from the original websites, configures and makes them and finally builds the root filesystem image. The default configuration creates an ext2 filesystem suited to be mounted via NFS and a UBIFS image suited to be installed on the board directly.

Compiling Buildroot the first time may take quite a while, probably more than an hour even on a very fast computer, so don't be surprised. But later when only updating minor modifications need to be done, compilation is usually done in a few minutes.

The standard configuration is meant to experiment with the board. You can add and remove packages to see how these packages work. Adding packages should usually be straightforward. However removing packages is not always this easy, as Buildroot does not keep track of what files of the root filesystem belongs to which package. Therefore it can not take out such files and they remain present in the root filesystem even if the corresponding package is removed from the Buildroot configuration. Just ignore this and go with the new packages. If things get too mixed up, you can always do a full rebuild with

```
make clean
make
```

The minimum configuration is meant as a clean starting point for your own system. Start with the busybox only and then add one by one the tools and libraries that you need in your system. Avoid removing packages for the reasons explained above. Then you'll get an optimal and minimal root filesystem suited for your needs.

12 Hello World

First you need to install the cross-compilation toolchain like described earlier in this document. Then you setup your environment:

```
export PATH=$PATH:/usr/local/arm/fs-toolchain-<version>/bin
export ARCH=arm
export CROSS_COMPILE=arm-linux-
```

Listing 18: Setup cross-compile environment

You can check which cross-compiler you set by

```
which arm-linux-gcc
/usr/local/arm/fs-toolchain-4.6.3-armv6-vfp/bin/arm-linux-gcc
```

12.1 Create your source file

Change to your development directory, create a new one and change to it. Then create the source file like:

```
mkdir helloworld
cd helloworld/
vim main.c
```

Modify your source file with typical „hello world“ source code:

```
#include <stdlib.h>
#include <stdio.h>


int main (void) {
  printf("Hello World... \n\n");
  return 1;
}
```

Listing 19: Hello World application

12.2 Compile your source file

As you targeting to run your application on a F&S board, which is based on a ARM processor, you need to cross-compile your application with

```
arm-linux-gcc -o helloworld main.c
```

12.3 Run your application on the device

After you cross-compiled your application it is ready to run on the target. You can run it on the target from MMC, USB Stick or you can copy it to internal NAND flash by network and execute it with:

```
ifconfig eth0 <ipaddr> netmask <netmask> up
tftp -g -r helloworld <serverip>
chmod a+x helloworld
./helloworld
```

13 Debugging

Create an application like:

```
//DebugApp
#include <stdlib.h>
#include <stdio.h>

int main (void) {
      int a,b,c = 0;
      a = 1;
      b = 2;
      c = a + b;
      return c;
}
```

Copy the executable to your device and start GDB with:

```
# gdbserver <Host-IpAddress>:<Port> DebugApp
Process secondtest created; pid = 808
Listening on port <Port>
Remote debugging from host <Host-IpAddress>
```

On your development host run:

```
$ arm-linux-gdb
GNU gdb 6.8
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute
it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-build_pc-linux-gnu –
target=arm-fs-linux-gnueabi".
```

Then connect ro your remote device with:

```
(gdb) target remote <Remote-IpAddress>:<Port>
Remote debugging using <Remote-IpAddress>:<Port>
[New Thread 808]
0xb6f58d50 in ?? ()
```

Select the debug executable:

```
(gdb) file /workspace/DebugApp/Debug/DebugApp
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from /workspace/DebugApp/Debug/DebugApp...done.
```

Setup a breakpoint with:

```
(gdb) break 13
Breakpoint 1 at 0x83a8: file ../main.c, line 13.
```

Start execution with:

```
(gdb) continue
Continuing.

Breakpoint 1, main () at ../main.c:13
13        a = 1;
```

Step by:

```
(gdb) step
14        b = 2;
(gdb) step
15        c = a + b;
(gdb) display c
1: c = 0
(gdb) step
16        return c;
1: c = 3
(gdb) display c
2: c = 3
(gdb) continue
Continuing.

Program exited with code 03.
(gdb)
```

14 IDE - Eclipse

Install Eclipse on your development host with your Linux Distribution specific package manager system. On Fedora do:

```
sudo yum install eclipse
```

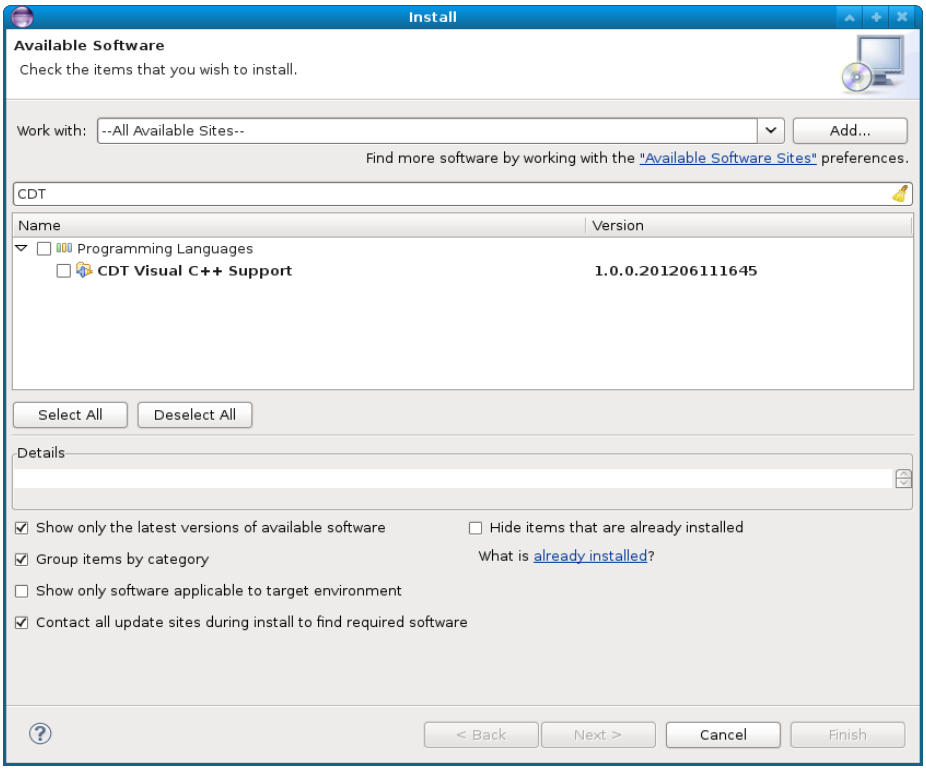
Then setup environment and start eclipse:

```
export PATH=$PATH:/usr/local/arm/fs-toolchain-4.6.3-armv6-vfp/bin
export ARCH=arm
eclipse &
```

Install Cross Development tools:

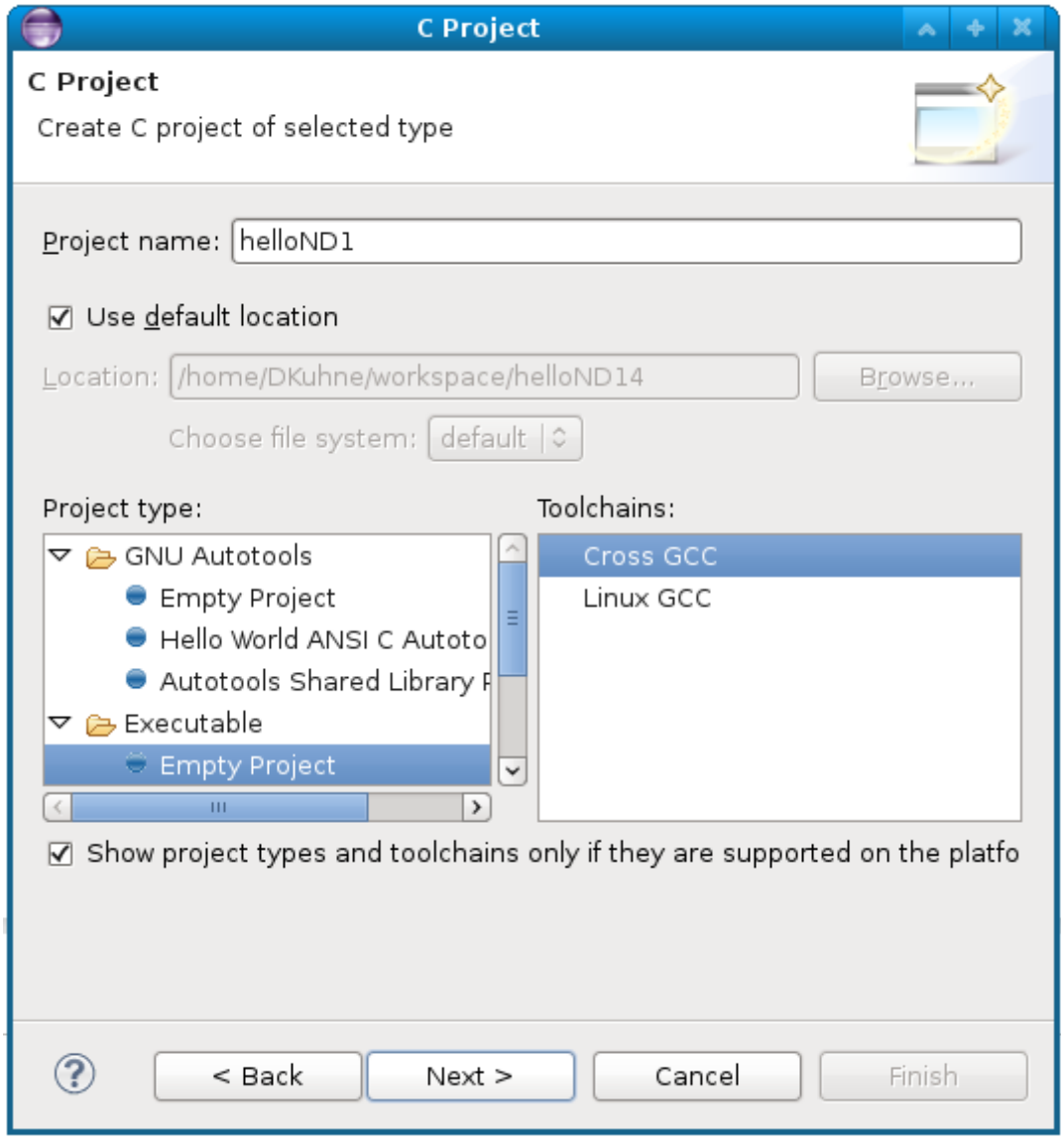
```
C/C++ Remote Launch
TCF Remote System Explorer add-in
Remote System Explorer End-User Runtime
C/C++ GDB Hardware Debugging
TCF C/C++ Debugger
C/C++ GCC Cross Compiler Support
Autotools support for CDT
CDT Visual C++ Support
```

From *Help – Install New Software*

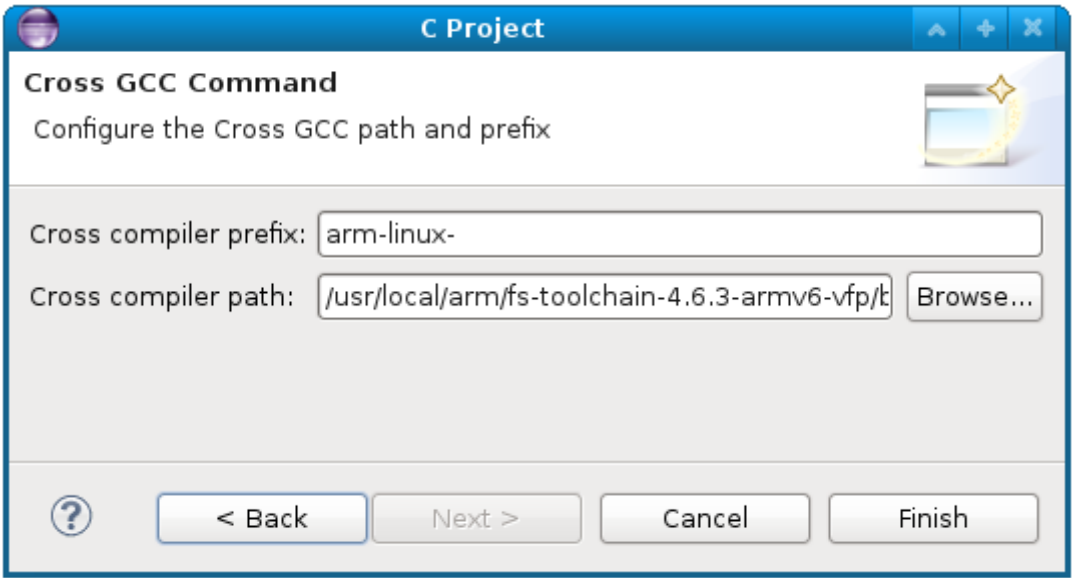


14.1 Create a Project

After all dependencies are installed within eclipse you can create a new project. From *File* select *New* and then C *Project*. Then click on *Next*. On the next dialog fill in a project name and select *Empty Project* from *Executable*. Select *Cross GCC* as toolchain.



Click on *Next* and setup toolchain:



14.2 Setup Properties

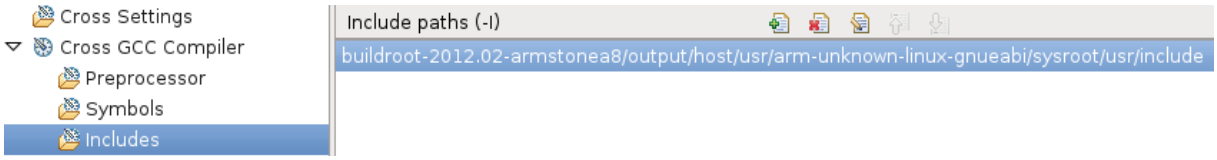
Do a right click on the project in the *workspace view* and choose *Properties* and then *C/C++ Build*.

Discovery Options: check *Compiler inovacation command* is set to *arm-linux-gcc*

Environment: check that PATH is set to fs-toolchain

Variable	Value	Origin
CWD	/home/DKuhne/workspace/helloND1/Debug	BUILD SYSTEM
PATH	/usr/local/arm/fs-toolchain-4.6.3-armv6-vfp/bin:/usr/lib/ccache:/usr/local/bin:/usr/bin	BUILD SYSTEM
PWD	/home/DKuhne/workspace/helloND1/Debug	BUILD SYSTEM

From *Settings – Cross GCC Compiler – Includes* add Buildroot sysroot path to *Include paths* (sysroot/usr/include):



From *Settings* – *Cross GCC Linker* – *Libraries* add Buildroot sysroot path to *Library search path* (sysroot/lib):

14.3 Add a source file and build the project

Do a right click on the project and select *New* – *Source File*. Write your `main` function.

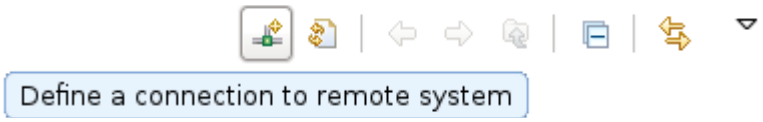
```
#include <stdio.h>
#include <stdlib.h>

int main (void){
      printf("Hello Universe.. \n");
      system("/bin/touch /home/helloND1.txt");
      return 1;
}
```

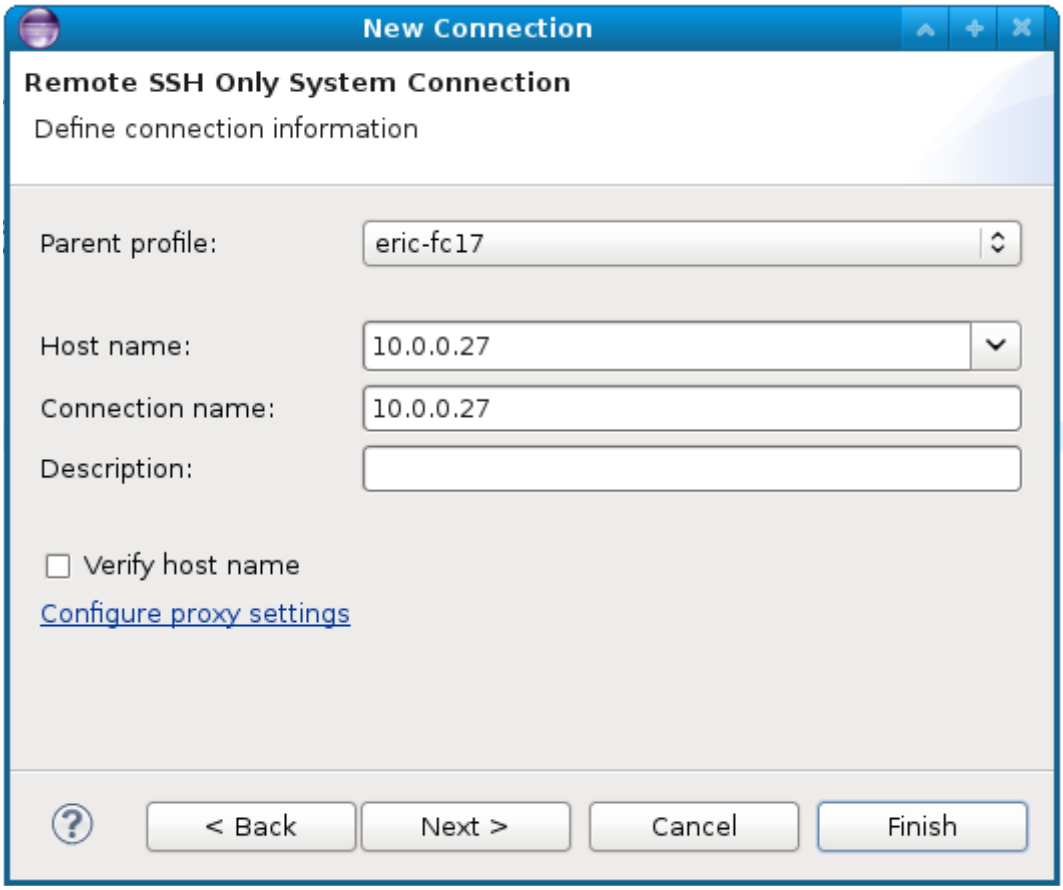
and build the application by right click on the project and choose *Build project*.

14.4 Remote connection

From *Window* – *Open Perspective* – *Other...* choose *Remote System Explorer*. Then click on *Define a connection to a remote system* and select *SSH Only*.

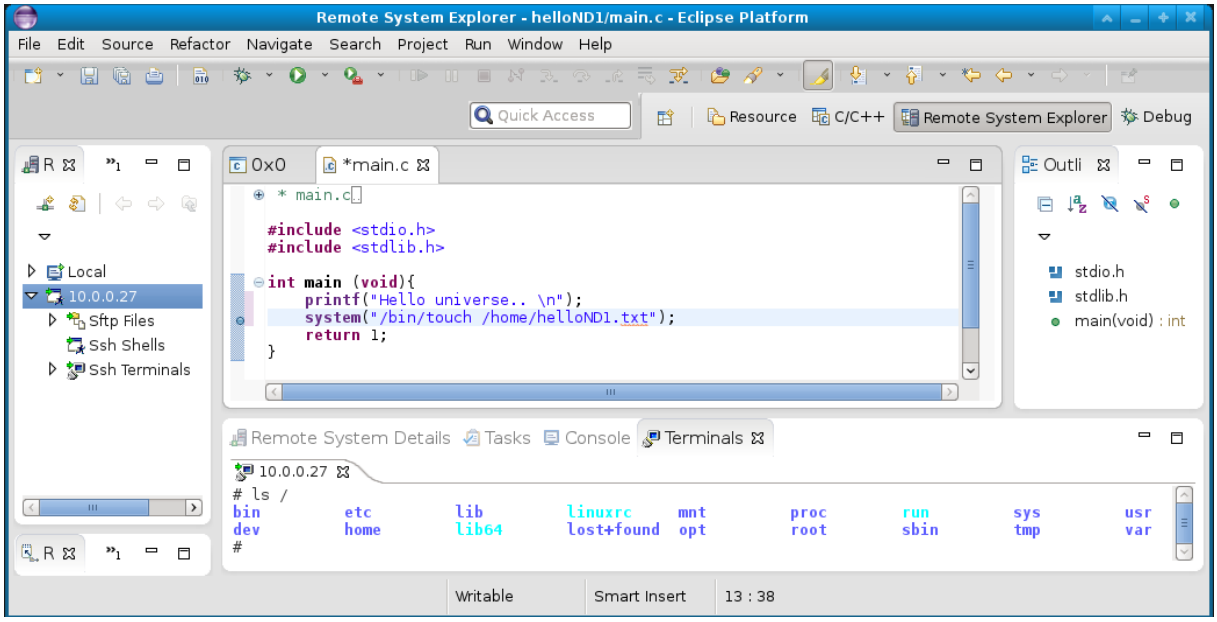


Enter Hostname/Ipaddress:



14.4.1 Connect and test

Do a right click on your remote host from the *Remote Systems* view and choose *Connect*. Enter User ID (root) and password (create a password for user root with `passwd` on the device). Then again, do a right click on *Ssh Terminals* and choose *Launch Terminal*.



14.5 Run your application

From *Run – Run configurations – C/C++ Remote Application* set *Connection* to your remote machine. Set *Remote Absolute File Path for C/C++ Application* to */home/<myapp>*. And set *Commands to execute before application* to *chmod a+x /home/<myapp>*. Then click on *Apply* and then on *Run*.

14.6 Debug your application

From *Run – Debug configurations – C/C++ Remote Application* set *Connection* to your remote machine. Set *Remote Absolute File Path for C/C++ Application* to */home/<myapp>*. And set *Commands to execute before application* to *chmod a+x /home/<myapp>*. Then click on *Apply* and then on *Run*.

15 Appendix

Listings

List of Figures

List of Tables

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.

F&S Elektronik Systeme reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S Elektronik Systeme makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S Elektronik Systeme assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Products are not designed, intended, or authorised for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S Elektronik Systeme could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S Elektronik Systeme product for any such unintended or unauthorised application, the Buyer shall indemnify and hold F&S Elektronik Systeme and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorised use, even if such claim alleges that F&S Elektronik Systeme was negligent regarding the design or manufacture of said product.