

Starterkit Manual

HiCO.SH7760 Linux 2.6 Starterkit

emtrion

© Copyright 2008 **emtrion GmbH**

All rights reserved. This documentation may not be photocopied or recorded on any electronic media without written approval. The information contained in this documentation is subject to change without prior notice. We assume no liability for erroneous information or its consequences. Trademarks used from other companies refer exclusively to the products of those companies.

Version 06/06/2008 19:45 - emdist-2.1

- 1 Introduction..... 6**
- 1.1 Starterkit contents 6
- 1.2 Some conventions in this manual..... 7
- 1.3 Starterkit Support..... 8

- 2 emDist Linux distribution 9**
- 2.1 Introduction..... 9
- 2.2 Development model..... 9
- 2.3 emDist web-interface and scripts..... 10
- 2.4 BSP files and directories..... 12

- 3 emDistVM virtual machine 13**
- 3.1 System requirements..... 13
- 3.2 Performance issues..... 14
- 3.3 Install VMWare Player 14
- 3.4 Installing the emdist VM..... 15
- 3.5 Sharing files with the Virtual Machine 17
- 3.6 Changing the default settings..... 19
- 3.7 Useful Tips 20

- 4 Getting Started..... 21**
- 4.2 Installing emDist for HiCO.SH7760..... 22
- 4.3 Start the emDist web interface..... 23
- 4.4 Install HiCO.SH7760 hardware..... 24
- 4.5 Connect to the target via serial port..... 25
- 4.6 Check that the network is working..... 26
- 4.7 Set IP address 27
- 4.8 Build target root file system 28
- 4.9 Configure NFS server 29
- 4.10 Run Linux kernel with NFS root mount 30

- 5 System build – the big picture 31**
- 5.1 Where do my SW project files fit in? 35

- 6 Using the toolchain..... 36**
- 6.1 Toolchain location and the PATH variable 36
- 6.2 Cross-compiling simple applications 37

- 6.3 Cross-compiling Qt applications..... 38

- 7 Debugging your applications..... 39**
 - 7.1 Remote debugging with plain GDB..... 40
 - 7.2 Remote debugging with DDD..... 41
 - 7.3 Start a debug session from the web-interface..... 42

- 8 Software packages 43**
 - 8.1 Building and installing packages 44
 - 8.2 Cross compiling open source packages 44

- 9 Flash boot and NFS boot..... 46**
 - 9.1 A quick look at the used filesystems 47
 - 9.2 Flash partitions..... 48
 - 9.3 Creating a JFFS2 flash partition 49

- 10 Working with the bootloader..... 50**
 - 10.1 Board identifier 51
 - 10.2 Boot mode jumper..... 51

- 11 Networking 52**
 - 11.1 Target network configuration..... 52
 - 11.2 Telnet server (telnetd)..... 53

- 12 System start-up..... 54**
 - 12.1 Device nodes on the target..... 54
 - 12.2 Start-up scripts..... 54
 - 12.3 Where to start my application? 54

- 13 Advanced usage of emDist scripts..... 55**
 - 13.1 Understandig the structure 55
 - 13.2 Macros..... 55

- 14 Hardware interfaces on HiCO.SH7760..... 57**
 - 14.1 Supported Hardware 57
 - 14.2 Flash partitioning..... 58
 - 14.3 Serial ports..... 58
 - 14.4 USB..... 59

14.5	Touchscreen.....	59
14.6	Real Time Clock.....	60
14.7	I2C Bus.....	60
14.8	CAN-bus (HCAN2)	61

1 Introduction

HiCO.SH7760 starterkit for Linux gives you a kick-start to developing embedded Linux software. Ready configured Kernel and collection of user space software, including graphics with Qt, serve as stable basis for your own extensions.

With the delivered Virtual Machine you can start development immediately on your Windows host. If you already have your choice of Linux, you can also easily install the BSP and Starterkit software on it. The BSP is designed to work with any modern Linux distribution.

1.1 Starterkit contents

The BSP consists of following components

- Linux kernel 2.6.15 for HiCO.SH7760
- Base userspace software (C/C++ libraries, busybox, etc.)
- Extension userspace software packages (Qt embedded, sample programs, networking utilities, etc)
- Preconfigured root file system, with well commented start-up scripts and configuration files.
- GNU cross compiling toolchain and libraries for HiCO.SH7760
- GNU cross debugger (`gdb`) for HiCO.SH7760
- Scripts and tools to build and configure customized filesystem and kernel images.
- VMware virtual machine (emDistVM), which contains a Linux distribution (Xubuntu) with a ready configured environment for development.

For a quick start, all above components are preconfigured and precompiled for HiCO.SH7760 in the BSP. Tools and scripts for building them yourself are also included in the BSP.

1.2 *Some conventions in this manual*

1.2.1 Path names

All of the given path names are given as relative paths to the BSP root directory. If the given path is outside the BSP directory structure, it is written as an absolute path. For example

```
$> cd hico7760/rootfs
```

means

```
$> cd $BSP/hico7760/rootfs
```

In some examples the BSP is given as directory `hico7760` in home directory of user 'hico'. For example:

```
/home/hico/emdist/hico7760/rootfs
```

1.2.2 IP addresses

For the sake of clarity, all the code examples in this manual are written with real ip addresses where:

Host ip address (aa.bb.cc.dd):	192.168.105.168
Target ip address (ee.ff.gg.hh):	192.168.105.56
Gateway ip address:	192.168.105.1
Subnet mask:	255.255.255.0

Always remember to replace these addresses to correspond your network setup.

1.3 **Starterkit Support**

The purpose of the 6 month support period is to help you to get started on embedded Linux development with the delivered tools and hardware.

Supported:

- Problems or general questions on getting started with the delivered Virtual Machine and development environment.
- Problems or general questions on the target board hardware interfaces and device drivers (USB, Ethernet, CAN, etc.).

Supported to some extent ⁽¹⁾:

- Getting started with the delivered BSP on another Linux distribution. The BSP is designed to be distribution independent. We cannot, however, guarantee that it will work out-of-the-box on any distribution or development host configuration.
- Customer specific configuration and modifications on the Linux kernel, build tools, user-space configuration (e.g. flash partitioning, boot method, etc), and open source packages (Busy box, Qt, etc.).

Supported only with extra support contract:

- Application development
- Issues on customer specific hardware
- Customer specific change/modification demanding notable effort.

⁽¹⁾ “Supported to some extent” means that we are willing to give tips and guidance, but are not bound to deliver any results.

2 emDist Linux distribution

2.1 Introduction

emDist is a Linux distribution developed at emtrion (**emtrion linux distribution**). The distribution provides a comfortable interface for building software packages and target images, *at the same time being very informative about what is actually being done*. It also provides a command line interface for developers who prefer to do the things on the console screen.

2.2 Development model

A common problem in embedded Linux development is, that in the development phase you need much more flexibility in the data exchange between target and host, than the actual product requires when it's ready.

For the development phase, the BSP provides a configuration for NFS boot, which means that the target board and the development host use and 'see' the same root filesystem directory (`uspc/rootfs`). You don't have to care about the size of the root filesystem because it is located on your host computer. No downloading or uploading with FTP or similar tools is required.

At some point when your application is ready and you are satisfied with the target system configuration, you want to build flashable filesystem images and boot the system entirely from flash (transition from development into *stand alone*). The starterkit includes tools and scripts for conveniently turning the "development root filesystem" into a lean flashable filesystem.

2.3 emDist web-interface and scripts

You could think of emDist web-interface as an *interactive command reference*. It shows you all the emDist scripts in a easily understandable fashion and lets you run the scripts directly without having to give any command line commands.

The screenshot shows the emDist web interface for the busybox package. The interface is divided into a left sidebar and a main content area. The sidebar contains a tree view of the emDist structure, with 'busybox' selected. The main content area displays the package details for 'busybox-1.7.3', including its description, address, source directory, build directory, and install directory. Below the details, there is a list of target scripts with 'Run' buttons and a green flag icon. The 'target.busybox.build' script is expanded, showing a terminal window with the following content:

```
# Check that the package is already prepared and
# configured...
Test-if-ok target.busybox.prepare \
            target.busybox.configure || Abort

# ...and change to the package directory
cd $EMDIST/hico7780/pkg/busybox-1.7.3
make ARCH=sh \
      CROSS_COMPILE=sh4-linux-
```

2.3.1 Many ways to run a script

All the scripts you can run from the web command reference, you can also run directly from command console. Here are three examples how to run the same script `emdlist.my_commands.telnet`, which will start a telnet session to the target board.

1. Run the script directly from command line

```
$> ./hico7760/scripts/emdlist.my_commands.telnet
```

2. ...or use the helper command `util/Run` to do this:

```
$> export PATH=$BSP/util:$PATH
$> Run emdlist.my_commands.telnet
```

3. ...or, use the most comfortable option, `emDist` web interface!

The screenshot shows the emDist web interface. At the top, there is a command input field containing `emdlist.my_commands.telnet` and a `Run` button. Below the input field, a description reads "start a telnet session to the target board". A terminal window is open, displaying the output of the telnet command: `telnet 192.168.105.52`, `Trying 192.168.105.52...`, `Connected to 192.168.105.52.`, `Escape character is '^'.`, and `192.168.105.52 login: root`. The terminal prompt is `~ #`. The terminal window title is `emdlist.my_commands.telnet`.

TIP: You can also give multiple script names as parameter to the `Run` helper command. Say, you are making some changes in the Busybox source code, you could build, install and start a serial console to the target with one command:

```
Run target.busybox.build \
    target.busybox.install \
    emdlist.my_commands.picocom
```

2.4 *BSP files and directories*

Before we go to the BSP installation let's take a look at some of the most important directories and files in emDist. The directories can be divided into static directories (i.e. the directories which are there when you checkout a fresh version of the emDist sources) and dynamically generated directories (generated when emDist is run for the first time)

Static directories

conf	Contains all the configuration files (package configuration, target board configuration, build commands etc.)
util	Diverse utilities needed for building flash images, establishing a serial connection etc. Most of the Host packages are installed here.
emdist	Source files for the emDist web-engine and configuration scripts. You shouldn't need to change anything here.

Dynamic directories

hico7760/pkgs	Package source directories for the target board
hico7760/rootfs	NFS mountable root filesystem for the target. All the package binaries are installed here. Flashable filesystem images are generated from this directory.
hico7760/rootfs/boot	This is a 'special' directory in rootfs, which is not included to the flashable images. The flash filesystem image and kernel images are installed here so that they are accessible from the target when using NFS mount.
hico7760/scripts	The emDist scripts are generated here.
Trash	This is a trash directory for the 'Trash' function in the emDist scripts.
downloads	The package tarballs, patches and everything which is downloaded from internet comes here.

3 emDistVM virtual machine

emDistVM is a VMware virtual machine with preinstalled Linux distribution (Xubuntu 6.06 “Dapper Drake”) with all required packages for the BSP and the BSP itself. You can start the VM on your Windows PC with the delivered VMware player. The purpose of the VM is to:

1. serve as a development host for people who don’t want to have an extra Linux PC next to their Windows PC. By using the delivered VM You can comfortably have both operating systems running on the same machine.
2. serve as a reference installation, for people who want to use their own Linux installation.
3. Give a quick start for anyone who doesn’t want to spend time on configuring the development environment.

Note: If you already have a Linux installation with which you are satisfied, you don’t have to use the virtual machine. You can install the BSP on your development host as described in chapter 4.

3.1 System requirements

In order to develop efficiently on the virtual machine it is recommended to have an development PC with ‘Pentium 4 class’ processor (or better) and at least 512MB memory. For development it is recommended to have 1GB of memory and set the VM RAM to 512MB (see next chapter).

The VM zip file is about 1.5G and after unzipping it requires 4G disk space.

3.2 *Performance issues*

3.2.1 RAM allocation

If you are going to use the Virtual machine as your Development environment it is recommended that you give it more than 512MB of RAM. With the default 256MB the VM is not well suited for running large graphical programs like IDEs. You can change the amount of RAM as described in Chapter 3.6.1.

3.2.2 Disk fragmentation

The mass storage device of the Virtual machine is seen as one file on the host (* .vmdk in the virtual machine directory). If this file is defragmented on your hard drive it can considerably slow down the virtual machine. In order to avoid this problem, defragment your hard drive on a regular basis.

3.3 *Install VMWare Player*

Insert the emDist VM DVD and open the start site `start_here.html`. From here you can copy the VMware-Player installation program onto your host and run it. (You can also download it from [VMware web site](#)). The installation should be straight forward.

<p>NOTE: In case you already have VMware workstation older than version 5.5, the delivered VM cannot be used with it. You need to upgrade your VMware software to 5.5 or later (or use VMware player).</p>

3.4 *Installing the emdist VM*

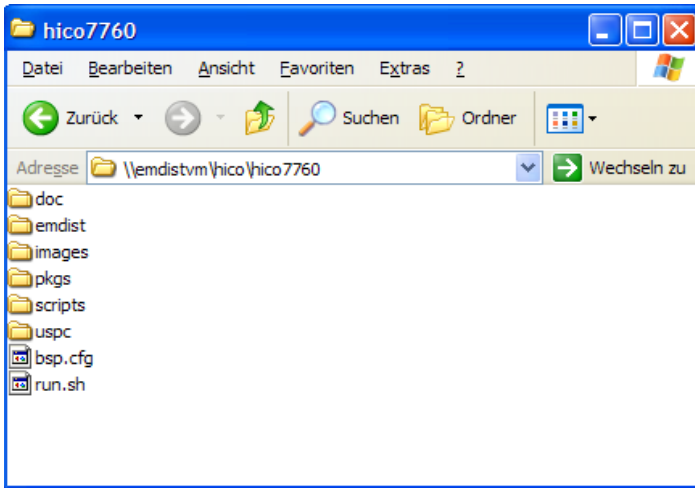
Unzip the emdistvm.zip file into a local working directory (For example “My Files\My Virtual Machines”). The location doesn’t matter as long as it’s not a network directory

Start VMwarePlayer and open the emDistVM configuration file from the location where you unpacked the zip file (VMware Player will ask you for the location when you start it). After opening the file with VMwarePlayer you should see the virtual machine booting. After logging in, you will get to the emDistVM workspace.

3.5 Sharing files with the Virtual Machine

3.5.1 Samba

The VM uses samba to share the home directory of user ‘hico’, where the BSP is installed. You can open and explore all the samba shares by simply giving address ‘\\emdistvm’ into the windows explorer (not internet explorer) address bar.



You are prompted for username and password. Both are ‘hico’.

TIP: If windows can't find the VM by its name (emdistvm) open a terminal in the VM (Applications->Accessories->Terminal) and check with command `ifconfig` the ip address of the VM and type it into the windows explorer address bar (i.e. '\\aa.bb.cc.dd')

3.5.2 NFS

emDistVM has an NFS server enabled in first place for the NFS root mount (i.e. the target board mounts the root file-system over network).

Windows doesn't have an NFS client included by default, there are, however, a number of commercial software for accessing NFS shares on windows (google for "NFS client for windows"). By default, only one

directory is exported and this is the target root file system in `uspc/rootfs`. If you want to add more exports, you'll need to edit the `/etc/exports` configuration file. See chapter 4.9 for more information.

3.6 Changing the default settings

3.6.1 Change RAM Allocation

With the default 256MB of RAM, the VM is not well suited for heavy development applications like IDEs.

In the VMware Player, you can change the amount of RAM for the virtual machine in menu “**Player -> Troubleshoot -> Change Memory Allocation**”. Just drag the slide to the desired value and reboot the virtual machine. Don’t just reboot the VM, but shut it down (System->Log Out->Shutdown), close VMwarePlayer and start it again.

3.6.2 Change Screen Resolution

By default, the screen resolution is 1024x768. If you want to work with the VM in full screen mode, you probably want to change the resolution to fit your monitor resolution (otherwise it just looks ugly). In order to change resolution, complete following steps:

1. Open a Terminal on the Virtual machine and run command `vmware-config-tools.pl` as root (or with `sudo`):

```
hico@emdistvm:~$ sudo vmware-config-tools.pl
```

2. The script will ask you if you want to change the screen resolution. Answer ‘yes’ and select the desired resolution. *To all the other questions the script asks; just give the default answer by pressing enter.*

```
Do you want to change your guest X resolution? (yes/no)
yes

Please choose one of the following display sizes (1 - 13):

[1]  "640x480"
[2]  "800x600"
.
.
[13] "2364x1773"
Please enter a number between 1 and 13:
```

4. After the script has finished, reboot the VM (System->Log Out->Restart the computer, or give command `sudo reboot`)

3.6.3 Change serial port

In order for the software in the VM to be able to use a serial port on your PC, check that you have the correct COM port in your VM configuration (*.vmx file in the VM directory). By default this is COM1.

```
serial0.fileName = "COM1"
```

3.7 Useful Tips

3.7.1 Copy and Paste

Clipboard function between the virtual machine and your Windows host requires the `vmware-toolbox` to be running on the VM. By default, `vmware-toolbox` is started at the VM boot time. If you happen to kill it you can start it again from a terminal window:

```
hico@emdistvm:~/ $ vmware-toolbox &
```

In order for the clipboard function to work, you need to start `vmware-toolbox` as a background process like above.

IMPORTANT: In some occasions, the `vmware-toolbox` seems to slow down the system notably, so it's better not to keep it running if you don't need it.

4 Getting Started

If you are using the delivered virtual machine, all the necessary packages, the BSP and toolchain are already installed and the NFS server is configured. It is highly recommended though, that you read everything carefully, to gain a better understanding of the system.

For taking Emdist into use on your own Linux installation, you should have a basic knowledge of Linux. During the installation and configuration you might have to install some software packages on your host (for example NFS server). Consult the documentation of your Linux distribution on this.

The BSP is designed to work with any modern Linux distribution. However, due to the great number of different Linux distributions out there, it cannot be assured that everything always will work "out of the box". Once you get the right packages installed from your Linux distribution, there shouldn't be any further problems though.

4.1.1 A word for Linux newbies

In following chapters you will probably spend some time configuring network communication between your development host and HiCO.SH7760. Things like setting up an NFS server is a very common task under Linux and your distribution most likely has documentation on how to do this. Most major distributions even have Wizards (for example SuSE) or a "share folders" configuration (Ubuntu) for this.

Remember, if you get stuck, always check the system messages for what might have gone wrong:

```
$> tail /var/log/messages  
or  
$> dmesg | tail
```

4.2 *Installing emDist for HiCO.SH7760*

You can skip this step if you are using the emDistVM virtual machine.

Installing the BSP means merely checking out the emDist sources from emtrions' subversion repository and setting the target link in the conf directory (you need to have a subversion client installed):

```
$> svn checkout \  
    --username=XX --password=XX \  
    svn://emtrion.de/emdist/branches/emdist-2.1 emdist  
$> cd emdist/conf  
$> ln -s hico7760.cfg target.cfg
```

After this you can start the emDist web engine and open the web interface with your browser

```
$> cd ../  
$> ./run  
emDist web-server running. Open address  
'http://localhost:8080' with your web browser
```

The next step is to run the command **emdist.packages.build_all** (you can start it from the web interface). This command will download, patch, configure, build and install all the needed packages for a fully functional embedded Linux system.

IMPORTANT: emDist scripts use **sudo** in many places to run commands with root privileges. *If you don't have sudo correctly configured on your system the build will fail.* See manual pages for sudo for more information. Also see `/etc/sudoers` file in the emDist virtual machine.

IMPORTANT: You will need a subversion account from emtrion, before you can check-out the emDist sources and some of the packages. Please contact us if you don't have one.

4.3 Start the emDist web interface

Script `run.sh` (emDist root directory) will generate the emDist scripts and start the emDist web interface.

```
~/emdist > ./run  
Open address 'http://localhost:8080' with your web browser
```

You can now open the emDist web interface by giving address <http://localhost:8080> in your web-browser.

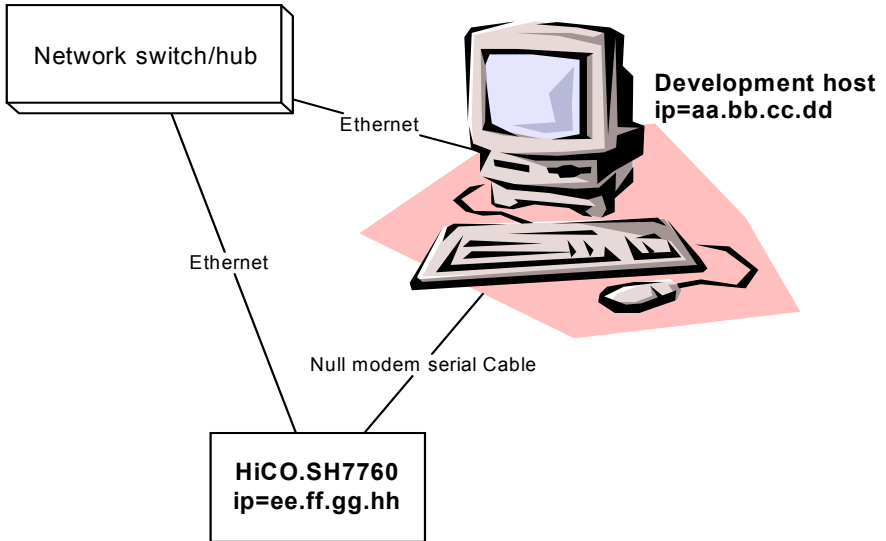
The script will might give you some warnings about issues that might cause some emDist commands not to work correctly. For example if you don't have sufficient rights for the serial port, you will get a warning and suggestion what to do:

```
~/emdist >  
~/emdist > export EMTRION_SVN_PASSWORD=XX  
~/emdist > export EMTRION_SVN_USERNAME=XX  
~/emdist > ./run  
Failed to open serial device '/dev/ttyS0': Permission  
denied. You can change the device node permissions with  
command:  
sudo chmod 666 /dev/ttyS0
```

NOTE: If you don't export the svn password and username, the run script will ask for them. Please contact us if you don't already have an account for emtrion's subversion server..

4.4 *Install HiCO.SH7760 hardware*

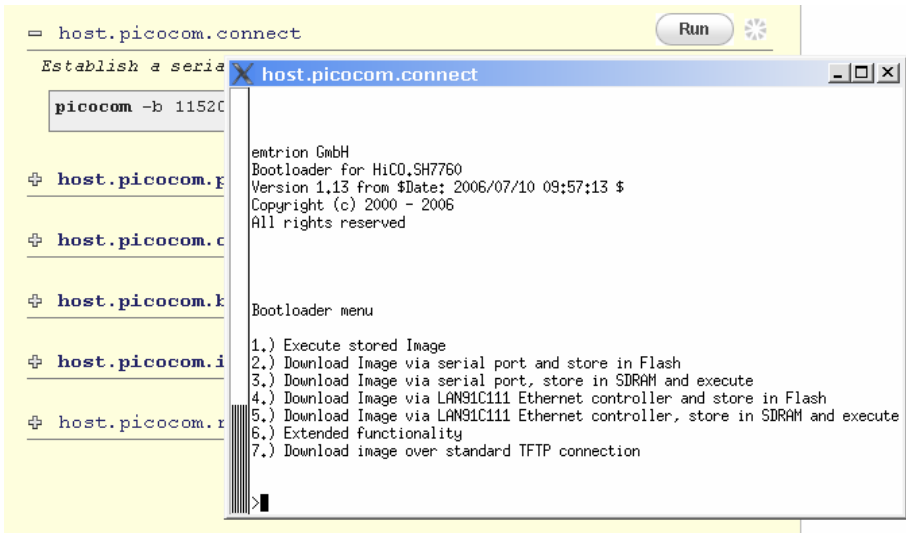
What you need is an Ethernet connection for HiCO.SH7760 to your local network and a free serial port on your development host. Install the hardware like illustrated in picture below.



4.5 Connect to the target via serial port

Run command `emdist.my_commands.picocom` from the emDist web interface. You should get the target boards bootloader menu (if not press the reset button on the target first)

NOTE: ensure that you have the bootloader jumper on the board set to the correct position (see chapter “[Boot mode jumper](#)”)



[reload cfg](#) | [Support](#) | Copyright © emtrion GmbH, All Rights Reserved

NOTE: Depending on your development host HW configuration you might have to use another serial port than `/dev/ttyS0` (defined in `conf/main.cfg`). You also might have to change the file access rights for `/var/lock` directory and `/dev/ttyS0` device file:

```
#> chmod 777 /var/lock
#> chmod +t /var/lock
#> chmod 666 /dev/ttyS0
```

4.5.1 Other serial terminal programs

On Windows you can also use HyperTerminal with following settings:

baud rate	115200
data bits	8
parity	none
stop bits	1
hardware flow control	no

4.6 Check that the network is working

In order to download images to the kernel via ethernet, port 980 needs free on your development host (it is usually blocked by Firewalls if you have one running). Proceed with following steps to check that the port is open and that the download can be done:

1. Open a terminal and run following command on your development host

```
$> sudo tcpdump port 980
```

2. Start a serial connection to the target board and in the bootloader Menu give command 4 „Download image via XX Ethernet controller...“
3. You should now see some messages on the tcpdump terminal window (messages with about 1 second interval):

```
112:50:48.705511 IP ced001c1e000031.980 > 255.255.255.255.980: UDP, length 64  
12:50:50.711419 IP ced001c1e000031.980 > 255.255.255.255.980: UDP, length 64  
.  
.
```

4.7 Set IP address

For development, you need a static IP address for the target board (ask for one from your IT administrator). It would be a good idea to get a static IP address for your development PC or for the emDistVM virtual machine as well. This way you can be sure to have a working TCP/IP connection between the target and host.

Write the IP address configuration file `conf/hico7760.cfg`.

```
.
[target]
ip=192.168.105.56
.
```

In some cases you could depend wholly on DHCP, but the practice has proven that using static IP addresses during development saves you from many troubles.

4.8 *Build target root file system*

On the virtual machine, there is already a working root filesystem for the target. If you installed the emDist on your own Linux distribution and ran the command `target.packages.build_all` you also already have a working root filesystem.

To build a new root filesystem from scratch, you need to remove the old one and install all the target packages again. This is easily done with the emDist scripts. First remove content from the old rootfs

```
$> sudo rm -rf hico7760/rootfs/*
```

Now you can run command **target.packages.install_all** from the emDist web interface.

In `emdist.my_commands` is an example 'my_build_command', which shows you how you can create a root filesystem image, download it and take a serial connection to the target with one command.

```
Run target.base_rootfs.install \  
      target.images.rootfs_cramfs \  
      target.images.download_all \  
      emdist.my_commands.picocom
```

You could of course run these scripts one by one from the emDist web interface, but after doing one procedure a couple of times it makes normally sense to combine them to be one command.

4.9 Configure NFS server

You can skip this step if you are using the emDistVM virtual machine. It is recommended to read it though.

NFS root mount allows the Linux kernel on HiCO.SH7760 to use `hico7760/rootfs` directly as its root filesystem. This is a major advantage in development phase since you don't need to copy files to the target using a program like `ftp`. Just copy any file into the `hico7760/rootfs` directory and the target board will "see it" immediately.

How to setup the server depends on the Linux distribution you are using and you should consult your distributions documentation on this. For example on SuSE you would run Yast (SuSE configuration tool), go to the "Network Services" and Click on the "NFS Server" icon. The configuration wizard asks for the exported directories and parameters.

The configuration should look something like following in your `/etc/exports` file:

```
/home/hico/emdist/hico7760/rootfs
*(rw,no_root_squash, sync)
```

You can make sure that your NFS server is working by mounting the exported directory locally. You should see the contents of `hico7760/rootfs` in the mount point:

```
$> cd ~
$> su
#> mkdir test
#> mount localhost:/home/hico/emdist/hico7760/rootfs test
#> ls test
.  ..  bin  dev  etc  lib  mnt  proc  sbin  sys  tmp  usr
var
#> umount test
```

NOTE: Don't forget to check the system messages if you get stuck:

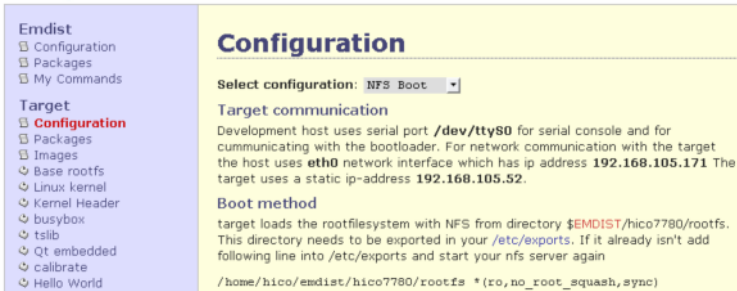
```
$> dmesg|tail
```

A normal cause for the root mount to fail is the firewall or other security settings on the development host.

4.10 Run Linux kernel with NFS root mount

In the delivered board, the kernel is set to get the root file-system from flash by default (i.e. it's a stand-alone system). Now we need to download a kernel which is configured for NFS boot and store it into the flash.

1. First check that you have the Configuration set to NFS foot:



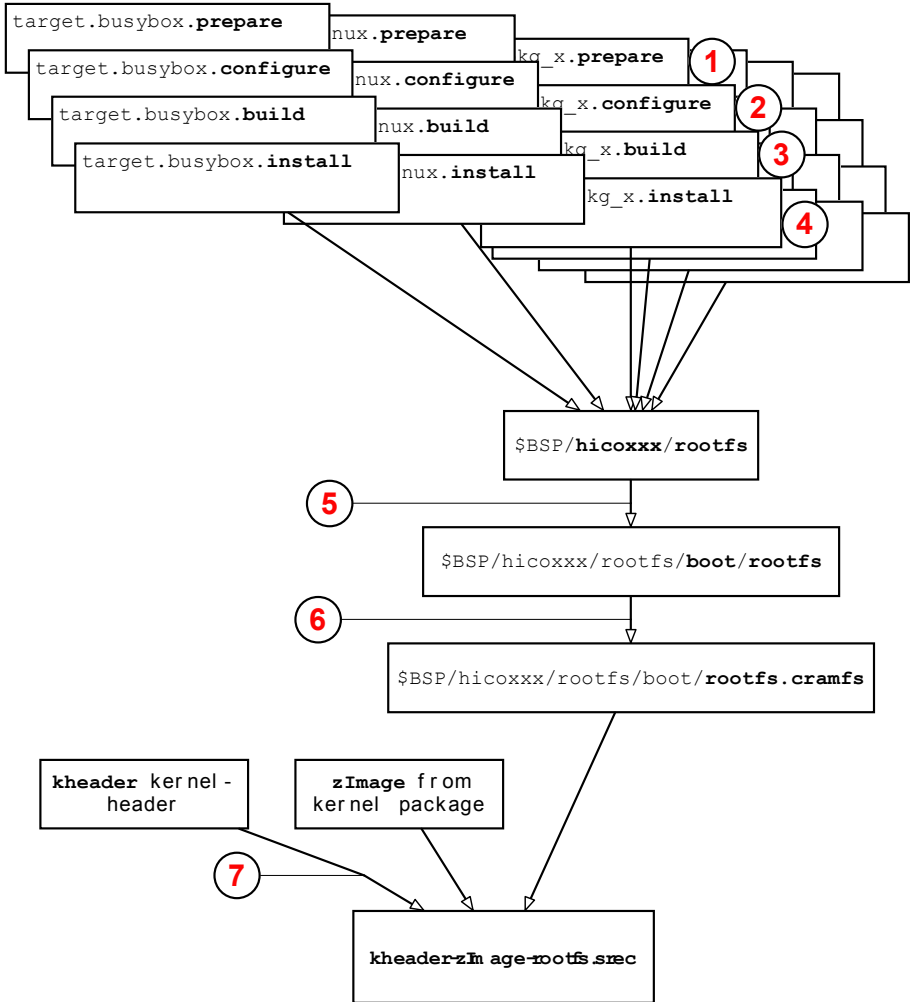
2. From the emDist web interface, run command **target.images.download_kernel**.
3. After the script is done. Take a serial connection to the target with **emdist.my_commands.picocom**.
4. Reset the board and in the bootloader menu give command 1. "Execute stored image"

You should now see the kernel boot messages after which you get the busybox command prompt. On the target serial console, you can try to create, for example, a directory (mkdir foo) and see if the directory appeared in hico7760/rootfs as well.

NOTE: Not all directories on the target can be seen from the host. For example the /var and /tmp are "temporary filesystem", which live in ram. You can see which filesystems are mounted with mount command (without any parameters)

5 System build – the big picture

The following picture describes the system build from the individual source package into the downloadable s-record file.



Every package has four “compulsory” commands; prepare, configure, build and install.

1. prepare – Downloads the package (normally a tar.gz file), unpacks it, applies any patches and does any other package specific preparation.

2. configure – This step normally includes running the packages ‘configure’ script with some target specific parameters (usually just the cross-compiler prefix and installation directory).

3. build – This command compiles/builds the package. Usually it just runs ‘make’

4. install – This command installs the resulting binaries into the targets root filesystem. If you browse the install scripts in the emDist web interface, you will see that this is often a single copy (cp) instruction which copies only the resulting binary. On many open source packages, if you run ‘make install’, you will get lots of files installed into the target filesystem, which are not really needed there (at least on an embedded system) – C-header files, manual/info pages and so on.

If you browse the package scripts, you’ll see that in many cases the scripts are empty – not every package needs a build command or configure command.

There are also other package specific commands for some packages. The linux kernel for example has a command for making a menuconfig etc.

7. In the `target.images.download_*` scripts the different flash area sections (i.e. the kernel header. The kernel itself and the root filesystem) are combined into one s-record image.

5.1 Where do my SW project files fit in?

You can add your own application as a package into the emDist system or you can manage the source code separated from the package management and just install your binaries into the targets root filesystem.

Adding new packages into emDist's package management is very easy. You just need to copy and paste one of the existing packages configuratio (for example `helloworld`) and edit the package parameters. Meaning of the parameters should be self explanatory – just use the existing packages as a reference. The effort is minimal.

6 Using the toolchain

Cross compiling a program with the cross-toolchain doesn't really differ from compiling programs for your host. Instead of invoking `gcc`, you invoke `sh4-linux-gcc`.

6.1 Toolchain location and the *PATH* variable

The toolchain package is not installed into your hosts binary libraries so you want to use the toolchain outside emDist you need to set it's location into your *PATH* variable.

You can see the correct path from the emDist interface (see menu **Configuration**):

PATH variable

emDist adds following paths to the *PATH* variable when running commands/scripts.

```
/home/hico/emdist/pkgs/toolchain-hicoarm9-2/bin  
/home/hico/emdist/util
```

If you prefer using command line, add at least the toolchain path above to your *PATH* variable.

TIP: If you want to set the *PATH* variable for good you can add following line in your `$HOME/.bash_profile` :

```
PATH=$PATH:/path/to/toolchain
```

6.2 *Cross-compiling simple applications*

The following example shows the Makefile of the “Hello world” program, which can be found in `hico7760/pkgs/helloworld`.

```
CFLAGS+=-g

hello: helloworld.c
    $(CC) $(CFLAGS) helloworld.c -o helloworld

clean:
    rm -f helloworld
```

In order to build and test the program:

```
$> cd hico7760/pkgs/helloworld
$> CC=sh4-linux-gcc make
sh4-linux-gcc -g hello.c -o hello
$> cp helloworld hico7760/rootfs/bin/
```

Change to the target serial console and run the program.

```
~ # helloworld
Hello World!
~ #
```

NOTE: You need to set the PATH variable to point to the toolchain directory. See chapter [Toolchain location](#)

6.3 ***Cross-compiling Qt applications***

When you're compiling Qt programs you have to have the QTDIR environment variable pointing to the right place. In this case it is the install directory of the Qt package `pkgs/bin/qt` (Needless to say that you have to have the Qt package installed – if `pkgs/bin/qt` is empty or doesn't exist, the package is not installed)

From the emDist web interface, select “**Packages -> Hello World (Qt)**”. Here are listed all the commands required for configuring, building and installing the example program.

For more information on qmake and Qt please refer to the Qt tutorials and documentation on Trolltech web site (<http://doc.trolltech.com/3.3>)

7 **Debugging your applications**

The de facto debugger in the Linux world is `gdb` and the numerous graphical debuggers out there are normally mere front-ends for it. One of the most popular graphical front-ends is `DDD`.

7.1 Remote debugging with plain GDB

Following example shows you how to start a remote debug session for the 'hello world' program `hico7760/pkgs/helloworld`.

Note that in following examples the development host ip is 192.168.105.168 and HiCO.SH7760 ip 192.168.105.56. You need to change these to match your ip addresses.

1. Start `gdbserver` on HiCO.SH7760 (`start-gdbserver` is a simple shell script which starts `gdbserver` with correct network parameters):

```
~ # start-gdbserver /bin/helloworld
Process /bin/helloworld created; pid = 322
Listening on port 4444
```

2. Open a text editor and write `.gdbinit` file for `helloworld` program - `hico7760/pkgs/helloworld/.gdbinit`:

```
set solib-absolute-prefix
/home/hico/emdist/hico7760/rootfs
file helloworld
target remote 192.168.105.56:4444
break main
continue
```

3. Start `gdb` in the same directory (export the toolchain directory first – see chapter [Toolchain location](#)):

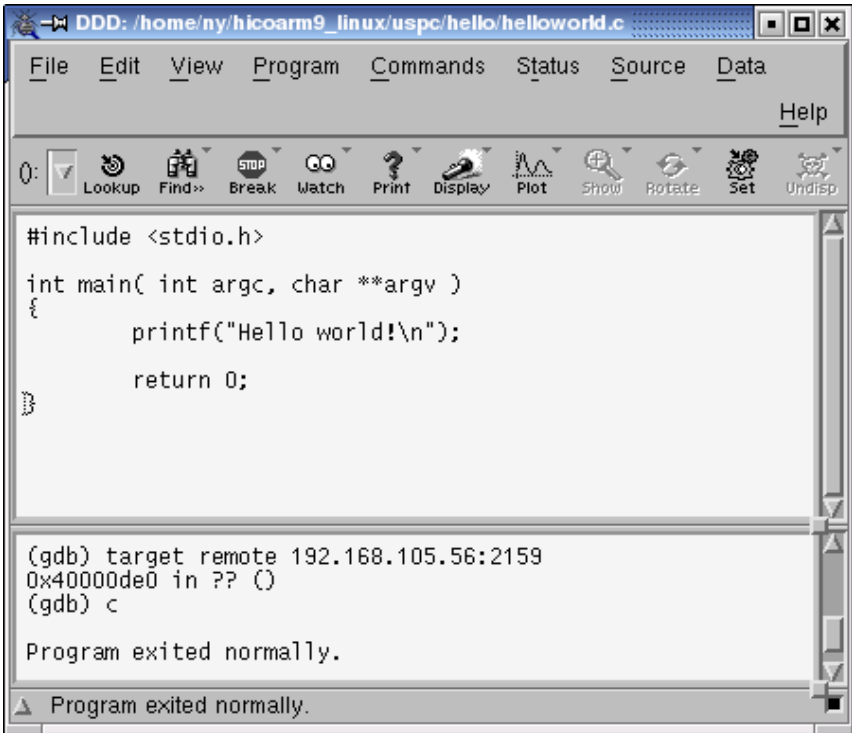
```
$> cd hico7760/pkgs/helloworld
$> sh4-linux-gdb
This GDB was configured as "--host=i686-pc-linux-gnu --
target=arm-softfloat-linux-gnu".
0x40000bd0 in ?? ()
Breakpoint 1 at 0x84f8: file helloworld.c, line 8.

Breakpoint 1, main (argc=1, argv=0xbe997db4) at
helloworld.c:8
8          printf("Hello world!\n");
(gdb)
```

For more information, see [GDB Website](#).

7.2 Remote debugging with DDD

DDD is a graphical front-end for `gdb` and in many cases more convenient to use than plain GDB. The underlying debug control still happens between `gdbserver` and `sh4-linux-gdb`.



Starting a `ddd` debug session doesn't differ much from starting a session with `gdb` (See chapter 7.1). You merely invoke `ddd` with the correct `gdb` command as parameter:

```
$> cd hico7760/pkgs/helloworld
$> ddd --debugger sh4-linux-gdb
```

You need the same `.gdbinit` file as with using plain `gdb` in last chapter.

After starting ddd you should now be able to set breakpoints and control the program execution via the graphical interface. The ddd screenshot shows how the session should look like.

For documentation and more information see DDD web site (<http://www.gnu.org/software/ddd>).

7.3 Start a debug session from the web-interface

The helloworld package has two examples how to start a debugging session comfortably (build, install and start a session). See commands

```
target.helloworld.debug_with_gdb  
target.helloworld.debug_with_ddd
```

In package helloworld. You can use the commands as templates to build your own debug session commands.

8 Software packages

The software packages are divided into two categories.

BSP base packages

This is the bare minimum you need on the target. If you leave any of the uninstalled the target filesystem image is not bootable. The base consists of following packages:

- Root filesystem skeleton and minimal set of target configuration files and scripts (`base_rootfs`)
- GNU C/C++ libraries (`base_libs`)
- Linux Kernel (`linux`)
- Busybox (`busybox`)

Since Busybox is a rather versatile package, for a headless system, this could be all you need.

extension packages

These packages are optional and you can include them into your target system depending on your needs.

Depending on your project requirements you probably need to include some other open source software as well (such as web-server, SSH server, etc.). In this chapter you learn how to add new software packages.

TIP: Always check if the required function or utility program is already included into `busybox`. For example, if all you need is a simple web-server you don't necessarily have to configure and cross-compile Apache for the task - you can use the one in Busybox.

8.1 **Building and installing packages**

In the following chapter, however, we discuss shortly about the most important issue when cross-compiling open source software packages. For the rest, the emDist web interface is far more better documentation on this issue than this manual can ever be.

8.2 **Cross compiling open source packages**

Normally when you build GNU-compatible software on your host, you just do the typical ‘configure’, ‘make’ and ‘make install’ procedure after which the compiled application is ready and installed on your system.

When compiling for a different architecture, however, you have take care of at least two things:

1. You have to tell the build system that you want to use a cross-compiler
2. You probably don't want to install the cross compiled binaries into your host computers system directories (which is the default place), so you have to give some other install target directory.

Configuring a GNU package for HiCO.SH7760 could look like something like this:

```
$> ./configure --host=sh4-linux  
--prefix=${HOME}/emdist/hico7760/rootfs
```

from the `--host=sh4-linux` option, the `configure` script knows to use `sh4-linux-gcc` cross-compiler. The `--prefix` option tells where to install the resulting files.

Ideally this is all you have to do. In reality, things are often not that simple, though. Depending on the complexity of the package, there are often other parameters to take into account as well. You can run the packages `configure` script with `--help` option to get a list of all parameters to tweak.

For software packages which are not compliant with the GNU build system (i.e. `configure` and `make`), you normally have to modify the packages Makefile directly. It is impossible to say a general rule for all the software out there, but the idea is the same - *you have to tell the build system that you*

are cross-compiling and that you want to install the resulting binaries into the `pkgs/bin/<package-name>` directory.

If you want to integrate the package into the emDist system, take a look at the `emdist/conf/target-packages.cfg` configuration file. You can easily add a new package entry with Copy&Paste method.

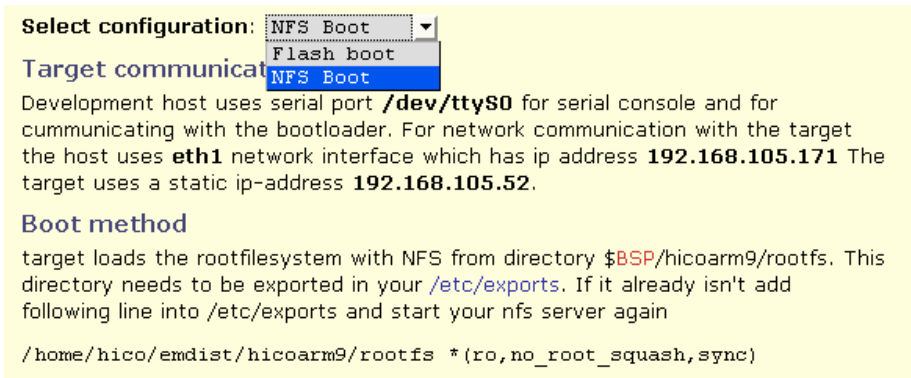
You can also contact emtrion GmbH if you want a specific package into the BSP.

9 Flash boot and NFS boot

Being able to load root filesystem automatically via network (i.e. NFS Boot) on every boot-up is a great help when you're developing software or debugging the system.

At some point, however, when you are somewhat satisfied with your root file system, you have to get everything into the flash memory so that HiCO.SH7760 can boot as a *stand-alone* system.

You can switch between flash boot and NFS boot from the emDist web interface



Select configuration: NFS Boot

Target communication: NFS Boot

Development host uses serial port `/dev/ttyS0` for serial console and for communicating with the bootloader. For network communication with the target the host uses `eth1` network interface which has ip address `192.168.105.171` The target uses a static ip-address `192.168.105.52`.

Boot method

target loads the rootfilesystem with NFS from directory `$BSP/hicoarm9/rootfs`. This directory needs to be exported in your `/etc/exports`. If it already isn't add following line into `/etc/exports` and start your nfs server again

```
/home/hico/emdist/hicoarm9/rootfs *(ro,no_root_squash,sync)
```

You can also add your own configurations to the menu (see `conf/configs.cfg`)

9.1 *A quick look at the used filesystems*

The Linux system on the target is a combination of multiple filesystem images depending on your needs. Here's a quick summary of the most important ones.

- `cramfs` Fast and reliable read only filesystem which can be located in RAM or linear flash.
- `jffs2` Journaling read/write flash filesystem. Use this if you want a writable and persistent flash file system.
- `tmpfs` "Temporary" filesystems which can be created at runtime and which live in RAM. This BSP uses `tmpfs` for example in `/dev` and `/tmp` directories. This is very useful in combination with read only root file systems like `cramfs`.

`jffs2` and `cramfs` are both compressed filesystems.

NOTE: When moving from NFS boot to flash boot, it often makes sense first to change the NFS root mount into a read only root mount. This way you can simulate the "stand alone" situation where the root file system is "read only". (at least most of it – depending on where you have mounted possible read/write `jffs2` flash file-systems and `tmpfs` RAM filesystems) . You can do this in `/etc/exports` (change the `rw` flag to `ro`)

9.2 *Flash partitions*

The flash area is partitioned in following blocks (see chapter *Hardware interface on HiCO.SH7760* for the real sizes and addresses):

bootloader - emtrion bootloader and the kernel header `kheader`

kernel - Linux kernel image (`zImage`).

rootfs - Read only root file system (`cramfs`). This is the place to put all the files and applications, which are not likely to be updated (at least not by product customers)

flashfs This partition is meant to serve as a read/write filesystem. See chapter 9.3, for how you can create a `jffs2` filesystem here

TIP: If you need a persistent writable root file system you can also use `jffs2` for `rootfs`. [DULG Documentation](#) includes some documentation how to do this. `uspc/make-flashing.sh` contains a “switch” for creating a `jffs2` root filesystem image. You might, however, consider think about the consequences first – a misbehaving script, program or user is able to destroy the whole root filesystem.

If you need to change the partitioning, you can edit mapping in the flash driver in file `linux/drivers/mtd/maps/hico7760-flash.c` or use the `mtdparts` kernel command line parameter. See file `linux/drivers/mtd/cmdlinepart.c` for description of the parameters.

TIP: On the target you can type command `cat /proc/mtd` to get information on the flash partitioning.

9.3 *Creating a JFFS2 flash partition*

JFFS2 flash filesystem images are, by default, not written to the target board as s-record files. The procedure is that you first create the image on the host, and use Linux on the target board to write it into flash.

The following example shows how the jffs2 filesystem (`flashfs` in `/dev/mtdblock3`) can be created and installed:

For an example with real values – see command `emdist.build_images.flashfs_jffs2` in the emDist web interface.

1. Create jffs2 filesystem image `flashfs.jffs2` (the image will be empty if the directory `flashfs` is empty).

```
$> mkdir -p flashfs
$> /sbin/mkfs.jffs2 -little-endian \
                    --pad=<image-size> \
                    --eraseblock=<block-size> \
                    --root=./flashfs \
                    --output flashfs.jffs2
$>
```

2. Copy the created image into `uspc/rootfs` (it is here assumed that the target board has mounted `uspc/rootfs` as its root with NFS boot)
3. On the target, write the file system image with `dd` to a free flash partition (this may take some time):

```
flash_eraseall /dev/mtd3
cat flashfs.jffs2 > /dev/mtd3
```

10 Working with the bootloader

From the bootloader options only **1** and **4** are of interest with this BSP. What these options do should be self explanatory.

```
emtrion GmbH
Bootloader for HiCO.SH7760
Version 1.11 from $Date: 2005/11/15 08:59:44Z $
Copyright (c) 2000 - 2005
All rights reserved

Bootloader menu

1. Execute stored Image
2. Download Image via serial port and store in Flash
3. Download Image via serial port, store in SDRAM and
execute
4. Download Image via LAN91C111 Ethernet controller and
store in Flash
5. Download Image via LAN91C111 Ethernet controller, store
in SDRAM and execute
6. Extended functionality
```

For the other options and bootloader features, please refer to the delivered bootloader documentation.

NOTE: The download helper script `dnldtool` communicates automatically with the bootloader via serial port.

10.1 *Board identifier*

Every emtrion board has an identifier which is used when downloading images via Ethernet. When you start to download an image (**option 4** “**Download Image via LAN91C111 Ethernet controller and store in Flash**”), the bootloader will write the board id (or ‘device name’) on the serial console:

```
.  
.br/>>Using Ethernet mode : 100BASE-TX, Full Duplex  
>Ethernet Address: 00:30:6C:50:01:3F  
>Using device name: HiCO7760_251  
>Wait for DHCP or enter new IP address:  
>DHCP IP Address Resolved as 192.168.105.113, netmask:  
255.255.255.0  
>Lease time: 14400 seconds  
>Sent BOOTME to 255.255.255.255
```

NOTE: The download helper script `dnldtool` parses the ID automatically from the bootloader output.

10.2 *Boot mode jumper*

The pin jumper (W30) on the side of the core module sets the bootloader either in

- direct boot mode – no bootloader menu is printed on the serial port and the saved OS image is started immediately.
- Boot menu mode – bootloader menu is printed on the serial console.

NOTE: on **HiCO.SH7780** the bootmode is not set with a jumper but with the DIP switch SW1 (switch 4).

11 Networking

11.1 Target network configuration

By default, the Linux kernel uses the network parameters given by the bootloader on kernel command line (`ip=...`). You can see the command line parameters from `/proc/cmdline`:

```
~ # cat /proc/cmdline
mem=64M console=ttyS1,115200 root=/dev/nfs rw \
nfsroot=/home/hico/hico7760/uspc/rootfs \
ip=192.168.105.56:192.168.105.168:192.168.105.1:255.255.25
5.0:hico7760_XXX::off)
```

If you don't want that network is configured during the kernel boot, you can disable it by giving 'ip=off' on the kernel command line (see `configs.cfg` and `target.cfg`) and write the network initialisation into the system start-up script `/etc/init.d/rcS`

NOTE: If you use NFS root mount, the network always has to be configured at kernel boot time.

11.2 **Telnet server (telnetd)**

Telnet daemon is started in the system start-up script `rootfs/etc/init.d/rcS`. In order to take a telnet connection to HiCO.SH7760, use following command on your development host (replace the ip-addresses to match yours):

```
telnet 192.168.105.56
```

Username is 'root' and there's no password.

12 System start-up

12.1 Device nodes on the target

Some device nodes are created statically in the `target.base_rootfs.install` command and the rest is created dynamically by the `mdev` utility in `busybox` (see `rcS` script in `rootfs.cfg`)

12.2 Start-up scripts

The first application to run is `busybox`. After initializing itself, it will execute the `etc/init.d/rcS` shell script. `rcS` makes the low level initialisation of the system. It mounts the necessary filesystems, sets the system clock and starts some services like the system logger and telnet daemon.

Before exiting the script runs all the executables in the `/etc/init.d` directory which start with 'S' following a number (for example `S20-my-start-script.sh`). The numbers in the script name indicate the execution order.

12.3 Where to start my application?

There are multiple ways to start your application at startup:

1. You can add a line in the `/etc/init.d/rcS` start-up file (i.e. `edit rootfs.cfg`)
2. add a line in the `/etc/inittab` file (`edit rootfs.cfg`)
3. Add your own `/etc/init.d/S[0-9]*` startup file. **Note:** be sure to add `#!/bin/sh` on the first line and mark the file as executable (i.e. `chmod +x <file>`).

From the given methods, the last one is preferred.

13 Advanced usage of emDist scripts

TIP: In order to get a more flexible access to the underlying data structure, you might want to set the `show_extra_cfg` parameter in `emdist.cfg` to `True`.

13.1 Understanding the structure

emDist scripts are shell scripts automatically generated from the definitions in the `conf/*.cfg` files. Adding new commands is very easy. You only write the most important shell commands – the rest (error checking, output formatting etc.) is generated automatically.

For example, by writing the following snippet into `emdist.my_commands.cfg` would create a command which establishes a serial port connection to the target (`picocom` is a small serial console program similar to `minicom`):

```
[emdist.my_commands.picocom.script]
% = picocom @(host.serialdev)
```

After adding these lines into the configuration file the emDist shell script is generated and added to the web interface.

Here is another dummy example which demonstrates the script configuration syntax.

```
[emdist.my_commands.do_something.script]
% = echo "doing something"
% = ls $HOME
% = cd $HOME/my_project
% = make
% = cp my_prog @(target.rootfs)/bin/
% = Run emdist.my_commands.telnet
```

13.2 Macros

You probably already figured out that the `@(...)` parts in the scripts are macros or placeholder for something. If you study the structure of the configuration data, you will realize that everything is a big tree of data nodes, and all the nodes can be used in the macros.

A macro will include the content of the specified node in the node tree. You can browse the available macros in the emDist web interface in addresses

- <http://localhost:8080/node/target:cfg>
- <http://localhost:8080/node/host:cfg>

14 Hardware interfaces on HiCO.SH7760

NOTE: Features described in this Manual are already enabled in the default configuration for HiCO.SH7760. Normally you only need to change the kernel configuration if you want to remove some drivers.

14.1 Supported Hardware

The BSP has support for following HW interfaces on HiCO.SH7760

- Onboard RAM 64MB
- Onboard Flash 32MB
- Ethernet 10/100 Mbit/s
- 1/4 VGA TFT graphics (full VGA optional)
- USB Host interface
- 3 Serial ports (scif0 - scif2)
- Touchscreen interface
- RTC
- I2C bus
- CAN Bus (HCAN2)

14.2 Flash partitioning

HiCO.SH7760 has 32 MB flash memory (0xA0000000-0xA3FFFFFF), which is partitioned as following table shows:

location	size	name	device (dev/*)	image filename
A0000000	0x80000 (512k)	bootloader+stub	mtdblock 0	stub
A0080000	0x200000 (2M)	kernel	mtdblock 1	zImage
A0280000	0xa00000 (10M)	rootfs	mtdblock 2	_rootfs.c ramfs
A0c80000	0x1380000 19.5M	flashfs	mtdblock 3	flashfs.j ffs2

14.3 Serial ports

HiCO.SH7760 provides three serial ports (SCIF0, SCIF1 and SCIF2). Following table shows the device node mappings:

```
/dev/ttySC0      SCIF0
/dev/ttySC1      SCIF1
/dev/ttySC2      SCIF2
```

ttySC2 is the default console (i.e. the 9-pin D-Sub connector)

14.4 USB

For USB mass storage devices (i.e. usb sticks etc.), the important kernel options are:

- Device Drivers / USB support / Support for host-side USB / USB Mass Storage support
- Device Drivers / SCSI device support / SCSI disk support
- Device Drivers / SCSI device support / SCSI generic support
- File systems / DOS/FAT/NT Filesystems / VFAT
- File systems / Base native language support / Codepage 437

Depending on what filesystem you are using on the storage device, you might have to configure the filesystem driver as well. By default USB mass storage devices are auto-mounted to `/mnt/<device_name>` (see chapter “udev and Hotplugging” for more information).

14.5 Touchscreen

The kernel support for touchscreen on HiCO.SH7760 can be built as a module or compiled directly to the kernel. You can insert the module with `modprobe` command:

```
modprobe hico_ts_input
```

14.5.1 Calibrating touchscreen

Calibration plugins for tslib are pre-configured in `/etc/ts.conf`. Calibration data is recorded in `/etc/pointercal`.

You can start calibration with command:

```
calibrate -qws
```

If you have problems with the calibration, try to remove the old `/etc/pointercal` before calibrating.

14.6 *Real Time Clock*

You can access the RTC clock with the `hwclock` utility. Here are some examples:

Read time from RTC and set Linux system date/time with it:

```
hwclock -s
```

Set Linux system time to a time fetched from internet using `ntp` and save this time to the RTC:

```
rdate tick.greyware.com  
hwclock -w
```

14.7 *I2C Bus*

Program `pkgs/src/dac` gives you an example how to use the `i2c` driver. It adjusts the background light of the LCD display.

14.8 CAN-bus (HCAN2)

The can driver (`hcan2.ko`) is delivered as a separate kernel module (i.e. it is not in the bsp kernel source tree). The driver sources contain an example program (`example.c`) and a well documented API header file (`hcan2_api.h`), which you can use as a starting point for your own applications.

NOTE: Just to get a idea of the drivers architecture, Please read the introduction in HCAN2 documentation in the SH7760 datasheet.

14.8.1 Driver description

HCAN2 interface has two CAN nodes, which both have 32 mailboxes. You can open (`open()`) the mailboxes as if they were independent devices, with exception tha some `ioctl` calls affect all of the mailboxes (like resetting the node or setting the baud rate). Opening a mailbox for multiple readers/writers is not allowed.

CAN telegrams are sent/received by opening one or more of the mailbox device nodes in `/dev/can[0-1]/[0-31]`, and by writing and reading data with `read()/write()` system calls.

- A mailbox can be configured as Receiving or Transmitting, not both.
- Every mailbox has its own device node in `/dev/can[0:1]/[0:31]`.
- Every mailbox has its own receive/transmit buffer.
- mailbox 31 has the highest priority and mailbox 0 lowest.
- An incoming CAN frame is written only to the first receiving mailbox where it passes the acceptance, starting from mail box 31.
- Mailbox 0 is read only.
- A mailbox can receive extended `_or_` standard can messages, not both. Same goes for normal messages and remote transmission request (RTR) messages.

TIP: If you don't need anything complicated, just open two mailboxes - one for reading and one for writing - and set the acceptance to receive all messages (i.e. use `IOC_SET_LAFM` with argument `0xffffffff`).

Following code snippet shows an example, how to open and configure a CAN node and send one message:

```
int Tx,Rx,ret;
struct can_msg msg;

/* Open a node */
Tx=open("/dev/can0/1",O_RDWR);

/* Reset the node. Which mailbox node is used for
this doesn't matter */
ioctl(Tx,IOC_RESET_NODE);

/* Set bitrate. */
ioctl(Tx,IOC_SET_BITRATE,BITRATE_1000k);

/* configure mailbox for writing */
ioctl(Tx,IOC_SET_MBOX_MODE,MBOX_MODE_WRITE);

/* compose a can telegram */
msg.stdid=0xab;
msg.extid=0;      // this value doesn't matter if ide=0
msg.rtr=0;       // normal messages
msg.ide=0;       // standard frames
msg.data[0] =0xab;
msg.dlc=1;

ret=write(Tx,&msg,sizeof(struct can_msg));
assert(ret==sizeof(struct can_msg));
```