

Starterkit Manual

HiCO.SH7760 QNX 6.3 Starterkit

emtrion

© Copyright 2006 **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 25/06/2008 14:13

- 1 Introduction..... 6**

- 2 Starterkit Contents 7**
 - 2.1 Starterkit CD..... 7
 - 2.2 QNX Momentics 6.3 evaluation CD 8
 - 2.3 Supported Hardware 9

- 3 System Overview 10**
 - 3.1 System Requirements 10
 - 3.2 BSP files and directories..... 11
 - 3.3 System boot..... 13
 - 3.4 Flash Images 14

- 4 Getting Started..... 16**
 - 4.1 A word for Windows development host users 17
 - 4.2 Installing HiCO.SH7760 hardware 18
 - 4.3 Install the BSP on the host 22
 - 4.4 Install and setup the kernel download tool 23
 - 4.5 Create a bootable flash image 24
 - 4.6 Connect the target to the host (serial) 25
 - 4.7 Connect the target to network 26
 - 4.8 Apply power to the target..... 27
 - 4.9 Transfer the OS image to the target..... 28
 - 4.10 Test Neutrino on your target..... 29

- 5 Building Flash Images..... 30**
 - 5.1 Example build files (IFS)..... 31
 - 5.2 Building your own IFS images 32
 - 5.3 Including Embedded Filesystem Images (EFS)..... 33
 - 5.4 Creating empty flash filesystem partitions 34

- 6 Getting started with SW development..... 36**
 - 6.1 Cross-debugging with Momentics IDE 37
 - 6.2 Cross-debugging with GDB..... 38

- 7 BSP Drivers 39**

- 7.1 A/D and Touch Driver (adcmgr) 40**
 - 7.1.1 Introduction 40
 - 7.1.2 Touch interface..... 41
 - 7.1.3 adcmgr Command line parameters 41
 - 7.1.4 Reading AD Values..... 43
 - 7.1.5 Reading touch coordinates 43
- 7.2 CAN Driver (hcan2mgr)..... 44**
 - 7.2.1 Introduction 44
 - 7.2.2 CAN channels 44
 - 7.2.3 Mailboxes 45
 - 7.2.4 Configuration file 47
 - 7.2.5 hcan2mgr Command line parameters 50
 - 7.2.6 Driver API..... 51
 - 7.2.7 Error Handling 56
 - 7.2.8 Controller reset..... 56
- 7.3 Photon input driver (devi-em)..... 57**
- 7.4 I2C Driver (i2c-camelot)..... 58**
- 7.5 RTC8564 Utility (rtc8564)..... 58**

- 8 BSP Utilities 59**
 - 8.1 kerneldnld - Download Tool..... 59**
 - 8.1.1 Description 59
 - 8.1.2 Command line options 59
 - 8.1.3 Examples 59

- 9 PC104 (ISA) bus 60**
 - 9.1 ISA-IO/MEM Mapping 60**
 - 9.2 ISA Interrupts 60**

- 10 HOWTOs 61**
 - 10.1 Establishing a Telnet connection to the target..... 61**
 - 10.2 Finding out library dependencies 62**
 - 10.3 mounting USB Mass storage 63**
 - 10.4 Mapping network shares 64**

- 11 Getting support..... 66**

- 12 HiCO.SH7760 interrupts 67**
 - 12.1 SH7760 external interrupts 68**
 - 12.2 SH7760 internal interrupts 69**

12.3 External interrupt mask registers 72

1 Introduction

This guide will help you get started with the QNX Neutrino RTOS on your HiCO.SH7760 platform. The Starterkit includes ready built file-system images (photon, networking, etc.), device drivers to use the HiCO.SH7760 peripheral and documentation to get started.

It is suggested that you first go through the chapters up to the *Chapter 4 “Getting Started”* and then carry out the steps in this chapter.

- Before you do anything else you should verify that the hardware works properly and the preinstalled OS image boots up.
- In order for the BSP to function properly **you have to have QNX 6.3 Service Pack 2 (SP2) Installed**. SP2 is delivered with this starterkit on a separate CD.

For more information about the hardware, please refer to the hardware documentation in [StarterkitCD]/hardware directory.

2 Starterkit Contents

2.1 Starterkit CD

Following list describes the top level directories on the starterkit CD.

bsp

QNX Neutrino 6.3 BSP for HiCO.SH7760. The BSP is zipped/tarred in one file which you can upack to a working directory on your development host. (see *section 3.2 “BSP files and directories”* for the contents)

doc

Contains the BSP/Starterkit documentation.

hardware

Contains hardware documentation and, depending on which version of the Starterkit you purchased, schematics for HiCO.SH7760 the hardware.

patches

Contains QNX Momentics 6.3 patches (if any) which are required for this BSP.

tools

Contains the GUI- and the command line kernel download tool.

bootloader

Actual bootloader image, documentation and source codes (sources only with the Starterkit Pro)

2.2 QNX Momentics 6.3 evaluation version

Please download the latest Momentics evaluation version from address

- ❖ <http://www.qnx.com/products/getmomentics/>

2.3 Supported Hardware

Supported hardware components by this BSP:

- Board startup (IPL, Startup code)
- Serial communications with SH7760 internal serial interfaces scif0, scif1 and scif2 (devc-sersci)
- 100/10 Mbit TCP/IP Networking with LAN91C111 (devn-smc9000)
- CAN networking with SH7760 internal CAN interface (hcan2mgr)
- USB Host interface with SH7760 internal USB host controller (devu-ohcibiscayne.so)
- Graphics with Photon and SH7760 internal LCD controller (devgsh7760lcdc.so)
- 4-Wire Touch driver for the LCD display (devi-em and adcmgr)
- Flash file system - Flash TDK required (devf-generic)
- AD-Converter interface (adcmgr)
- I2C Bus (i2c-camelot)
- Real Time Clock (rtc8564)
- PC104 (ISA) bus access and interrupts

3 System Overview

3.1 System Requirements

In order to start to develop software using the HiCO.SH7760 with QNX Neutrino 6.3 you need to:

- install QNX Momentics 6.3 on your development host. The host operating system can be any of the QNX Momentics 6.3 supported host systems.
- install all the QNX patches in the [starterkitCD]/patches directory.
- install QNX Neutrino 6.3 BSP for HiCO.SH7760 [starterkitCD]/bsp on your development host

You also need:

- a free serial port on your development host (cable delivered in the Starterkit).
- a free Ethernet (RJ-45) connection to your local network (cable delivered in the Starterkit).

3.2 BSP files and directories

The HiCO.SH7760 QNX Neutrino 6.3 BSP follows the QNX BSP convention in the directory and Makefile structure with some exceptions. Here's a small description of the most important files and directories in the bsp:

Makefile

Top level makefile for the BSP. It copies the prebuilt to install, builds all the sources in the src and installs the resulting binaries to install. It then runs the images/Makefile which builds the flash image. You only have to run this once if you're not doing changes in src.

images/hico7760.*.build

Example build files.

images/hico7760.*.sre.ref

Prebuilt downloadable s-record files. Note that these images include the emptyefs partition (i.e. /root on the target is a read/write flash filesystem).

images/last_build.sre

Symbolic link to the last created s-record file by the mkflashimage). (TIP: You can use this as an input file for the kernel download tool)

images/emptyefs.bld

Build file for the mkefs utility - creates an empty flash file system which mounts to /root.

images/mkflashimage

Shell script to build flash images (see contents for more information).

doc

contains the BSP documentation.

install

the build system uses this as a "scratch" directory. It is used as the primary search path when images are built.

prebuilt/shle

Prebuilt BSP binaries.

prebuilt/usr/photon

Prebuilt Photon configuration files.

src

BSP source codes. The top level Makefile first builds all the sources located here and then installs them to the install directory.

util/kerneldnld

Command line kernel download utility (The GUI kernel download tool is in theTools directory on the starterkit CD).

3.3 System boot

From a software perspective, the following steps occur when the target system starts:

1. The processor begins executing at the reset vector, where the HiCO.SH7760 bootloader is located
2. The HiCO.SH7760 bootloader initializes the most important the hardware (processor registers, RAM, etc). After this it copies the image that was last downloaded wholly to the RAM and jumps to its first address (which is the start address of the IPL)
3. For now, the IPL has nothing to do since the OS image already resides in RAM the HiCO.SH7760 bootloader already initialised required hardware. The IPL simply makes a jump to the start-up code section of the OS image.

The startup code fills the QNX Neutrino 6.3 system page with information about HiCO.SH7760 memory layout, interrupts, etc. It also initialises some peripheral hardware. When it's done it jumps to execute the Neutrino microkernel and the process manager (`procnto`). This is the `.bootstrap` section in the `mkifs` build file.

4. the `.script` section of the build file is executed. This normally includes:
 - serial port driver `devc-*` is started and possibly some environment variables are set
 - The flash driver `devf-*` is started. The driver automatically detects the file system images which begin at the flash block boundaries
 - Network resource manager is started and network is configured
 - Shell is invoked on the console (i.e. the serial port)

3.4 Flash Images

The `images/mkflashimage` shell script produces Motorola S-record images which can be downloaded to HiCO.SH7760 by using the `kernelndld` command line utility or the GUI based `emtrion` kernel download tool. The OS/file-system image consists of multiple sections which differ in contents and run-time behaviour:

ipl

Initial Program Loader. This section is included to every flashable image. Usually its purpose is to initialise hardware, but on the HiCO.SH7760 it's almost empty because the HiCO.SH7760 boot-loader does all the initialisation

ifs

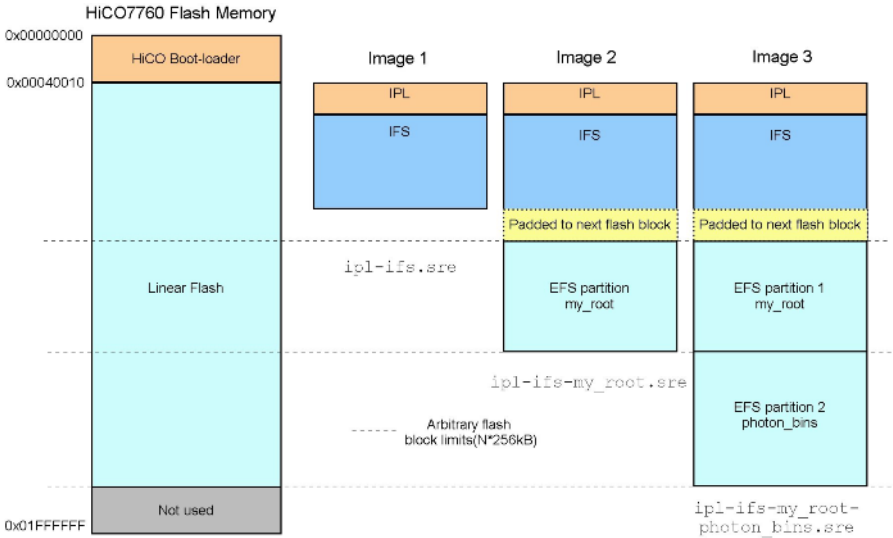
This is the actual OS image made with the `mkifs` utility. It contains the Neutrino Microkernel and all the other components included to the `ifs` build file. The whole `ifs` section is loaded into RAM during the bootup and it doesn't contain any writable file-system. Consider keeping it small.

efs images

EFS sections are optional embedded filesystem images generated with the `mkefs` utility. These sections are padded by the `mkflashimage` to flash block offsets so that the flash file system driver `devf-generic` finds them without any command line parameters (See chapter 5.3, *"Including Embedded Filesystem Images (EFS)"*).

Figure below represents three cases (images 1 -3) how the different s-record files are loaded into the flash.

From the Figure you can see that if you make minor changes to the IFS part, you would only have to download image 1 again even though you are using all of the partitions (given of course that the flash block limits remain the same).



Note

At board boot time, the emtrion bootloader copies the last downloaded image wholly into the RAM before starting it. This is, however, necessary only for the ipl-ifs part. In order to reduce the boot time, download always the ipl-ifs last (e.g. If you need to replace the whole image, download first the ipl-ifs-mypart1-my part2.sre and after that the ipl-ifs.sre again).

4 Getting Started

In this chapter we setup the development environment, build one of the example builds and download it to the target. Before you start you should:

5. build the hardware as shown on Figure 1, “HiCO.SH7760 interfaces”
6. Apply power to the target verify that the preinstalled image boots up (you can see the calculator program on the LCD display)

Important

Be sure to check the install/release notes on the Starterkit CD before you start.

4.1 A word for Windows development host users

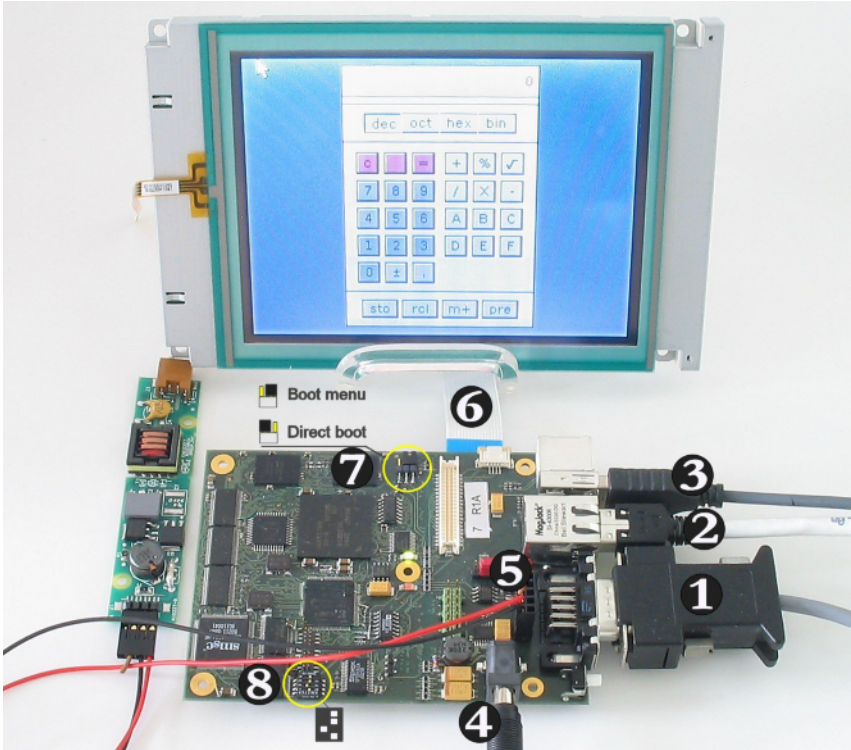
compiling and building the bsp is done by typing commands on command line (which is the usual way to do things on a Unix like platform). On Windows this is the command prompt or "DOS shell" (Start this from "Start Menu" -> "Run" and write cmd). After you have learned how the build system works you can write your own batch file (*.bat) or a shell script to automate the build process.

Some of the host side command examples given in this guide are done under unix shell (Bash/KSH). The same commands should work also under the Windows command prompt. If you want to use a UNIX shell, you can start it by typing "bash" or "ksh" under the command prompt.

It is recommended that you do some experimenting with the Korn Shell. This is the same shell as on the target. Sometimes writing a shell script instead of writing a C or C++ program can save you a great deal of time.

4.2 Installing HiCO.SH7760 hardware

Figure below shows you the most important interfaces and components for this BSP.



Important: Above figure is out-dated.

- On this version of the Starterkit, the default input device for Photon is the touch interface while on the picture it is USB-mouse/keyboard. Please attach the touch to hico7760 instead (If you wish, you can change the input driver back to USB in the hico7760.photon.build script).
- Backlight inverter has it's own power +5v connector (next to the two jumpers) (see HiCO.SH7760 hardware manual).

- 1.** RS232 serial interface. The bootloader menu is "printed" to this port. Use a NULLModem cable to connect it to a free COM interface of your host computer.
- 2.** Ethernet interface. You will need this when you download OS images. Connect it to your local network.
- 3.** USB host interface. Connect an USB mouse for the Photon demo application.
- 4.** Power supply.
- 5.** Power supply for the LCD backlight inverter.
- 6.** LCD interface.
- 7.** jumper to select the bootup behaviour (direct boot / print boot menu on serial port)
- 8.** LCD Display selector (Digital input used by the bootloader). For the delivered LCD display set the dip switches as shown on the picture. See next chapter for more information.

Note

For other interfaces (CAN, USB function, Touch, etc.) see HiCO.SH7760 hardware manual.

4.2.1.1 HiCO.SH7760 dip switches

The bootloader checks the state of the four dip switches (S30) on HiCO.SH7760 core module and sets the display driver according to the state. Following settings are used:

1	2	3	4	SW value	purpose
off	off	x	x	3	Reserved
on	off	x	x	2	TFT 800x600 ^(a)
off	on	x	x	1	1/4 VGA Display ^(b)
on	on	x	x	0	VGA Display

a) Not supported by this BSP

b) default starterkit setting

4.3 Install the BSP on the host

It is assumed that you have already installed the QNX Momentics 6.3 on your system. If you haven't, please refer to the documentation delivered with the QNX Momentics 6.3. Install all the parts which are for SH4 Processor platform (shle).

The BSP is delivered as a tar.gz (QNX, Linux, Solaris) or zip (Win) file. Unpack the package to a working directory (For example c:\bsp). After this you will find hico7760 directory which is the root directory of the BSP.

Note

Do not unpack the BSP to a path which contains empty characters (for example "c:\My Documents\bsp" or "c:\Program Files\bsp").

4.4 Install and setup the kernel download tool

There are two versions of the emtrion kernel download tool - a GUI based windows application [StarterkitCD]/tools/KernelDlnd_GUI and a simple command line tool [StarterkitCD]/tools/kerneldnld.

The GUI tool runs only on Windows so if your development host is a POSIX OS, the choice would be the kerneldnld command- line tool.

4.5 Create a bootable flash image

In order to try the installation, open a command console and run the top level Makefile:

```
cd path-to-bsp/hico7760
make
```

This will populate the install directory and make the flash image (see also *chapter 3.2 “BSP files and directories”*). By default mkflashimage builds the hico7760-photon s-record image.

If the build was succesfull you will find ipl-ifs-emptyefs.sre file in the images directory (see *chapter 3.4 “Flash Images”*). This file should be nearly the same (not identical) as the delivered hico7760.photon.sre.ref reference image in the same directory.

4.6 *Connect the target to the host (serial)*

1. Connect one end of a NULL-modem cable to the 9-pin serial port on the target.
2. Connect the other end of the NULL-modem cable to a free serial port on your host development system. For the purposes of this doc, we'll assume that it's ser1 (e.g. COM1 in Windows).
3. For host-target communications, make sure that your communications program (e.g. Qtalk, tip, or HyperTerminal) is set up properly. The target board uses vt100 terminal protocol by default.

For serial port use these settings:

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

Note

If you are using the GUI based kernel download tool on Windows it is recommended that you use HyperTerminal for the communication instead of the built-in serial terminal (close the terminal on the GUI tool).

4.7 Connect the target to network

Communication with the HiCO.SH7760 bootloader happens via serial port, but the kernel image is downloaded via ethernet. For this you need to connect the HiCO.SH7760 to your local network using the RJ-45 connector on the connector board.

4.8 Apply power to the target

You should see the output of the bootloader on your terminal program:

```
emtrion GmbH
Bootloader for HiCO.SH7760
Version 1.7 from $Date: xxxx/xx/xx xx:xx:xxx $
Copyright (c) 2000 - 2004
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
>
```

Note

You can check the bootloader version from this output. For this BSP the version should be should be 1.7 or later. A bootloader update you can find in [\[starterkitCD\]/bootloader](#)

4.9 *Transfer the OS image to the target*

Start the kernel download tool and select the newly created image (or the symbolic link `last_build.sre`) in the images directory. Here's an example session on authors' development host using Windows dos shell:

```
cd C:\work\hico7760\kerneldnld\cygwin
kerneldnld.exe c:\work\hico7760\images\last_build.sre
waiting for BOOTME...
```

The `kerneldnld` now waits for the target board to send a `BOOTME` message. On the terminal console to the target - select option *“Download Image via LAN91C111 Ethernet controller and store in Flash”*. The image is now downloaded to the target. When the download and flash programming are done, bootloader asks you to restart the board.

4.10 Test Neutrino on your target

After restarting the board give option 1 "*Execute stored Image*" to the boot-loader. The preinstalled image (hico7760.photon.sre) starts a minimum photon environment with the calculator application. Korn shell is invoked on the serial terminal. See images/hico7760.photon.build for startup Rutines.

5 Building Flash Images

In this chapter we go through some methods for building flash images for HiCO.SH7760. Before you try any of the described methods it is recommended that you first go through the steps in *chapter 4 "Getting Started"*

Note

There's only limited support for building flash images using in Momentics IDE. The preferred way is to write your own build file with a text editor and make the image by running the delivered mkflashimage script. See comments for variable IFS_BINARY_IMAGE in mkflashimage script for how to use a your own build file.

5.1 Example build files (IFS)

The ifs image which contains the actual OS needs a build script which defines its contents (see documentation for mkifs). HiCO.SH7760 BSP includes three example ifs build files:

hico7760-minimal.build

a minimal build to serve as a skeleton for your own builds. It includes serial communications, networking, flash and some shell tools.

hico7760-headless.build

includes USB, an advanced shell environment, and some extra tools, otherwise the same as minimal.

hico7760-photon.build

includes necessary files for graphics and simple Photon environment, otherwise the same as the headless image.

The default build file is hico7760.photon.build. You can change the build file by changing the value of BUILD_NAME and IFS_BUILD_FILE variables in the mkflashimage shell script.

By QNX convention you can find the same buildfiles hico7760.*.build in directories

```
prebuilt/shle/boot/build/  
src/hardware/startup/boards/hico7760/
```

5.2 Building your own IFS images

It is recommended that you:

1. take one of the example build files and copy it to a safe location.
2. Set the `IFS_BUILD_FILE` variable in the `mkflasimage` script to point to the copied build file.
3. You can now build the image by running `make` in the `images` directory.

You can also build the flash image by running the `mkflasimage` shell script directly:

```
cd path_to_bsp/images  
./mkflashimage
```

This way you can give parameters to the script as well (see the `mkflasimage` script).

Caution

If you have the BSP sources, the build process (i.e. Running `make` in the top level BSP directory) will copy the buildfiles in `src/hardware/startup/boards/hico7760/` over the ones in `images/`. That is, you will lose any changes you made to them.

5.3 Including Embedded Filesystem Images (EFS)

The efs sections are included by giving an efs build file as a parameter to the delivered mkflashimage shell script. For example

```
./mkflashimage my_root.bld photonbins.bld
```

would produce image ipl-ifs-my_root-photonbins.sre

You can also use the EFS_BUILD_FILES variable in the mkflasimage script (see the script) to give a list of efs build files to include.

Note

Combining images for HiCO.SH7760 with the QNX mkimage utility is not supported. Use always the mkflashimage delivered with this BSP (see the script for more information).

5.4 Creating empty flash filesystem partitions

You can create flash file partitions generally in two ways - with the flashctl utility directly on the target, or you can include an empty efs image to the downloaded image as explained in *chapter 5.3 "Including Embedded Filesystem Images (EFS)"*.

For example, the image which is pre-programmed to the HiCO.SH7760 includes an empty efs partition which is mounted to /root. This means that /root directory is persistent read/write storage for the runtime programs.

```
$ cd bspdir/images
$ ./mkflashimage emptyefs.bld
.
.
.
NAME START END SIZE
ipl-ifs 0x00040010 0x00800000 7935k
emptyefs 0x00800000 0x00880000 512k
-----
total 8447k
unused 24320k
file name : ipl-ifs-emptyefs.sre
last_build.sre linked to ipl-ifs-emptyefs.sre
```

On the target which contains the delivered default image has following partitions in the /dev directory:

/dev/fs0

raw entry to the whole flash. You should leave this untouched if you don't know what you're doing.

/dev/fs0p0

raw entry to the ifs image. You shouldn't be doing any changes to this either.

/dev/fs0p1

Raw entry to the efs partiton we included to the downloadable image.

/dev/fs0p2

Rest of the flash which is unused. You can create new partitions here.

Here's an example session how to create a flash partition on the target with the flashctl utility:

```
# 1. Before we do anything with the partition we erase it
$ flashctl -p /dev/fs0p2 -ev
Erasing device /dev/fs0p2
.....
# 2. Format a ne filesystem partition
$ flashctl -p /dev/fs0p2 -f
# 3. Kill the flash driver and start it again
$ slay devf-generic
$ devf-generic -s0xa0000000,32M&
# The new filesystem partition was automatically mounted to
/fs0p2
$ ls /
bin etc lib root tmp
dev fs0p2 proc sbin usr
```

Note

flashctl utility is available in the QNX Flash Technology Development Kit (Flash TDK).

6 Getting started with SW development

There are generally two ways to work with the QNX Momentics 6.3 tools:

- Command line - You use your favourite editor for editing code, write your own Makefiles and with time build your own development environment. On windows this would be the Cygwin environment (www.cygwin.com)
- Momentics IDE - The QNX IDE provides virtually all you need to build software/images for the target.

If you are an experienced GNU/Linux/Unix user, you'll probably feel yourself at home with the command line utilities. If you, however, are a beginner with UNIX like environment it's better to start with the QNX Momentics 6.3 IDE and perhaps later examine more what happens "under the hood". "Programmer's Guide" in the delivered documentation helps you to get started with the command line tools and Makefile conventions. "Momentics IDE - Users Guide" is the documentation you want to read for developing with the IDE.

6.1 Cross-debugging with Momentics IDE

1. Open a terminal window on your target (For example Hyperterminal on windows)
2. start the TCP/IP networking on the target. For example by giving commands:

```
io-net -d smc9000 iorange=0xa4000300,irq=0 -p tcpip &  
ifconfig en0 192.168.105.52 #Change the IP address to  
match yours!
```

(for more info see the hico7760-net script in the example build file)

3. Start the target agent by entering this command:

```
qconn
```

4. Open Momentics IDE and start a new QNX C project: *File->New->Project->QNX C Project*. Give a project name and select next. From the target architecture list select only SH little endian and then finish.
5. compile the example program: *Project->Build All*
6. Create a debugging target: *Run->debug As-> C/C++ QNX QConn*. You will get a dialogue window asking some target information. Important here is to give the correct target IP address (The same IP address as above). If you used dhcp.client to get the IP for the target, use ifconfig to get the correct address.
7. After clicking OK the IDE should connect to the target and you should now be able to debug the program

6.2 Cross-debugging with GDB

1. On the target, start the pdebug server on port 8000 (you have to have network running):

```
pdebug 8000&
```

2. Open a command console on your host and change to the directory where the executable to be debugged is located and start gdb.

```
cd path-to-my-executable
ntosh-gdb.exe
.
.
This GDB was configured as "--host=i686-pc-cygwin --
target=ntosh".
```

3. under gdb shell connect to the target (change the IP address to match yours)

```
(gdb) target qnx xx.xx.xx.xx:8000
Remote debugging using 192.168.105.52:8000
The target is assumed to be little endian
Remote target is little-endian
```

4. read symbols from your executable

```
(gdb) sym my_executable_g
```

Reading symbols from my_executable_g...done.

5. upload the the program to target

```
(gdb) upload my_executable_g /tmp/my_executable_g
```

6. set program arguments and start the debug session

```
(gdb) set args /tmp/my_executable_g
(gdb) break main
(gdb) run
Starting program: /tmp/my_executable_g
```

7 BSP Drivers

Drivers provided by QNX are documented in the QNX Momentics documentation. This chapter includes only drivers written/provided by emtrion GmbH

7.1 A/D and Touch Driver (*adcmgr*)

7.1.1 Introduction

adcmgr is a QNX 6.X resource manager for Renesas SH7760 internal A/D Controller. The driver provides also a touch interface as implemented on the EDOSK7760 extension board from and HiCO.SH7760 boards from emtrion GmbH.

adcmgr creates file-system entries for the AD-channels and for the Touch. The touch interface can be disabled, in which case, all the AD-channels get own file entries.

Example:

```
Touch interface using channels 0,1:  
/dev/adc/touch  
/dev/adc/2  
/dev/adc/3  
Touch interface disabled:  
/dev/adc/0  
/dev/adc/1  
/dev/adc/2  
/dev/adc/3
```

7.1.2 Touch interface

The touch interface provided by `adcmgr` can be used with the `devi-hico` photon input driver. Example command:

```
adcmgr -tx=1, y=0, ymin=90, xmin=110, maxd=10, sr=10 &  
waitfor /dev/adc/touch  
devi-hico emtouch sfd abs -x -y -f./touch.calib
```

See also the `boardcode-ph` script in the BSP build file.

7.1.3 `adcmgr` Command line parameters

-v[level]

Set verbosity level. 0 means no output. Default is 1.

-v

print version and exit

-d

debug mode. Values are returned as strings. With this option you can read the values with shell tools like `cat`.

-t{params}

Activate touch with given paramters. Parameters are:

- **x** - X AD channel
- **y** - Y AD channel
- **xmin** - minimal x value (default 0)
- **ymin** - minimal y value (default 0)
- **sr** - sample rate (default 10ms)
- **maxd** - maximum standard deviation (default 15)
- **irq** - QNX irq vector for the touch.

the `xmin`, `ymin` and deviation parameters are used for error correction. You can get the `xmin` and `ymin` values by touching to the lower/upper/left/right thresholds of viewing area and reading the values. Samples below the minium values are ignored.

The sampled values are checked against standard deviation of three samples. A small value will filter out fast moves and thus possible errors. A higher value will let the fast moves through, but you can also expect some erroneous samples (due to touching the screen too lightly etc.)

With the sample rate you can adjust the response time. A lower value will make lower the response time, but will also put a greater burden on the processor.

Examples:

- Start the driver without touch interface:

```
# adcmgr &
# ls /dev/adc
0 1 2 3
```

- Start the driver without touch in debug mode:

```
# adcmgr -d&
# cat /dev/adc/1
90 91 92 91 91 91 90...
```

- Start the driver with touch interface

```
# adcmgr -tx=1,y=0,ymin=90,xmin=110,maxd=10,sr=10 &
# ls /dev/adc
2 3 touch
```

- Start the driver with touch interface in debug mode (you have to touch the screen to get some data displayed):

```
# ./adcmgr_g -tx=1,y=0,ymin=90,xmin=110,maxd=10,sr=10 -
d&
# cat /dev/adc/touch
x=527, y=446, buttons=0x4, count=3, xdev=0.00, ydev=0.00
x=527, y=445, buttons=0x4, count=4, xdev=0.00, ydev=0.58
x=527, y=445, buttons=0x0, count=8, xdev=0.00, ydev=0.58
.
.
```

7.1.4 Reading AD Values

The device files are read with the POSIX file-system calls `open()`, and `read()`. The 10bit values are read as integers. Example:

```
int fildes, value, ret;
/* open the device for reading */
fildes=open("/dev/adc/1",O_RDONLY);
ret=read(fildes,&value,sizeof(int));
printf("read value %d from AD channel 1\n");
.
.
```

7.1.5 Reading touch coordinates

A read call on the `/dev/adc/touch` entry will return a datapacket which contains the `x/y/buttons` information. The datapacket is defined in `em-touch.h`. All of the device nodes are read only.

7.2 CAN Driver (*hcan2mgr*)

7.2.1 Introduction

hcan2mgr is a QNX 6.X resource manager for HCAN2, Renesas SH7760 internal CAN controller. Before starting with the driver, you should take an overview on the HCAN2 data sheet in the SH7760 documentation.

hcan2mgr creates POSIX filesystem API for the 32 Mailboxes of the HCAN2 controller. After starting the driver you can find following entries in the /dev directory:

```
/dev/can[0:1]/[0:31]/data  
/dev/can[0:1]/[0:31]/status
```

where [0:1] is the CAN channel number and [0:31] is the mail box number.

7.2.2 CAN channels

SH7760 has two identical CAN channels, which both have 32 mailboxes. One *hcan2mgr*

process controls one channel. If you want to use both CAN channels you have to start

two *hcan2mgr* instances:

```
hcan2mgr -c0&  
hcan2mgr -c1&
```

7.2.3 Mailboxes

One HCAN2 CAN controller has 32 mailboxes (0-31) where mailbox 0 is read only. Here are the characteristics of the mailbox system:

- A mailbox can be configured as Receiving or Transmitting, not both.
- Every mailbox has its own entry in `/dev/can[0:1]/[0:31]`.
- Mailbox entry can operate in two data modes: PIPE or CAN_FRAME.
- 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 acceptance, starting from mailbox 31.
- Mailbox 0 is read only.

You can open the mailboxes as if they were independent devices. Multiple opens for one mailbox is not allowed.

7.2.3.1 Tx/Rx buffers

Every mailbox has a buffer which is transmit buffer in transmit mode and receive buffer in receive mode. You can set the size of the buffer in the configuration file. The buffer size is given in CAN messages and can be anything from 0 to memory limit.

Each buffer entry takes 24 bytes of memory, so setting a buffer of 1000 CAN messages for one mailbox would consume about 25kB of RAM.

When you start `hcan2mgr` it starts to fill receive buffers with incoming CAN traffic (given of course that it passes the acceptance masking).

7.2.3.2 PIPE mode

In pipe mode the mailbox settings given in `hcan2.conf` are used statically for the CAN traffic. This means that you will be writing and reading 'pure' data to/from the mailbox entry (i.e. no CAN identifiers or DLCs). A simple shell script demonstrates this:

```
echo "hello world" > /dev/can0/1/data
```

In this case the receiver would get two CAN messages. First with 8 bytes of data "hellowo" and second 4 bytes "rld\n" (See also chapter "Sending/Receiving CAN messages from shell").

7.2.3.3 CAN_FRAME mode

In can frame mode CAN messages are written and received as C data structures (see `can2_api.h`):

```
typedef struct _CanMsg
{
    unsigned int id; /* Standard ID, RTR, IDE, extended ID */
    unsigned short dlc; /* Data Length Code */
    unsigned short ts; /* TimeStamp */
    unsigned char data[8]; /* Message data */
}CanMsg;
```

7.2.3.4 Which mode should I use?

Here are some rules to give you direction:

- If you want to write CAN messages with multiple CAN IDs from the same mailbox - use CAN_FRAME mode.
- If you want to receive CAN messages with diverse CAN IDs to one mailbox – use CAN_FRAME mode.
- If you want to read/write data from shell - use PIPE mode.
- If you want to keep things simple - use PIPE mode.

If you have, for example, 15 or less CAN nodes to communicate with, you could set on receiving mailbox and one sending mailbox per CAN node, set all mailboxes in PIPE mode and write/read data to them as if they were POSIX pipes.

7.2.4 Configuration file

hcan2mgr needs a configuration file as input (default is `/etc/hcan2.conf`). This configuration file can be generated with the `confgen.py` Python script (delivered with the driver). The script calculates the required register settings.

1. Write your CAN configuration into the `confgen.py` (Acceptance masks CAN identifiers etc.)
2. generate `hcan2.conf` by running the script:

```
python confgen.py > hcan2.conf
```

(If you don't have the Python interpreter on your development host you can download it from <http://www.python.org/download/>)

1. Start `hcan2mgr` with the new configuration.

The `confgen.py` script is well commented and the syntax should be self explanatory.

7.2.4.1 Baudrates

Required baudrate settings for the HCAN2 controller are calculated in the `confgen.py` script:

```
baud_rates=[10000,20000,50000,100000,  
125000,250000,500000,800000,1000000]
```

You can add your own baudrates or remove unneeded ones if you like.

7.2.4.2 Mailbox Settings

7.2.4.2.1 CAN identifier

The CAN ID is integer (32 bits), which contains both, standard and extended ID. The binary format is:

```
Nsss ssss ssss RIee eeee eeee eeee eeee
```

where:

- N: not used
- s: standard ID
- R: Remote Transmission Request
- I: Identifier Extension
- e: extended ID

When a mailbox is configured for receiving data, the CAN identifier is used as acceptance identifier (see next chapter)

7.2.4.2.2 Acceptance Masking

Acceptance masking for receiving mailboxes is configured in `hcan2.conf`. Acceptance Mask has the same format than the CAN Id with the exception that RTR and IDE bits have to be always 0. See the HCAN2 data sheet in the SH7760 documentation for more info.

Important

When setting the acceptance masking you should keep in mind that a CAN message is written only to the first matching mailbox (starting from 31).

Here are some examples:

Receive all CAN messages

```
if mbox==0:  
    settings['mbc']=MBC_RECEIVE  
    settings['amask']=0xffffffffL
```

Receive only CAN messages with CAN identifier 0x1a000000

```
if mbox==0:
    settings['mbc']=MBC_RECEIVE
    settings['canid']=0x1a000000L
    settings['amask']=0x00000000L
```

Receive all CAN messages where bits 4-7 of the standard identifier are 0x0-0xf and all other bits are 0.

```
if mbox==0:
    settings['mbc']=MBC_RECEIVE
    settings['canid']=0x00000000L
    settings['amask']=0x0f000000L
```

Note

If you use PIPE mode, it is recommended that you set the acceptance mask to 0x00000000. This means that the ID of an incoming CAN message has to match exactly to the one written in the mailbox ID fields (settings['canid']). If you receive CAN messages with different CAN IDs to the same mailbox in PIPE mode, you have no way of knowing which node sent the data (unless you write your own identifier fields in the data)

7.2.5 hcan2mgr Command line parameters

-v[level]

Set verbosity level. 0 means no output. Default is 1.

-c[channel]

Choose CAN channel [0:1]. Default is 0.

-f[filename]

Configuration file path. Default is hcan2.conf in the same directory.

-b[baudrate]

Baudrate for the channel (corresponds to the value in the configuration file). Default is 20000.

-V

Print version and exit

7.2.6 Driver API

As earlier mentioned, the Driver API is the POSIX filesystem API. In this chapter you can find a small description and the special characteristics of the system calls you can use with the driver.

See QNX library reference for prototypes and more information of these system calls.

7.2.6.1 *open()*

`open` system call on a `/dev/can[0:1]/[0:31]/data` entry will return a file descriptor which is the 'handle' to the opened mailbox. `open` returns with `errno` set to `EINVAL` if you try to open a receiving mailbox for writing or transmitting mailbox for reading. If the CAN controller is in erroneous state (e.g. `BUS_OFF`) `open` returns with `errno` set to `EIO`. Example:

```
/* open mailbox 0 of can channel 0 for reading */
mailbox_fd=open("/dev/can0/0/data",O_RDONLY);
```

Open on any of the status entries will always succeed.

7.2.6.2 *read()*

`read()` system call will read one CAN message from the `hcan2mgr` receive buffer and write it to the given data buffer.

In PIPE mode only the telegram data is written to the data buffer and the return value corresponds to the data length (dlc).

If the mailbox is in `CAN_FRAME` mode `read()` reads one `CanMsg` structure (defined in `hcan2_api.h`) from the receive buffer to the given buffer and returns `sizeof(CanMsg)`. Example:

```
/* Read one message in PIPE mode */
ret=read(fildes,buffer,8);
printf("read a CAN message with %d bytes of data\n",i);
```

7.2.6.3 write()

write() system call writes one message directly to the CAN controller. If the controller is busy sending the message is written to the transmit buffer instead. Characteristics in PIPE and CAN_FRAME mode are as with read(). Example:

```
CanMsg msg;
.
.
/* write one CAN message in CAN_FRAME mode */
i=write(fildes, &msg, sizeof(CanMsg));
printf("wrote %d bytes\n", i);
```

7.2.6.4 return values

write and read system calls always return the number of read/written bytes on success and -1 on error with errno set to the error value (as per POSIX). In CAN_FRAME mode the number of read/written bytes should always be sizeof(CanMsg). In PIPE mode the value can be 0-8.

7.2.6.5 Blocking and non-blocking operations

If you open a mailbox with O_NONBLOCK flag, read() and write() syscalls will return with errno set to EAGAIN if the transmit/receive buffer is full/empty.

In blocking mode (default) write() and read() will block until there's data to be read in receive buffer or space left in transmit buffer. Example:

```
/* read data in PIPE mode with non-blocking read */
mbox_fd=open("/dev/can0/6/data", O_RDONLY|O_NONBLOCK);
i=read(mbox_fd, buffer, 8);
if(i==-1 && errno==EAGAIN) printf("no data to read\n");
```

7.2.6.6 Device files

/dev/can[0:1]/status

Status of the CAN channel can be read here as an integer (seehcan2_api.h)

/dev/can[0:1]/[0:31]/status

Mailbox status can be read from here as an integer, (seehcan2_api.h). Note that reading will also clear the status flags.

/dev/can[0:1]/[0:31]/data

Data read/write entry to the mailbox.

7.2.6.7 Example C-Program

The example program canrw (delivered with the driver) gives you a full example. Following snippet gives you a hint what it's like (without error checking):

```
#include "hcan2_api.h"
int main(int argc, char *argv[])
{
    int fildes,i;
    CanMsg msg;
    int mailbox_no=3;
    int can_channel=0;
    char mailbox[100];

    /* put some data to the CAN frame */
    msg.data[0]='a';
    msg.data[1]='b';

    /* set message length */
    msg.dlc=2;

    /* set CAN Identifier */
    msg.id=0x1a000000;

    /* open a CAN mailbox.. */
    sprintf(mailbox, "/dev/can%d/%d/data", can_channel,
            mailbox_no);
    fildes=open(mailbox,O_WRONLY);

    /* ..and write the CAN frame */
    i=write(fildes,&msg,sizeof(CanMsg));
    printf("wrote %d bytes of data\n",i);
}
```

7.2.6.8 Sending/Receiving CAN messages from shell

If you use shell for writing data between CAN nodes you probably want to use the PIPE mode (see *chapter 7.2.3.2 "PIPE mode"*). If a Mailbox is in PIPE mode you can use the mailbox/data entries as if they were normal UNIX pipes. Following example demonstrates this.

```
cat source_file.txt > /dev/can0/1/data &  
cat /dev/can1/1/data > target_file.txt
```

The example, of course, assumes that the given mailboxes are configured properly and that the two CAN nodes are connected together. Following configuration in `confgen.py` would be sufficient:

CAN channel 0 configuration script:

```
if mbox==1:  
    settings['mbc']=MBC_TRANSMIT  
    settings['bufsz']=10  
    settings['canid']=0x1a000000L
```

CAN channel 1 configuration script:

```
if mbox==1:  
    settings['mbc']=MBC_RECEIVE  
    settings['bufsz']=100  
    settings['canid']=0x1a000000L
```

When writing data to the CAN pipe, the driver always takes maximal only 8 bytes from the data, so a more efficient way would be:

```
dd bs=8 if=source_file.txt of=/dev/can0/1/data &  
cat /dev/can1/1/data > target_file.txt
```

7.2.7 Error Handling

If an error occurs the system calls `open`, `write` and `read` return `-1` and `errno` is set to the error value (as per POSIX). When a system call returns `EIO`, the controller or CAN bus is in erroneous state.

Use the `-v[verbosity]` option for `hcan2mgr` to get more information why a certain error is returned.

7.2.7.1 CAN bus errors

Functions `write()`, `read()` and `open()` for data entries will return with `errno` set to `EIO` if the controller is in `BUS_OFF` or `PASSIVE_ERROR` state (with exception that `read()` is able to read from the receive buffer until it's empty).

You can read the error from the `/dev/can[0:1]/status` entry as an integer (see `hcan2_api.h`).

7.2.7.2 Mailbox Errors

You can read the status of the mailboxes from `/dev/can[0:1]/[0:31]/status`.

A read call to one of these entries returns the mailbox status as an integer (see `hcan2_api.h`).

Important

Reading the status value will also clear it (i.e. the internal status variable will be reset).

7.2.8 Controller reset

In order to reset the HCAN2 controller/driver you have to restart the driver (i.e. kill it and start it again). This would have to be done, for example, when the controller gets in `BUS_OFF` state.

7.3 Photon input driver (devi-em)

devi-em is a Photon driver specially for the touch interface provided by adcmgr. You can start the driver after starting the adcmgr. Example:

```
devi-em emtouch sfd abs -x -y -f./touch.calib
```

The driver opens by default the /dev/adc/touch device file and starts to read data from there. A calibration file is required for correct function with Photon. See QNX documentation for calib utility. For the abs filter module parameters see the QNX documentation, for example, on devi-elo.

See also the hico7760-ph script in the build file for an example how to start the driver.

7.4 I2C Driver (*i2c-camelot*)

BSP directory `src/hardware/i2c` contains the necessary information and header files to use the `i2c-camelot` driver. Source code for this driver is not available. RTC utility `rtc8564` serves as a programming example. See README file in the `src/hardware/i2c` directory for more information.

In order to make the `i2c` api globally available – you can copy the `<BSP>/src/hardware/i2c.h` header file from the BSP sources into `<QNX_TARGET>/usr/include/hw/i2c.h`.

7.5 RTC8564 Utility (*rtc8564*)

`rtc8564` reads or sets the time value from the battery backed up real time clock RTC8564. The time value printed or read by `rtc8564` is compatible with the `date` utility. Examples:

```
# read time value from rtc8564
rtc8564
192413071803.03

# set rtc to Tue Jul 26 11:34:12 2005
rtc8564 -s200507261134.12

# Set the system clock to the RTC time
date $(rtc8564)
```

Note

This utility requires the `i2c-camelot` driver.

8 BSP Utilities

8.1 *kerneldnld* - Download Tool

8.1.1 Description

The *kerneldnld* is used to download OS image files to emtrion target boards. The command has following synopsis:

```
kerneldnld [-v level] [-V] {-i boardID} {imagefile}
```

When the program is executed it starts to listen to the network and wait for a BOOTME command from a target board with a specific identifier *boardID*;. If you don't use the *boardID* option, BOOTME is accepted from any target. When the command is received the *imagefile* is downloaded. Note that *kerneldnld* doesn't initiate the download. The user initiates it by choosing an appropriate command from the bootloader menu.

8.1.2 Command line options

-v level

Set verbosity level. Default level is 1. Increase it for more debug output or set it to 0 for no output

-V

Print version and exit

-i boardID

identifier of the board. You can find this from the bootloader output messages

8.1.3 Examples

download a s-record file to a target board with id HiCOSH4212

```
kerneldnld -i HiCOSH4212 ip1-ifs.sre
```

9 PC104 (ISA) bus

If you use HiCO.SH7760 with the emtrion PC104 base board it is possible to access the ISA bus area through the PCMCIA area in the SH7760 processors address space. The ISA bus access has a couple of extra issues you have to take care of:

- Special memory mapping
- ISA bus Interrupts must be initialised (unmasked)
- You have to issue an end-of-interrupt

How this is done is shown in the `src/hicodio/*` examples. The sources are an example case for HiCO.DIO PC104 Digital I/O board (16xDI, 8xDO).

9.1 ISA-IO/MEM Mapping

Following table shows the ISA bus Physical mappings on HiCO.SH7760.

Memory area	Physical Address	P2 area access
ISA memory	0x16000000- 0x16FFFFFF	0xB6000000- 0xB6FFFFFF
ISA IO	0x17000000- 0x170FFFFFF	0xB7000000- 0xB70FFFFFF

Mapping of ISA memory/io areas to the processes address space cannot be done directly with the `mmap_device_io/mem`. In file `src/hicodio/isalib.c` is a function which shows how to do the mapping.

9.2 ISA Interrupts

PC104 cards are allowed to use only ISA interrupts 5,9,11 and 12. Other interrupt requests are ignored. See chapter 67 “*HiCO.SH7760 interrupts*”, for how they are mapped to QNX interrupt vectors.

ISA bus interrupts have to be unmasked (once) by writing 0 to address 0x170C0000 and every interrupt has to be acknowledged by issuing an end-of-interrupt EOI to address 0x17080000. The value written to the EOI register is the ISA bus irq number (not the QNX interrupt vector).

10 HOWTOs

This is a collection of useful "How To" documents.

10.1 Establishing a Telnet connection to the target

1. Download the "headless" or "photon" example image to the target.
2. Configure and start the network on the target by running the `/proc/boot/hico7760-net` script:

```
hico7760-net
```

by default the script uses DHCP to configure the network, but you can give the network parameters manually as well (study the script and run the commands manually).

3. Start the `inetd` `inetrnet` daemon on the target:

```
inetd &
```

`inetd` uses `/etc/services` and `/etc/inetd.conf` for the configuration (see build file)

4. You can now open a telnet session from your host to the target. Open a command console and give command:

```
telnet [targets_ip_address]
```

10.2 Finding out library dependencies

When you include a program to your build file you also have to include the libraries it needs. There are generally two ways to find out the dependencies:

- On the host command:

```
ntosh-objdump.exe -x application |grep NEEDED
```

will list the needed libraries.

- Download the executable to the target, set the DL_DEBUG environment variable and run the program. Here's an example:

```
# export DL_DEBUG=1
# phcalc &
[1] 217112
# load_object: attempt load of libAp.so.3
load_elf32: loaded lib at addr 78200000(text)
78210050(data)
load_object: attempt load of libph.so.3
.
.
.
```

The output will tell you which libraries are missin or could not be loaded.

10.3 mounting USB Mass storage

It is here assumed that you have an USB flash stick with the dos (i.e. the normal Windows) file system.

1. Download the headless or photon example image to the target.
2. Start the usb subsystem (see build file for the script):

```
hico7760-usb
```

3. attach now your usb mass storage device to the USB port and check with the usb utility that the device is found:

```
usb
USB 0 (Biscayne) v1.10, v1.01 DDK, v1.01 HCD
Device Address : 1
Vendor : 0x0ea0 (USB )
Product : 0x2168 (Flash Disk )
Class : 0x00 (Independant per interface)
```

Here an USB Flash stick was detected.

4. Start the devb-umass USB mass storage driver:

```
devb-umass &
```

you should now find a /dev/hd0 and /dev/hd0tX entry for the device. To get some information about the entry give command:

```
fdisk /dev/hd0 info
```

5. You can now mount the partition with command:

```
mount -t dos /dev/hd0tX /foo
```

See also chapter Neutrino users guide -> Connecting Hardware -> USB in the QNX documentation.

Note

Not all USB mass storage devices are recognized. If you encounter any problems try a device from another manufacturer.

10.4 Mapping network shares

A great way to speed up the development is to map an external network drive on your target board. For example a directory on your development host so that you can exchange files between host and target fast. Here is an example session for Windows host:

1. On your windows host open the explorer (right click on the start menu -> explorer)
2. select some directory you want to share with the target and right click on it.
3. From the popup menu select sharing and check the share this folder radio button. Give some name to the share and set the access rights. *You have to accept the settings before you push the ok button and exit.*
4. On the target, start the networking (see hico7760-net) and start the fs-cifs:

```
fs-cifs -L -a -dDOMAIN \  
//HOST_NAME:xx.xx.xx.xx:/SHARE_NAME \  
/mount_point username
```

Replace the DOMAIN, HOST_NAME and SHARE_NAME to match your network settings. For example on authors system command

```
fs-cifs -L -a -dintern \  
//NY_XP:192.168.105.109:/hico7760-work /tmp2 ny
```

would map a share with name hico7760-work to the /tmp2 on the target. Whether you need the -dDOMAIN option depends on your network configuration.

For mapping external directories you can use Network File System (NFS) as well (see QNX documentation for fs-nfs2 fs-nfs3). See also chapter "Working with Filesystems" of the Neutrino User's Guide.

10.5 Calibrating the Touchscreen

Touchscreen can be calibrated with the `calib` utility. In order to get the calibration data for the calibration file (`touch.calib` by default) run the command with `-v` option:

```
# calib -v
Point 0: 982,106
Point 1: 973,945
Point 2: 85,944
Point 3: 86,95
Calibration: 0x0:639x479:1008 54 60 984 0
```

Copy and paste the marked calibration data into the file. Here's an example how to automatize the task with a bit shell string manipulation:

```
# Save the calib output into a shell variable
LINE=$(calib -v | grep 'Calibration:')

# Echo the data into the calibration file
echo ${LINE#*: } > /tmp/touch.calib
```

11 Getting support

A good place for discussion and asking advice are the free QNX newsgroups at inn.qnx.com. emtrion GmbH support can help you with issue related to this BSP and HiCO.SH7760 hardware. To handle support requests, following information are required:

- Company name, contact person and a valid e-mail address
- Product name and version (include the name of the zipped/tarred BSP package).
- Information about your development environment (which host, installed patches,etc.)
- Detailed description of the problem and, if possible, how it can be reproduced.

You can mail your support request to support@emtrion.com.

12 HiCO.SH7760 interrupts

12.1 Interrupt Priorities

Interrupt Priorities are written in `init_intrinfo.c` in the BSP start-up files. Priorities for the external interrupts cannot be defined with software.

12.2 SH7760 external interrupts

Name ⁽¹⁾	SH7760 exception	Priority ⁽²⁾	QNX irq vector
Ethernet, LAN91C111	0x200	15	0x0
HiCO.nect (code 0x1)	0x220	14	0x1
HiCO.nect (code 0x2)	0x240	13	0x2
USB Function, ISP1181	0x260	12	0x3
HiCO.nect (code 0x4)	0x280	11	0x4
HiCO.nect (code 0x5, I-SA_IRQ9)	0x2A0	10	0x5
CF Interface	0x2C0	9	0x6
HiCO.nect (code 0x7, I-SA_IRQ11)	0x2E0	8	0x7
HiCO.nect (code 0x8, I-SA_IRQ12)	0x300	7	0x8
Touch (pen up/down)	0x320	6	0x9
HiCO.nect (code 0xA, I-SA_IRQ5)	0x340	5	0xa
MMC card (detect change)	0x360	4	0xb
COM1 (DCD change)	0x380	3	0xc
HiCO.nect (code 0xD)	0x3A0	2	0xd
Real Time Clock (RTC)	0x3C0	1	0xe

1) "Code" means the level on IRL0-IRL3 lines

2) 15 = highest priority

12.3 SH7760 internal interrupts

Function/Peripheral	QNX irq vector
Hitachi-UDI (UDI)	0x2010
Cascade from GPIO (GPIO)	0x2011
DMAC 0 transfer end (DMTE0)	0x2012
DMAC 1 transfer end (DMTE1)	0x2013
DMAC 2 transfer end (DMTE2)	0x2014
DMAC 3 transfer end (DMTE3)	0x2015
DMAC address error (DMAE)	0x2016
DMAC 4 transfer end (DMTE4)	0x201C
DMAC 5 transfer end (DMTE5)	0x201D
DMAC 6 transfer end (DMTE6)	0x201E
DMAC 7 transfer end (DMTE7)	0x201F
IRQ4	0xC000
IRQ5	0xC001
IRQ6	0xC002
IRQ7	0xC003
SCIF0 (ERIO)	0xC004
SCIF0 (RXIO)	0xC005
SCIF0 (BRI0)	0xC006
SCIF0 (TXIO)	0xC007
CAN Channel 0 (HCAN0)	0xC008
CAN Channel 1 (HCAN1)	0xC009
I2S0	0xC00A
I2S1	0xC00B

HiCO.SH7760 QNX 6.3 Starterkit

AC97I0	0xC00C
AC97I1	0xC00D
I2C0	0xC00E
I2C1	0xC00F
USB	0xC010
LCDC VINT	0xC011
USB DMA cascaded (BRG BRGI0)	0xC014
A0 DMA cascaded (BRG BRGI1)	0xC015
A1 DMA cascaded (BRG BRGI2)	0xC016
SCIF1 (ERI1)	0xC018
SCIF1 (RXI1)	0xC019
SCIF1 (BRI1)	0xC01A
SCIF1 (TXI1)	0xC01B
SCIF2 (ERI2)	0xC01C
SCIF2 (RXI2)	0xC01D
SCIF2 (BRI2)	0xC01E
SCIF2 (TXI2)	0xC01F
SIM Card Module ERI	0xC020
SIM Card Module RXI	0xC021
SIM Card Module TXI	0xC022
SIM Card Module TEI	0xC023
SPI	0xC024
MMC MMCI0	0xC028
MMC MMCI1	0xC029
MMC MMCI2	0xC02A
MMC MMCI3	0xC02B

HiCO.SH7760 QNX 6.3 Starterkit

MSIF MINTI	0xC02C
MSIF MERI	0xC02D
MSIF MTRSI	0xC02E
MSIF MTEI	0xC02F
MSIF MREI	0xC030
MSIF MITOI	0xC031
MSIF MHSTOI	0xC032
MSIF MPEI	0xC033
MSIF MFII	0xC034
FLCTL FLSTE	0xC038
FLCTL FLTEND	0xC039
FLCTL FLTRQ0	0xC03A
FLCTL FLTRQ1	0xC03B
ADC ADI	0xC03C
CMT CMTI	0xC03D
USB Transfer End (BRG cascade)	0xC100
USB Addr Error (BRG cascade)	0xC101
A0 Tx End (BRG cascade)	0xC110
A0 Rx End (BRG cascade)	0xC111
A1 Tx End (BRG cascade)	0xC112
A1 Rx End (BRG cascade)	0xC113
A0 Tx Midpoint (BRG cascade)	0xC120
A0 Rx Midpoint (BRG cascade)	0xC121
A1 Tx Midpoint (BRG cascade)	0xC122
A1 Rx Midpoint (BRG cascade)	0xC123

12.4 External interrupt mask registers

See also hardware manual

HiCO.SH7760 interrupt mask low byte at 0xa4000000

7	6	5	4	3	2	1	0
ext8	touch	extA	MMC	com1	extD	rtc	X

HiCO.SH7760 interrupt mask high byte at 0xa4000002

7	6	5	4	3	2	1	0
lan	ext1	ext2	usb_f	ext4	ext5	CF	ext7