

User's Manual for the Boundary Devices Neon[®] board

April 5, 2005



April 5, 2005

Contents

1	Intended Audience	3
2	Overview of features	3
3	Hardware feature	3
3.1	Layout	3
3.2	Mounting	4
3.3	Connector reference	4
3.4	Electrical characteristics	5
4	Software features	6
4.1	Das U-Boot	6
4.1.1	Requirements for building under Linux	6
4.1.2	Requirements for building under Windows with Cygwin	6
4.1.3	General build steps	7
4.1.4	Tailoring U-Boot for your application	7
4.2	Windows CE	9
4.2.1	Prerequisites and components	9
4.2.2	BSP Installation	9
4.2.3	Building the demo	10
4.3	Linux Support	11
4.3.1	Toolchain	11
4.3.2	Kernel	11
4.3.3	Libraries and Applications	11
5	Development Tools	12
5.1	minidebug	12
5.1.1	mdebug	13
5.2	JTAG system-level debugger	13
5.2.1	Requirements	13
5.2.2	Startup Options	14
5.2.3	Control Keys	15
5.2.4	Blast protocol	15
5.2.5	Quick-start download and burn	15
5.3	TeraTerm blast extensions	17
6	Configuration Notes	17
6.1	LCD Panel configuration	17
6.2	Memory size configuration	18
6.3	Touch Panel Calibration	19

1 Intended Audience

This document aims to provide the information needed to integrate the Neon[®] board into your application. As such, it addresses both hardware and software integration.

2 Overview of features

The following are highlights of the Neon[®] board.

- Available with Windows Ce or Linux Operating Systems
- Full featured [Boot Loader](#) for custom startup
- 400 MHz Intel PXA-255 CPU
- 32 or 64MB SDRAM
- 8 or 32MB Intel StrataFlash (tm) EEPROM
- Silicon Motion SM-501 Graphics Controller
- Active Matrix LCD Support,
- Including Full-Motion Video
- STN Passive LCD Display Support
- 4 or 5-Wire Resistive Touch-Screen Support
- 44KHz Stereo 16-Bit Audio Output, for Headphones or Speakers
- 44KHz Monaural Audio Input (microphone)
- 1 RS-232 or TTL Serial Port
- 1 USB 1.1 Slave Port
- 1 USB 1.1 Master Port
- Built-in 10/100 Ethernet Controller,
- Built-in Interface for Magnetic Stripe Readers and Printers
- MMC Slot for Expanded Storage
- General Purpose I/O for Device Control
- Built-in Switching Power Supply for 5V DC Input
- JTAG Interface
- Customized Versions Available

3 Hardware feature

3.1 Layout

As shown in Figure 1, the Neon[®] board contains a wide variety of I/O options for use in your application. Note that some of these may not be populated on an evaluation or production board.

April 5, 2005

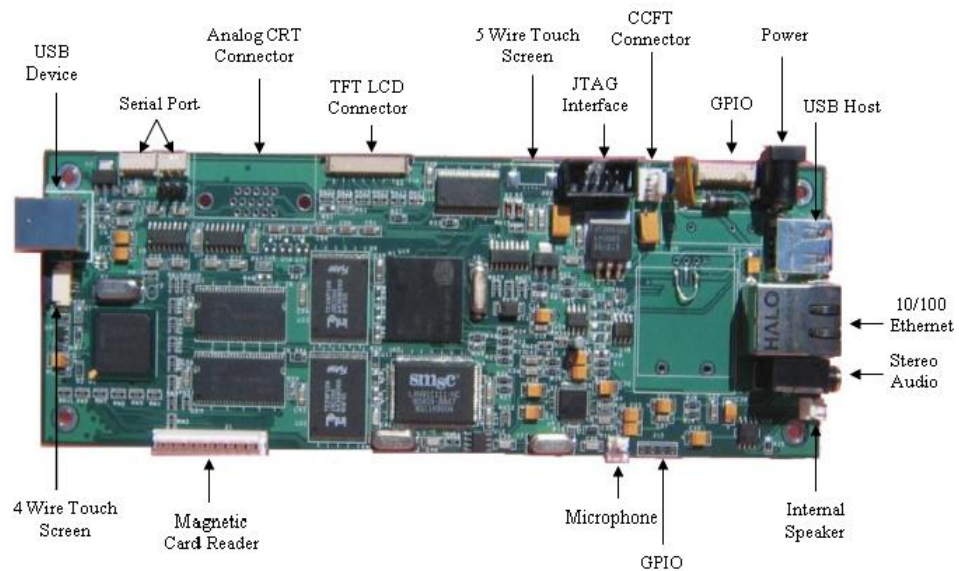


Figure 1: Neon board

3.2 Mounting

The Neon® board measures 2.75" by 6.75", slightly larger than the Hitachi® 6.2" display, to allow for easy mounting.

There are four mounting holes 1/4" from each edge in each of the four corners, and the holes are 1/8" in diameter.

3.3 Connector reference

The following is a list of all connector part numbers used on the Neon® platform for use in identifying mating parts for your application. Note that Boundary Devices will periodically switch vendors for these parts, but will notify you of any changes that require a new mating part.

Description	Manufacturer	Part
USB Master	FCI	87520-0010B
USB Slave	SINGATRON	KS-001-BNW
I2C	FCI	68897-001
Ethernet	Halo	HFJ11-2450E
Stereo Audio	Singatron	2SJ-43723N13
Backlight inverter	Molex	53048-0210
MMC/SD	AVX	14 5638 009 511 862
TFT Display		
Touch Screen	Molex	52207-0590
Serial Port	FCI	68897-001
JTAG	Molex	53048-0810

3.4 Electrical characteristics

4 Software features

As provided by Boundary Devices, the Neon[®] board supports either Windows CE 5[®] or Linux.

To simplify the installation of either, the [Das U-Boot](#) boot loader is installed on our evaluation boards, and two MMC cards are shipped to allow the use of either operating system.

4.1 [Das U-Boot](#)

The [Das U-Boot](#) Boot Loader is a full-featured loader for either Linux or Windows CE that supports a wide variety of options for loading your Operating System and application.

Boundary Devices ships U-Boot both as a binary image and as source code in the form of a patch that adds support for either Neon or BD-2003 devices.

The binary image may be burned directly to sector zero of the on-board flash.

The source code will require a set of Linux or [Cygwin](#)(Windows) tools for cross-compilation. The following section will detail the requirements and steps for building.

4.1.1 Requirements for building under Linux

Since the [Das U-Boot](#) project uses GNU tools, most of the required components will generally be available on a GNU/Linux system.

The three pieces which may not commonly be installed are the [bzip2](#) and [wget](#) packages and an ARM cross compiler.

Boundary Devices typically uses GCC-2.95.3 to create U-Boot images, since that matches what we use to build the Linux image to run on the Neon itself, but the binary distribution of GCC-3.4.3 from [GNUARM](#) is a nice alternative.

4.1.2 Requirements for building under Windows with [Cygwin](#)

There are two primary requirements for building under Windows.

The first, [Cygwin](#), provides a set of Unix utilities under the Windows operating system. Since the Cygwin installer allows components to be selected individually, the following list shows the requirements for building a [Das U-Boot](#) image with Neon[®] support. Note that this list is probably incomplete, but these should be the only required items which differ from the Cygwin defaults.

Base/diffutils
Devel/binutils
Devel/gcc
Devel/make
Devel/patchutils
Utils/bzip2
Web/wget

The second requirement for building is the X-Scale cross-compiler itself. The [GNUARM](#) project provides a wealth of information needed to build a cross-compiler for ARM processors. Thankfully, it also provides an [installer](#). As of this writing, Boundary Devices currently uses the GCC-3.4.3 package for [Cygwin](#).

4.1.3 General build steps

Quick start:

```
wget http://easynews.dl.sourceforge.net/sourceforge/u-boot/u-boot-1.1.2.tar.bz2
bzipcat u-boot-1.1.2.tar.bz2 | tar -xvf -
wget http://boundarydevices.com/uboot_neon_bd2003.diff
patch -p0 < uboot_neon_bd2003.diff
cd u-boot-1.1.2
CROSS_COMPILE=arm-elf- make neon_config
CROSS_COMPILE=arm-elf- make
```

Explanation.

The first four lines retrieve and extract the [Das U-Boot](#) sources and add support for the Neon[®] and BD-2003 devices.

The last two lines configure for the Neon[®] board itself, and finally, build a U-Boot binary. When complete, you'll find a file named `u-boot.bin` in your `u-boot-1.1.2` directory.

4.1.4 Tailoring U-Boot for your application

The Boundary Devices patches (`uboot_neon_bd2003.diff`) make a variety of decisions about the boot process which may not match with the needs of your application.

In general, the file `u-boot-1.1.2/include/configs/neon.h` defines these choices.

In particular, the distributed copy currently expects a Windows BMP file named `bdlogo.bmp` to be present on the MMC card and writes it to the display, then loads an operating system image from a file named `nk.nb0` to

RAM address 0xa0030000 and executes it.

Both of these are defined by the lines which resemble this:

```
#define CONFIG_BOOTCOMMAND "mmcinit; " \  
    "fatload mmc 0 a0008000 bdlogo.bmp ; " \  
    "bmp display a0008000 ; " \  
    "fatload mmc 0 A0030000 nk.nb0 ; " \  
    "g A0030000"
```

As mentioned previously, the [Das U-Boot](#) Boot Loader is a very capable loader with support for USB and network boot, including BOOTP/DHCP, and NFS mounting support. Please refer to the [Das U-Boot](#) website for details.

4.2 Windows CE

As mentioned earlier, the Neon[®] board ships with a runnable Windows CE 5.0 image on MMC card. A [Board Support Package](#) is also available and necessary to tailor the operating system for a given application.

The following sections describe the process of producing an image matching the one shipped with the Neon[®] board.

4.2.1 Prerequisites and components

Most of the tools needed to create a bootable Windows CE 5[®] application for the Neon[®] board are provided by Microsoft. The following is a complete list of components and where they may be obtained.

Windows CE 5 [®]	Microsoft
Embedded Visual C++ 4.0	Microsoft
Embedded Visual C++ Service Pack	Microsoft
Neon [®] Board Support Package	Boundary Devices

4.2.2 BSP Installation

The Neon BSP is made available as a Windows installer file on the [Boundary Devices](#) website. This file defines a single BSP for the BD2003 and SM501-supporting variants. Installation consists of running the `.msi` file.

```
c:\> wget http://www.boundarydevices.com/bspNeon320x240.msi
c:\> .\bspNeon320x240.msi
```

As a reference tool for the content of the BSP, you should consider using [MSI2XML](#) to view the content.

4.2.3 Building the demo

The [Platform Builder project](#) used to construct our sample image may be found on the [Boundary Devices](#) web site.

After installation of the BSP, this project may be copied to a new directory within the WINCE500 PBWorkspaces directory and built using Platform Builder.

```
C:\WINCE500\PBWorkspaces>md bdWeb
C:\WINCE500\PBWorkspaces>cd bdWeb
C:\WINCE500\PBWorkspaces\bdWeb>wget http://boundarydevices.com/bdWeb.pbxml
--17:37:40-- http://boundarydevices.com/bdWeb.pbxml
      => 'bdWeb.pbxml'
Resolving boundarydevices.com... 66.113.228.134
Connecting to boundarydevices.com[66.113.228.134]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 45,478 [text/plain]
100%[=====>]
17:37:40 (58.90 KB/s) - 'bdWeb.pbxml' saved [45478/45478]

C:\WINCE500\PBWorkspaces\bdWeb>.\bdWeb.pbxml
C:\WINCE500\PBWorkspaces\bdWeb>
```

After this is done, you should be able to build the sample application through the Build OS|Sysgen and Build OS|Build and Sysgen Current BSP menu options.

4.3 Linux Support

The Linux Environment for Boundary Devices boards consists of three primary pieces, a toolchain, the kernel and device drivers, and a set of libraries and applications.

In order to allow configuration of each, a modified version of the [PTXdist](#) tool is used to configure and build each.

4.3.1 Toolchain

Before the kernel and applications can be built, it is first necessary to have a cross-compiler toolchain. The following [package](#) contains a build script for building a GCC-2.95.3 toolchain under Linux.

The following is a short list of the usage steps.

You will be prompted for two paths during the build process: the archive directory and the installation directory. The archive path is used to store the source tar-balls for the compiler, binutils, and glibc. Keeping this separate from the xchain directory allows you to re-build the toolchain without downloading the source packages again.

```
$ wget http://boundarydevices.com/xchain.tar.gz
$ tar zxvf xchain.tar.gz
$ cd xchain
$ make all
```

Note that building the toolchain (especially glibc) takes forever (on the order of a couple of hours), so after you've supplied the directory information, find something fun to do for a while!

We typically use `/armArchives` for the archive path and `/usr/local/arm` for the installation path.

After the build of your toolchain is complete, you'll want to add `$INSTALLPATH/bin` to your `.bashrc` or equivalent.

```
export PATH=\$PATH:/usr/local/arm/bin
```

4.3.2 Kernel

Arm-Linux kernel version 2.4.19	Linux kernel patches for ARM processors
PXA Patches	Intel PXA support for ARM-Linux
Boundary Devices patches	Boundary Devices support

4.3.3 Libraries and Applications

Place links and references as in [this page](#) here.

April 5, 2005

5 Development Tools

5.1 minidebug

`minidebug` is a small (under 16k) debugger designed to fit completely within the instruction cache on the PXA-255 processor to allow testing of boards even in the absence of ROM or RAM.

It also includes features to download over either serial or Ethernet, allows the display and manipulation of registers and memory, and supports controlled execution through breakpoints and data watchpoints.

Upon entry, `minidebug` generally displays a dot (.) prompt, sometimes pre-pended by a string that looks like `$$00#b3`. Fear not. The `$$00#b3` string is used to allow `minidebug` to work in conjunction with the `gdb` debugger on the attached system.

The following is a list of commands that can be issued at the dot prompt. Note that this list can also be retrieved through `minidebug` by entering a question mark (?).

command	params	description
BC	address	Breakpoint clear
BE	address	Breakpoint examine
BS	address	Breakpoint Set
BURN	address range	Burn image at address range to flash
E	address	Examine and modify memory
D	address value	Deposit
DL	address	Start XModem for serial download
DLW	address	Download wireless
G	address	Go
GL		Go Linux
GG	address	Go no cache clear
R		Display content of registers
SSID		Set Wireless SSID string
T		Trace
TT		Trace no cache clear
V	address range	Verify content of flash
WC	address	Watch clear
WR	address	Watch read
WRW	address	Watch read/write
WW	address	Watch write
?		Show this list of commands

5.1.1 mdebug

The mdebug image adds Ethernet and wireless download capabilities using the Blast protocol to the Neon®. The SSID and DLW commands above are only valid when mdebug is present.

5.2 JTAG system-level debugger

The jtag executable provided by Boundary Devices is based on the one provided by the [Open WinCE](#) project.

Our main goals in developing the jtag program were to aid in hardware debugging and to allow the first flash EEPROM image to be burned onto a new device. That said, we also use it extensively as a terminal emulator during development and have added a number of extensions for that purpose.

The current release supports the PXA250, PXA255, and SA1100 (lart untested). It checks the IDCODE register and uses the appropriate BSDL structure.

5.2.1 Requirements

The jtag executable runs either under Linux or Cygwin.

Under Linux, there are no known dependencies except for libc and libstdc++.

Under Cygwin, the jtag executable requires the [ioperm](#) driver to be installed. This driver makes the ioperm() and iopl() system calls available under Windows for access to the serial and parallel ports. Note that after the cygwin package is installed, you still need to enable the driver through the use of the ioperm executable

For the cmd.exe inclined:

```
c:\> c:\cygwin\bin\ioperm.exe -v -i
```

or for the bash-inclined.

```
user@machine ~/u-boot-1.1.2
$ /bin/ioperm.exe -iv
```

Either way, the output should be something like the following.

```
Installing ioperm.sys...
OpenSCManager      ok
CreateService      ok
OpenService        ok
```

April 5, 2005

```
StartService      ok
ioperm.sys is already running.
```

5.2.2 Startup Options

`jtag -t` Generate a square wave on the processor pins.

This option allows pins to be checked in a sequence defined by the hardware file. A '+' or '-' keypress will scroll forward or backward through the list. Also, pin name can be entered directly. Entering GP0 will generate a square wave on GP0. A '?' will list matching pin names. Entering GP? will list all gpio pins.

`jtag -i` Identify the flash part used

This option tries to identify the part number of the Flash EEPROM. Currently supported parts are 28F160F3B, 28F320J3A, 28F128J3A, 28F320C3B, and 28F320S3, though not all have been tested. It should be relatively easy to add new parts.

`jtag -f` Generate the appropriate signals to program a flash.

This option is rarely used, since we normally program the flash through the minidebug software.

`jtag -c` Download code to the mini and main instruction cache.

This option is used to load a file into the instruction cache. Usually -x, -e, -or -d option is used to load minidebug. The -d option just loads minidebug. The -x option then proceed to dowload a file over the serial port using xmodem. The -e option dowloads a file using ethernet (wireless and wired support.) The -ssid option can be used to specify a wireless essid value to pass to minidebug.

`jtag -s` Terminal emulator option.

The parallel port is still searched because [Ctrl A] B can be used to send a JTAG break and attempt to return control to minidebug.

`jtag -N` Burn the entire flash.

This option can be used to burn a flash for the first time.

It first downloads the file mdebug to ram address A1800000.

Then it executes an ethernet download of the file totalflash.

If successful, it then burns the flash using the minidebug(mdebug) command BALL (burn all).

5.2.3 Control Keys

Once running, the jtag program responds to a number of command sequences, all beginning with [Ctrl A] .

[Ctrl A] B	Send a break
[Ctrl A] S	Send a file using XModem
[Ctrl A] T	Send an ascii file
[Ctrl A] P	Choose baud rate
[Ctrl A] Q	Quit

5.2.4 Blast protocol

When used with the mdebug image, the jtag program recognizes the start-of-download request sent by the device, and will prompt the user for a file name to send.

5.2.5 Quick-start download and burn

If you have a minidebug for your platform in the current working directory, the following sequence shows the process of using it to download and burn a new u-boot image.

Start debugger.

```
$ cd ..
$ ./jtag -d
ioport 3bc wrote 5d read ff
using printer port at 378

IDCODE: 69264013 - 0110 1001001001100100 00000001001 1
Halt released
Waiting for stub
LDIC finished

This uses the program minidebug on the arm to download to ram
using the serial port(xmodem protocol) or blast the file using
ethernet
^A Q for quit, ^A B external break, ^A S for sending a file with xmodem,
^A I for sending an RGB bitmap with xmodem, ^A P baudrate
^A T to send an ascii file

DBG-Vector Trap A0008000
R0: 00000000 R1: 0000014C R2: 00000000 R3: 00000003
R4: 0000001E R5: 81A0F288 R6: AAA00010 R7: 000ED784
R8: 00000000 R9: 81A18774 SL: AAA0001C FP: 81A1606C
IP: 80039094 SP: A0003400 LR: 8006C8CC PC: A0008000
CPSR 600000D3 FPO: 0000000000
```

To download using serial, use the 'dl address' command. Hit [Ctrl A] S to send the file (assumes u-boot.bin in the current directory). After issuing the DL command, the minidebug will begin sending C's. These are the start commands for XModem, and signal the readiness to receive a file. Use the [Ctrl A] S sequence to instruct jtag to prompt for and send a file using XModem.

April 5, 2005

To abort the operation, either when prompting for a filename or before, use [ctrl-C].

```
. dl a1f00000
CCCCCCCCCCCCC
enter binary file name: u-boot.bin
CCCCCCCCCCCCC.....
81292 bytes, 80 packets, 0 retries
OK A1F00000-A1F14000
```

To burn a range of data from RAM to the start of flash, use the 'burn' command like this. Note that the end address was given above at the end of the DL response.

```
. burn a1f00000 a1f14000
Sector 04000000 Erasing Programming Verifying...
Success
```

5.3 TeraTerm blast extensions

Describe TeraTerm extensions and the Blast[®] protocol here. Also add download [link](#).

6 Configuration Notes

6.1 LCD Panel configuration

The Neon[®] supports a variety of LCD panels. The following section describes the process of configuring the board for a known, currently supported display panel as well as a [Das U-Boot](#) utility command for testing settings on a new panel.

If you know the type of panel at compile time, you can place a selection from the list below in the [Das U-Boot](#) configuration file `include/configs/neon.h`. The `CONFIG_EXTRA_ENV_SETTINGS` macro is used to define a compile-time choice. If you are using EEPROM to store environment settings, these can be saved in the environment as well.

Name	Resolution	Description
hitachi_qvga	320 x 240	Hitachi Quarter VGA
sharp_qvga	320 x 240	Sharp Quarter VGA
hitachi_hvga	640 x 240	Hitachi Half VGA
sharp_vga	640 x 480	Sharp 10.4 inch VGA
hitachi_wvga	800 x 480	Hitachi Half VGA

For example:

```
#define CONFIG_EXTRA_ENV_SETTINGS "panel=hitachi_hvga" "\0"
```

Note that the `LCD_XRES` and `LCD_YRES` macros must also be defined because the boot loader needs to allocate memory for its' internal frame buffer prior to the use of the environment variable to initialize the LCD controller.

The boot loader settings for the LCD panel will carry through to the Linux driver.

Under Windows CE, is is also necessary to define an environment variable in the Platform Builder project. The following options are currently supported:

Name	Resolution	Description
BSP_DISPLAY_240X320	240 x 320	Hitachi Quarter VGA 3.5 inch
BSP_DISPLAY_320X240	320 x 240	Hitachi or Sharp Quarter VGA 5.7 inch
BSP_DISPLAY_640X240	640 x 240	Hitachi Half VGA
BSP_DISPLAY_800X480	800 x 480	Hitachi Half VGA

To select a choice, define the environment variable named in the list above and give it a value of 1.

If you're using the Neon[®] with a new panel, you'll need to determine and define the following fields for the panel.

field name	type	description
name	string	used to identify the panel
pixclock	number	Divisor for the pixel clock. Generally 3 for QVGA, 1 for higher resolution.
xres	number	Horizontal pixel count
yres	number	Vertical pixel count
act_high	number	Clock polarity, 0 (default) or 1
hsync_len	number	Horizontal sync pulse
left_margin	number	Idle pixels before leftmost pixel
right_margin	number	Idle pixels after rightmost pixel
vsync_len	number	Vertical sync pulse
upper_margin	number	Idle rows before topmost
lower_margin	number	Idle rows after bottom
active	number	Active Matrix (1) or Passive (0)

Once you have collected this information, a corresponding entry must be added to the list of panels.

```
u-boot-1.1.2/common/lcd_panels.c
```

To allow the testing of these settings and the use of a different display without re-compiling, the `lcdp` boot loader command is available. It may be used in one of the following ways:

command string	description
lcdp	Show the current lcd panel settings
lcdp ?	Show the list of currently supported lcd panels
lcdp panelname	Select and initialize panelname
lcdp +	Add a new panel (prompts for details)

Note that the boot loader text display will not be updated properly if the X and Y resolution don't match the current default display. Use the `bmp` commands to test the new panel configuration after using the `lcdp +` command string.

6.2 Memory size configuration

The Neon[®] supports either 32 or 64MB of RAM.

Most of the default boot loader configuration assumes at least 32MB of RAM is available. In particular, the `TEXT_BASE` variable in `board/neon/config.mk` links the `uboot.bin` image at 31MB from the start of RAM.

Use the `PHYS_SDRAM_1_SIZE` variable in `include/configs/neon.h` to specify the actual size for your hardware.

The Windows CE image supports either, but defaults to 32MB. Set the `RAM_SIZE_64 MB` environment variable in your project to indicate that 64MB should be present.

The RAM size set in the boot loader is passed to the Linux kernel.

6.3 Touch Panel Calibration

[Touch panel calibration notes go here.](#)