Operating Instruction Manual

netX Bootwizard

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC070502OI16EN | Revision 16 | English | 2013-12 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This operating instruction manual describes how to use the Hilscher netX Bootwizard application. This document is aimed at design-in OEM developers who want to create bootable images (i. e. firmware files) for the netX controller which can be started by the ROM loader of the netX.
This document is also aimed at all OEMs who need to download their own custom-made bootable firmware file or a Hilscher Second Stage Bootloader file to the Flash memory of a netX based hardware (design-in OEM netX device or Hilscher NXHX Development Board).

1.1.1 Reference to Software

This operation instruction manual refers to the Hilscher netX Bootwizard application version ≥ 1.3.15584.0.

1.1.2 List of Revisions

Rev	Date	Chapter	Revision
14	2013/03	all	Document completely revised
15	2013/04	4.2	note about parallel flash and SDRAM on netx 50/51/52/10
16	2013/12	8	added command "list_interfaces" to flash.bat
		5.2	Description of handling of “Flash Device Label” added
		11	Chapter “Flash Device Label” added

Table 1: List of Revisions

1.1.3 Terms, Abbreviations and Definitions

Term / Abbreviation	Description
boot image, bootable image	An executable memory image which can be recognized, loaded and started by the netX ROM code.
boot header, boot block	The 64 byte header at the beginning of a boot image which is used for identification. Allows the image to be recognized, loaded and started.
(serial) boot mode	A console mode for external configuration which the ROM code enters when it does not find a boot image or when signaled by external pins.
ELF	Executable and Linkable Format, standardized object file structure
GCC	GNU Compiler Collection
JTAG	Joint Test Action Group, a standard debugging interface
Aboot	netX 500 and netX 100 ROM code
Hboot	netX 50 ROM code
GUI	Graphical User Interface
OEM	Original Equipment Manufacturer
NXF	Format of a Hilscher standard loadable firmware file
LFW	Hilscher standard loadable firmware
Flash Device Label	A data structure in the flash memory of some devices which contains device identification information and other information about the device.

Table 2: Terms, Abbreviations and Definitions

1.1.4 Conventions in this Manual

Notes, operation instructions and results of operation steps are marked as follows:

Notes

Important: <important note>

Note: <note>

<note, where to find further information>

Operation Instructions

1. <instruction>
2. <instruction>

or

➢ <instruction>

Results

<result>

1.2 Legal Notes

1.2.1 Copyright

1.2.2 Important Notes

The user manual, accompanying texts and the documentation were created for the use of the products by qualified experts, however, errors cannot be ruled out. For this reason, no guarantee can be made and neither juristic responsibility for erroneous information nor any liability can be assumed. Descriptions, accompanying texts and documentation included in the user manual do not present a guarantee nor any information about proper use as stipulated in the contract or a warranted feature. It cannot be ruled out that the user manual, the accompanying texts and the documentation do not correspond exactly to the described features, standards or other data of the delivered product. No warranty or guarantee regarding the correctness or accuracy of the information is assumed.

We reserve the right to change our products and their specification as well as related user manuals, accompanying texts and documentation at all times and without advance notice, without obligation to report the change. Changes will be included in future manuals and do not constitute any obligations. There is no entitlement to revisions of delivered documents. The manual delivered with the product applies.

Hilscher Gesellschaft für Systemautomation mbH is not liable under any circumstances for direct, indirect, incidental or follow-on damage or loss of earnings resulting from the use of the information contained in this publication.

1.2.3 Exclusion of Liability

The software was produced and tested with utmost care by Hilscher Gesellschaft für Systemautomation mbH and is made available as is. No warranty can be assumed for the performance and flawlessness of the software for all usage conditions and cases and for the results produced when utilized by the user. Liability for any damages that may result from the use of the hardware or software or related documents, is limited to cases of intent or grossly negligent violation of significant contractual obligations. Indemnity claims for the violation of significant contractual obligations are limited to damages that are foreseeable and typical for this type of contract.

It is strictly prohibited to use the software in the following areas:

- for military purposes or in weapon systems;
- for the design, construction, maintenance or operation of nuclear facilities;
- in air traffic control systems, air traffic or air traffic communication systems;
- in life support systems;
- in systems in which failures in the software could lead to personal injury or injuries leading to death.

We inform you that the software was not developed for use in dangerous environments requiring fail-proof control mechanisms. Use of the software in such an environment occurs at your own risk. No liability is assumed for damages or losses due to unauthorized use.

1.2.4 Warranty

Although the hardware and software was developed with utmost care and tested intensively, Hilscher Gesellschaft für Systemautomation mbH does not guarantee its suitability for any purpose not confirmed in writing. It cannot be guaranteed that the hardware and software will meet your requirements, that the use of the software operates without interruption and that the software is free of errors. No guarantee is made regarding infringements, violations of patents, rights of ownership or the freedom from interference by third parties. No additional guarantees or assurances are made regarding marketability, freedom of defect of title, integration or usability for certain purposes unless they are required in accordance with the law and cannot be limited. Warranty claims are limited to the right to claim rectification.

1.2.5 Export Regulations

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different counters, in particular those of Germany and the USA. The software may not be exported to countries where this is prohibited by the United States Export Administration Act and its additional provisions. You are obligated to comply with the regulations at your personal responsibility. We wish to inform you that you may require permission from state authorities to export, re-export or import the product.

2 Descriptions and Requirements

2.1 Features of the Bootwizard

The Bootwizard is a software application for constructing bootable software images (i. e. firmware files) for the netX controller which can be started by the ROM loader of the netX. The Bootwizard can also be used to write files (i. e. customized bootable firmware or Hilscher Second Stage Bootloader) to flash devices connected to netX based hardware via serial, USB or JTAG interface. In case of netX 51/52 controllers, files can also be flashed via Ethernet interface.
Since version 1.3.15584.0, the Bootwizard supports so-called “flash device labels”. This means that the Bootwizard is capable of recognizing protected areas in a flash memory device that are marked as a “flash device label” and performs its flasher tasks (like e. g. erasing or writing to flash) without violating these protected areas in the target device.

In detail, you can use the Bootwizard to

- generate bootable binary images from ELF files or raw binaries on your PC (a compiler/linker toolchain for ARM – e. g. GNU ARM toolchain – needs to be installed separately)
- modify bootable images on your PC
- perform a basic check on a bootable image on your PC
- flash any file from your PC to serial/parallel flash on a netX device
- compare the contents of flash memory on a netX device to a file on your PC
- save the contents from flash memory on a netX device to a file on your PC
- erase flash memory on a netX device.

The functions listed above can be accessed and performed in the graphical user interface (GUI) of the Bootwizard.

The Bootwizard installation also includes the **bootblocker.bat** and **flash.bat** scripts, which allow you to execute these functions from a command line without having to use the GUI. In addition to the functions of the Bootwizard, the **bootblocker.bat** script allows you to

- update checksums in NXF files
  (i. e. Hilscher standard loadable firmware files)
- include tag lists into NXF files.

For more detailed information about the **bootblocker.bat** and **flash.bat** scripts, see chapter *Using the Command Line Bootblocker Script* on page 47 and chapter *Using the Command Line Flash Script* on page 53.

2.2 Limitations

- The Bootwizard application does not recognize the common header V3 and tag lists in NXF firmware files.

Note: The **bootblocker.bat** command line tool, however, is able to insert tag lists and update the checksums of an NXF file. Use the bootblocker tool to build NXF files or to update the checksums after modifying an NXF file using the Bootwizard. For more information, please refer to chapter *Using the Command Line Bootblocker Script* on page 47.

- The flasher does not support parallel flash on netX 51/52 controllers.
- The netX51 step A cannot be flashed via the serial port.

2.3 Requirements

- netX based hardware (design-in OEM netX device or Hilscher NXHX Board) with a USB, RS232 or JTAG interface
- PC running under Windows XP, Service Pack 2 (32 Bit) or Windows 7 (32/64 Bit) with installed GNU ARM toolchain (for generating bootable images)

Note: Support for the "*Hitex GNU Compiler for ARM*" is pre-configured. The toolchain can be downloaded from www.hitex.com.

- For connecting PC to netX device via USB: libusbX/WinUSB and USB CDC driver (the Windows versions of these drivers are included in the Bootwizard installation program)
- For connecting PC to netX device via JTAG interface:
  either use a NXHX Board with onboard USB JTAG or use Amontec JTAGkey for design-in OEM netX devices

2.4 Description of the Data Flow

2.4.1 ROM Loader, Bootable Image and Bootwizard

After power-on reset at a netX device, the **ROM Loader** (which is a hardware function and always present in the netX) is started. The ROM loader initializes the netX controller and its optional non-volatile boot devices such as serial or parallel Flash etc. It then reads the available bootable image (i. e. the firmware or the Second Stage Bootloader) from the Flash memory or receives it via Dual-Port Memory. After this, the image is written to the SDRAM (or the internal RAM of the netX) and executed.

In order to load and start a program/firmware, the netX ROM loader code needs:

- the program as a single continuous memory image
- magic numbers and checksums to recognize a valid boot image
- parameters to configure the memory
- the load address to which the program is copied (unless it is run in flash)
- the entry address at which the program is started

With the Bootwizard, you can construct bootable images that fulfill these conditions out of ELF or raw binary files.

2.4.2 Creating Bootable Images

Creating Bootable Images from ELF Files

The Bootwizard (or the Bootblocker.bat) itself is not able to extract the memory image from an ELF input file. It simply passes the input file to a script contained in an XML file. The XML file is named netx.xml and is included in the Bootwizard installation (default directory: `Hilscher GmbH\Bootwizard\bootwizard`). The netx.xml file contains preconfigured scripts for the following kinds of toolchains:

- Codesourcery 4.5.2
- HiTex GNU Tools 4.00
- non-HiTex GCC under Windows
- Linux gcc-arm-elf

These toolchains can be selected in the Bootwizard GUI or in the Bootblocker.bat. The corresponding script in the netX.xml file then calls the binary utilities included with the toolchain to extract all necessary information from the ELF file.

Note: If your toolchain is not among the preconfigured toolchains listed above, you need to adapt a script in the netx.xml file. For more information on this, see *Editing the netx.xml File* on page 56.

So, for example, if you are developing software using GCC for ARM, the Linker will generate an ELF file. If you then open the Bootwizard and select the toolchain which has been used to generate the ELF file, e. g. HiTex GNU, the script for the HiTex GNU Tools in the netx.xml file expects that the environment variable "PATH_GNU_ARM" points to the GCC installation and uses the tools included with GCC to convert the ELF file to a memory image and to extract the load and start addresses. The Bootwizard then puts a boot header (containing additional information about the image and the memory configuration) in front of this image.

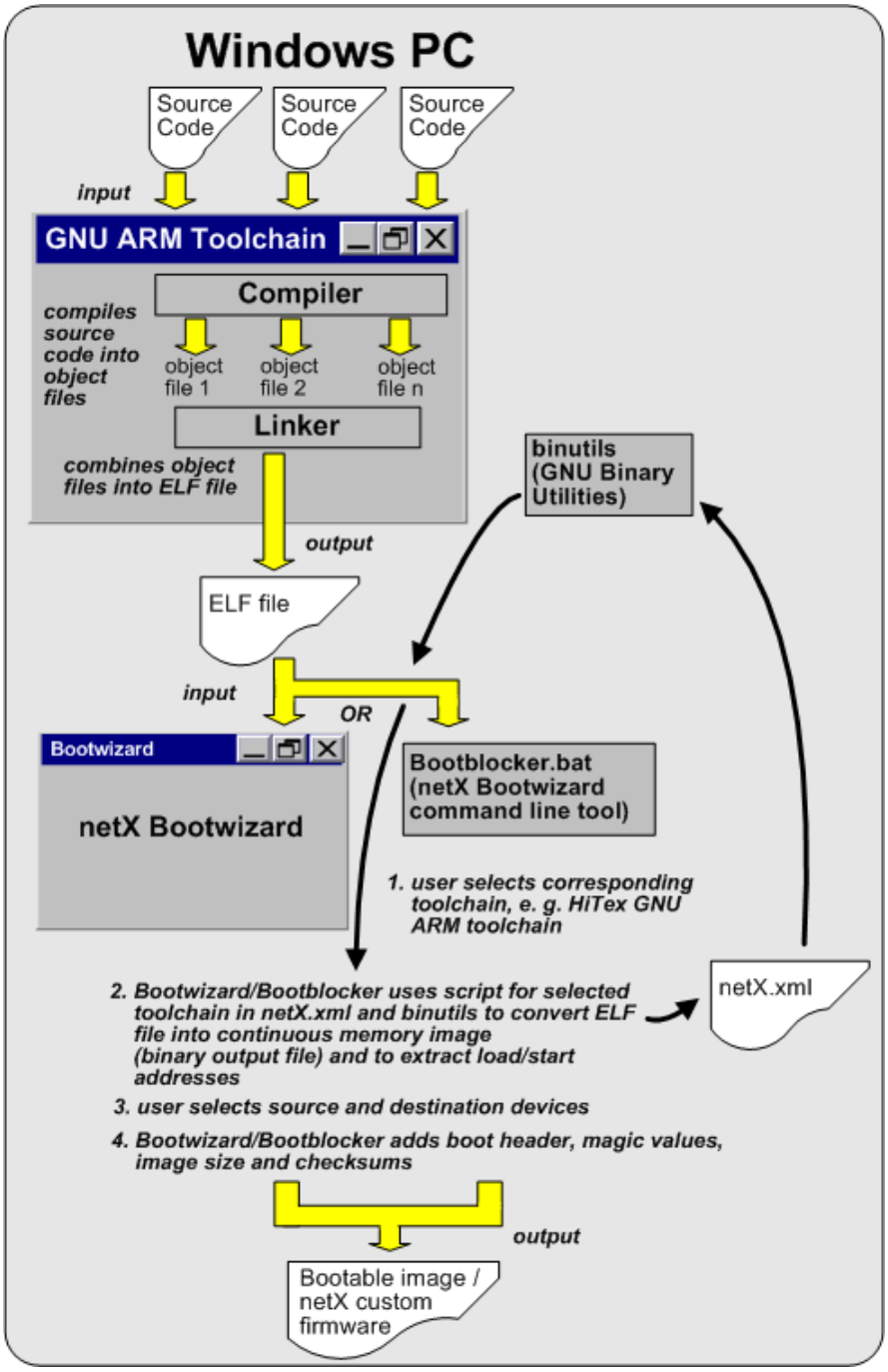

Figure 1: Creating Bootable Image from ELF file and GNU ARM Toolchain

Creating Bootable Images from Raw Binary Files

In case a continuous memory image already exists as “raw binary” file, the corresponding toolchain does not need to be installed on your PC and also does not need to be specified in the Bootwizard/Bootblocker or the netx.xml file.
However, the load and start addresses in the boot header of the bootable image must then be entered manually.

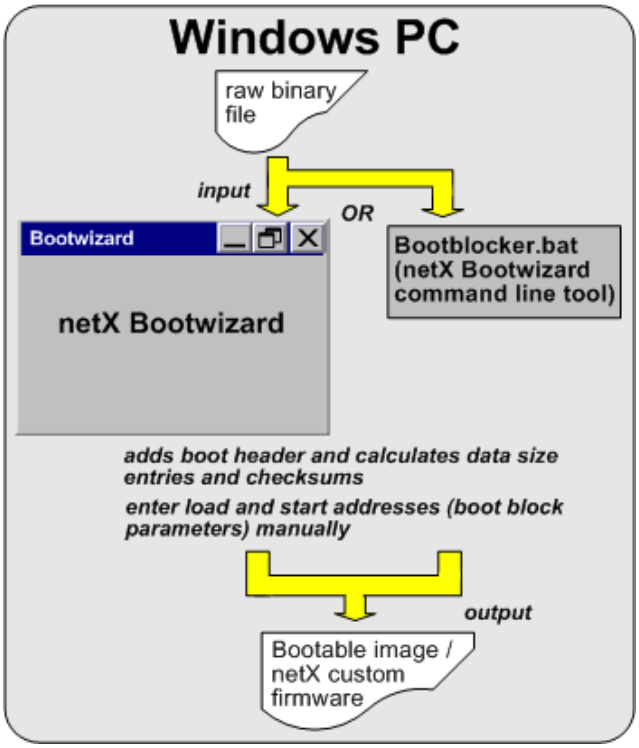

Figure 2: Creating Bootable Image from Raw Binary File

Creating NXF Files from ELF Files

With the Bootblocker.bat command line tool, you can also create firmware files in the NXF format.

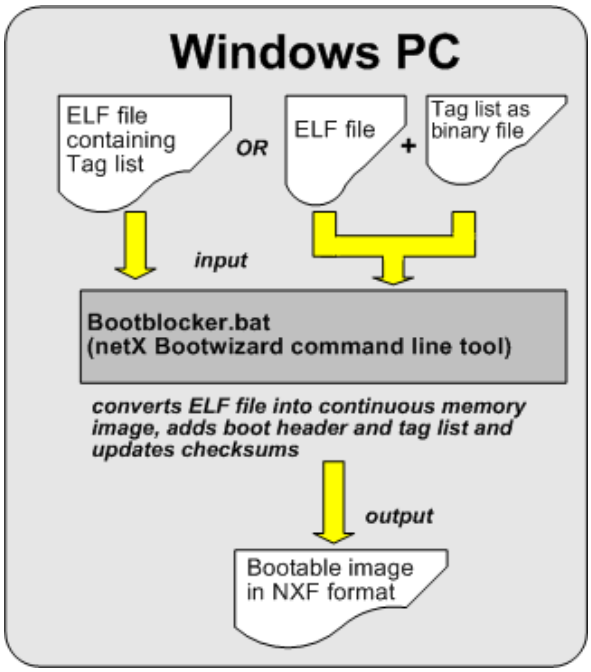

Figure 3: Creating NXF File from ELF File

For more information on this, please refer to section *Building NXF Files With Common Header V3/Tag List* on page 51.

2.4.3 Editing Bootable Images

Editing Boot Block Parameters of Bootable Images

The **Modify image** function of the Bootwizard and the Bootblocker.bat allows OEMs to edit the boot block parameters (load address, entry point, user data, source and destination device) contained in the boot header of an existing bootable image.

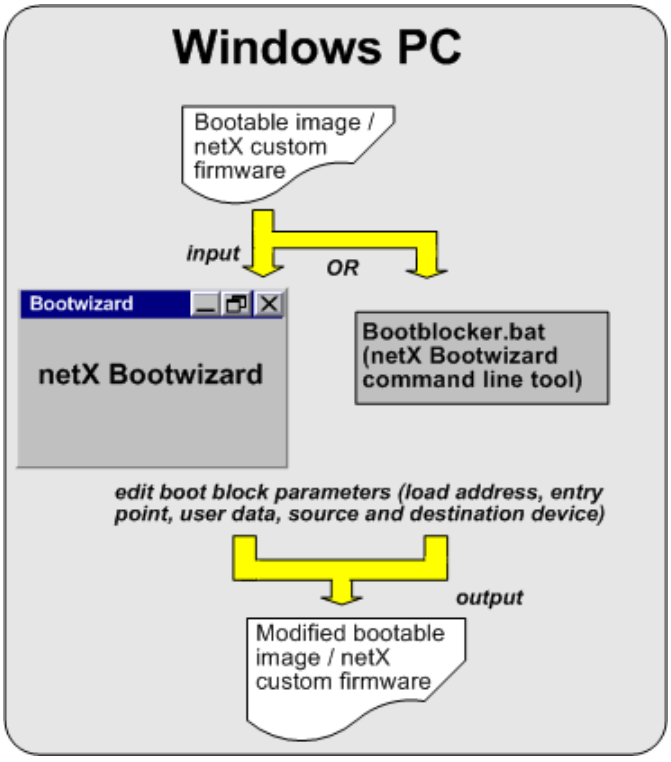

Figure 4: Editing Bootable Image

Editing Boot Block Parameters of NXF Files

OEMs using Hilscher standard loadable firmware in NXF format can use the Bootwizard or the Bootblocker.bat to edit the boot block parameters of the NXF file.

Note: In a netX device, NXF firmware is started by a special software module called **Second Stage Bootloader**, and SDRAM parameters are usually provided by the **Security Memory** (Sec Mem).
If no Sec Mem is present on the device (i. e. on slave devices), SDRAM parameters will be read from the tag list of the Second Stage Bootloader. Only if the Second Stage Bootloader is not capable of providing these parameters will they be taken from the boot header of the NXF file.

Note that if the **Bootwizard** instead of the **Bootblocker.bat** has been used to change the NXF file, the checksums in the so-called “common header” of the NXF file will not be correct any longer. The checksums then must be updated with the **Bootblocker.bat** command line tool or the Hilscher **Tag List Editor**.

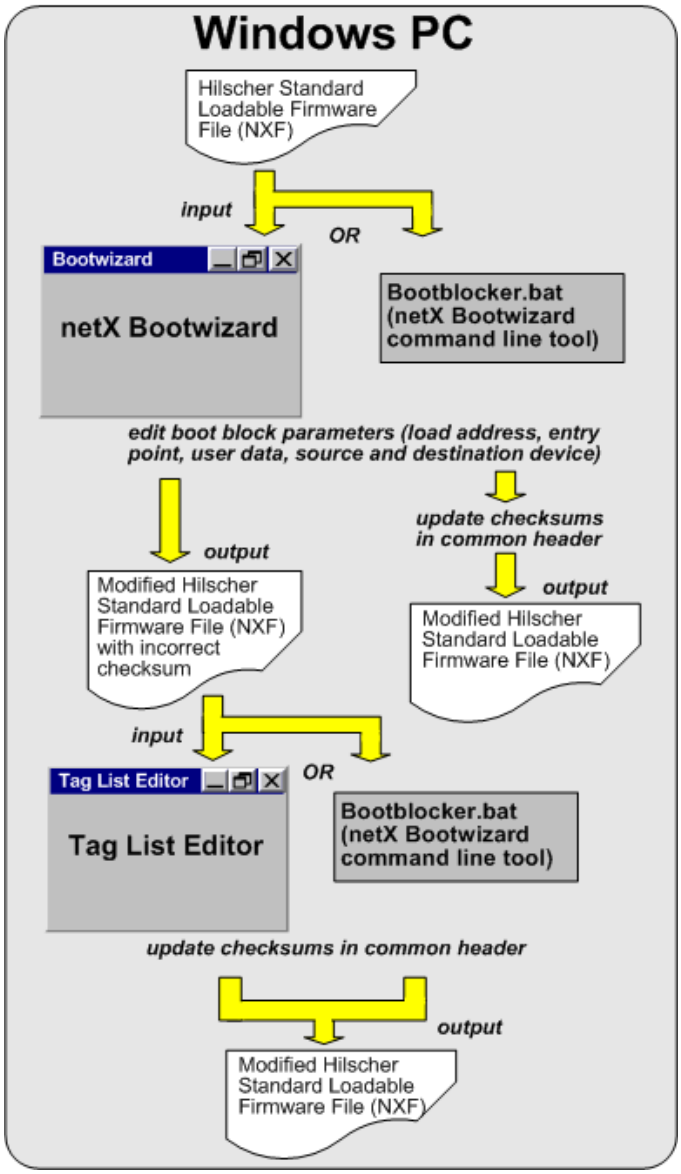

Figure 5: Editing NXF File

2.4.4 Checking Bootable Images

The validity (magic cookie, netX signature, boot block checksum, application checksum, application size in words) of existing bootable images can be checked with the Bootwizard or the Bootblocker.bat.

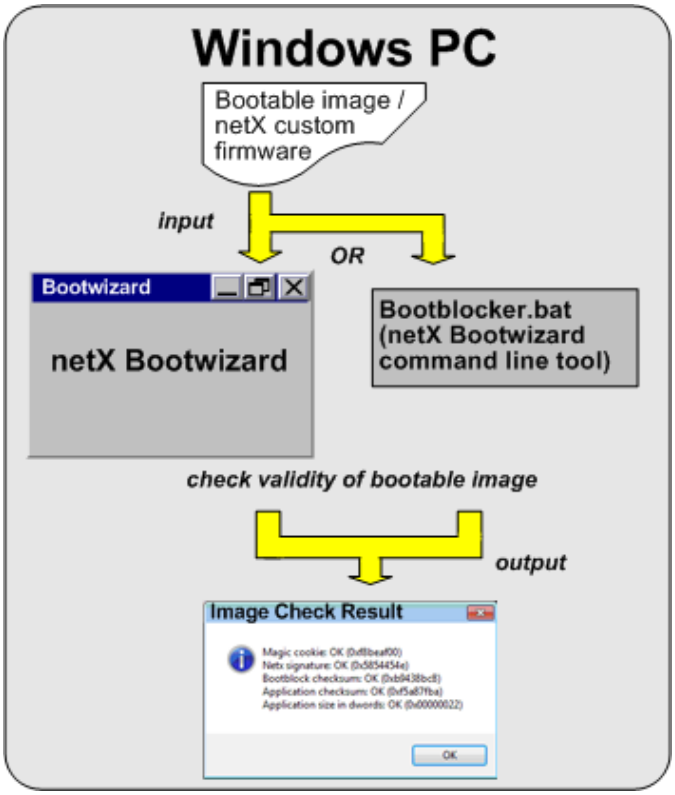

Figure 6: Checking Bootable Image

2.4.5 Downloading Bootable Images to Flash Memory

The Bootwizard and the Bootblocker.bat can be used to download a bootable image for a netX controller from a PC to a flash memory connected to the netX. This can be a custom-made bootable image / firmware or the Second Stage Bootloader, which is needed to start-up standard Hilscher Loadable Firmware in NXF format. The flasher functions are not limited to bootable images, the flasher can also write and read arbitrary data.

Note: Since version 1.3.15584.0, the Bootwizard detects Flash Device Labels and will protect them from being overwritten. For more information, see *Flash Device Label* chapter on page 66.

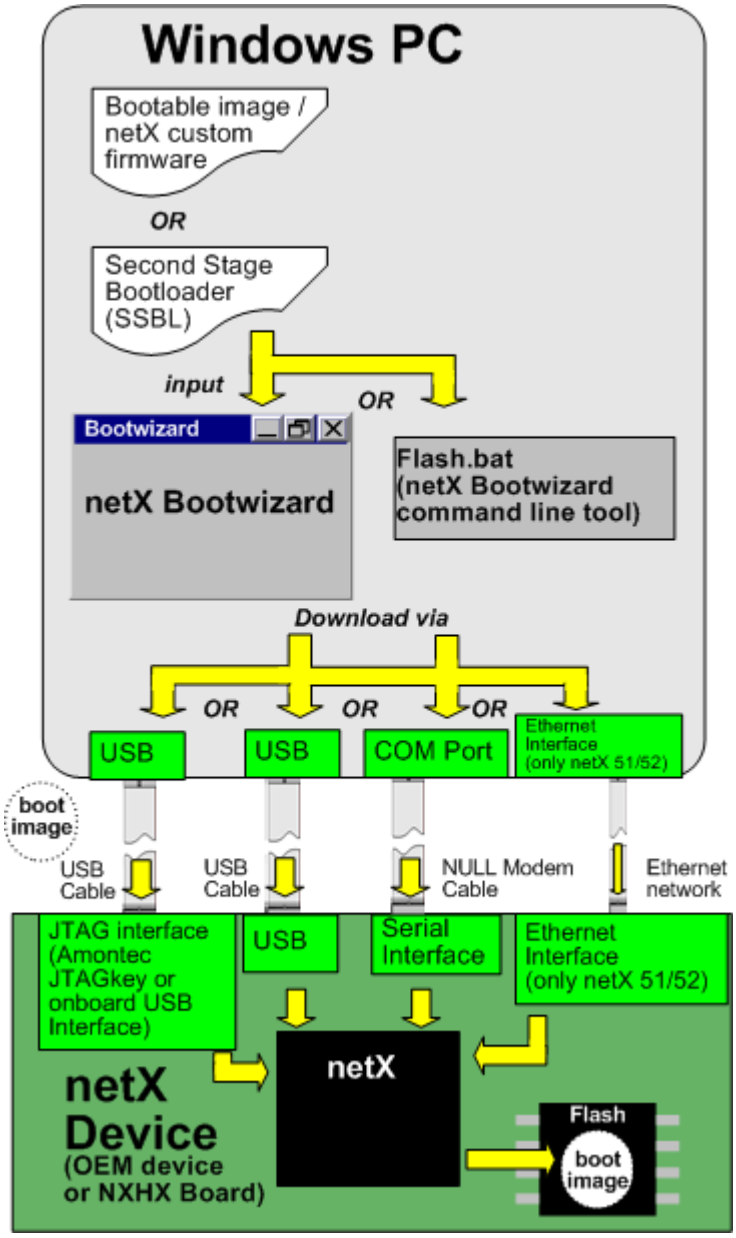

Figure 7: Downloading Bootable Image

3 Installation

3.1 Bootwizard Installation

In order to install the Bootwizard and/or the drivers under Windows XP, you must be logged in as an administrator. In order to install the Bootwizard under Windows 7, you must enter an administrator password at the UAC prompt. After installation, the application will run from a limited user account.

3.1.1 Step-By-Step Instructions for Installing the Bootwizard

Note: The latest software and driver versions can always be obtained from http://www.hilscher.com/.

The installer is a single executable file. Run it and follow the instructions:

1. On the welcome page, click on **Next**.
2. Read the license agreement. If you accept it, select **I accept the agreement** and click on **Next**. Otherwise, click on **Cancel** to abort the installation.
3. Select a directory for the installation, or accept the predefined one. Then, click on **Next**.
4. Select the components to install. You can select and de-select each component, or choose one of the pre-defined installations using the combo box at the top. Normally, you should select the standard installation.
5. On the same page, select the drivers you need to install. Please note the following:
   - You only need these drivers if you want to flash the netX via USB.
   - For netX 500, 100 and 10, install the libusbx/WinUSB driver.
     For netX 50, install the USB serial emulation driver.
     For netX 51 and 52, install both drivers.
   - If you update from a version prior to 1.3.x.x, install the libusbx/WinUSB driver.
   - Regardless of whether you select any drivers, the driver files can always be found in the Bootwizard installation directory, subdirectory `driver`, and installed manually.
6. On the same page, there is an option to install ftd2xx.dll in the application directory. It is selected by default. If you encounter problems flashing via JTAG and ftd2xx.dll is present in the system32 directory, try installing the Bootwizard with this option disabled.
7. Click on **Next** to continue.
8. Decide whether you want to create a Start menu entry, and name it. Click on **Next** to continue.
9. On the next page, you can define whether you want to create a Start menu icon, a desktop icon and a quick launch icon.

On the same page, specify if the environment variable PATH_BOOTWIZARD should be defined. This option is enabled by default since the variable is required by the bootblocker and the command line flasher.

Click on **Next** to continue.

10. You are now ready to install. Review the installation settings on this page and click **Install** to proceed or **Back** to go back and change the installation settings.
11. If you have selected any drivers in step 5, they are installed. If any matching netX devices are connected via USB during the installation, the respective drivers will be installed for these devices. Otherwise, they will be installed later when you connect the device.

    Windows 7: If you have selected drivers in step 5, you will be asked if you want to install them: **Would you like to install this device software?**. Click on **Install**.
12. After installing, the change log is displayed. Click on **Next**.
13. Click on **Finish** to exit the installer.
14. If you have selected drivers in step 5, connect the respective netX devices to allow Windows to recognize them and to install the drivers.
    - Disconnect all netX devices from all USB ports.
    - Put the netX device into serial boot mode.
      (Consult your device documentation to find out how to do so.)
    - Connect the device to a USB port.

    On Windows XP, the **Found New Hardware Wizard** appears.
    - Select **No, not this time** and click on **Next**.
    - On the next page, select **Install the software automatically** and click on **Next**. The driver will be installed automatically.

    On Windows 7, the driver will be installed automatically.

Note: If you connect to a netX via UART or USB, a new monitor program with faster transfer routines is downloaded to the internal RAM.

On the netX 500/100 and netX10, the new monitor routine will set a new USB ID. If you unplug and re-plug the USB cable without resetting the netX, the PC will recognize a new USB device and the driver must be installed once more.

3.1.2 JTAG Driver Installation

The Bootwizard package does not include any USB-JTAG drivers.

JTAG drivers for NXHX Boards are included with HITOP.

Drivers for the Amontec JTAGkey are available from the company's website: http://www.amontec.com/jtagkey.shtml

3.1.3 Configuration files

The Bootwizard uses two configuration files. One is installed in the application directory and contains a default configuration. The other one contains any changes you have made and is written to your user directory every time you close the Bootwizard. The settings stored in the personal configuration override those in the default configuration.

Default configuration:
`<install directory>\application\Bootwizard.cfg`
e.g.:
`C:\Program Files\Hilscher GmbH\Bootwizard\application\Bootwizard.cfg`

User configuration:
`<User local application data directory>\Bootwizard.cfg`
e.g.:
`C:\Documents and Settings\<user name>\Local Settings\Application Data\Bootwizard.cfg`

The user configuration is not deleted when you uninstall the Bootwizard. If you encounter any problems after an update, delete this file manually. The file is located in a hidden directory; in order to make it visible, open the Explorer's folder options and select **Show hidden files and folders**, or press **Win** + **R** and enter the command **shell:Local AppData**.

3.1.4 Files and Directories

The following table shows the included files and directories:

File/Directory	Description
bootwizard.lnk	Click on this to run the Bootwizard
application\	The platform on which the Bootwizard is running
muhkuh.exe, serverkuh.exe	The platform server.
lua.exe	Command-line Lua interpreter
*.dll	Lua, wxWidgets and wxLua libraries
lua, lua_hilscher	General Lua scripts
plugins\	Communication plugins
Bootwizard.cfg	Default configuration
<user local application data>\Bootwizard.cfg	User-specific configuration
bootwizard\	The bootwizard, a script running on the platform
*.lua	Bootwizard / Lua scripts
test_description.xml	Refers to the platform LUA scripts
flasher*.bin	Flasher binaries
netx.xml	Scripts to extract binary and addresses from ELF files Device description and parameter
bootblocker.bat	Batch file to call bootblocker.wx.lua
bootblocker.wx.lua	Command line tool to generate boot images
flash.bat	Command line flasher
autoflash.xml	Reference to autoflash.lua
autoflash.lua	Lua script to run flasher from the command line
crypto*.lua, crypto*.bin	Read SDRAM parameters from security memory
docs\	Directory with main documentation and additional readmes
docs\licenses.txt	Licenses of the components of the software
docs\changelog.txt	List of changes
docs\Bootwizard.pdf	Bootwizard documentation (this document)
driver\	Windows XP/Windows 7 USB drivers for netX
winusb	WinUSB driver for netX 500, 100, 50, 51, 52 and10
vcp	USB serial port driver for netX 50, 51 and 52. This is only an INF file which associates the virtual COM port via USB with the Windows usbser.sys driver.
dpinst.exe	Driver installer
unins000.exe, unins000.dat	The uninstaller

Table 3: Bootwizard File List

4 Description of the Graphical User Interface

The image below shows the Bootwizard's GUI. While using the Bootwizard, you will usually see only a subset of the fields and controls shown below, because the GUI displays only the elements relevant to the currently selected task.

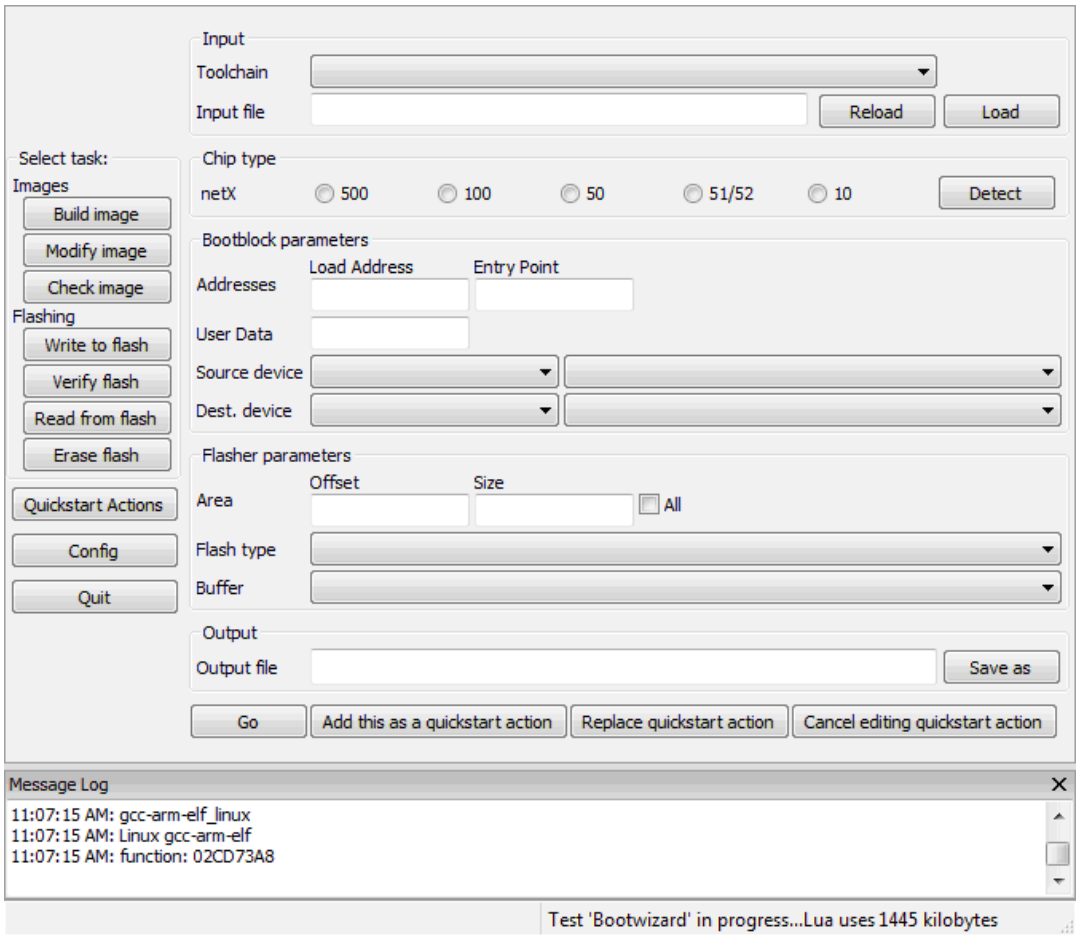

Figure 8: All Elements of the Bootwizard GUI

4.1 Workflow in the Bootwizard GUI

First, use one of the buttons in the **Select task** area on the left hand side of the GUI to choose a task:

- **build**, **modify** or **check** bootable images or
- **write**, **verify**, **read** or **erase** flash memory.

The GUI on the right hand side will be configured to display the elements which are relevant to the task, and the **Go** button at the bottom is renamed according to the task.
The **Go** button is disabled until all necessary selections have been made. Click the **Go** button to perform the task (e.g. build and save the image, or start flashing). The three other buttons are only relevant to quick start actions.

4.2 Elements in the Bootwizard GUI

The right-hand side of the GUI consists of the following elements:

Input: Here you specify the ELF file, raw binary file or boot image to work on. If you intend to download a file to the Flash memory of a netX device, you also can select the file here.

- **Toolchain**: If you want to create a bootable image from an ELF file, you first have to select one of the extraction scripts specified in netx.xml.
- Press the **Load** button to load an ELF file, a raw binary, or a bootable image. The input file field can also be edited by hand. Press <enter> to load the file.
- Use the **Reload** button to re-load the selected input file, if the file has changed.

Chip type: You have to select the type of netX chip you are using before you can make any further selections, because the boot block and flasher parameters depend on the chip type. Either select the chip type manually or use the **Detect** button to connect to the device and detect the chip type.

Bootblock parameters: These parameters will be stored in the bootblock.

- **Addresses (Load Address/Entry Point)**: When the image is booted, the program binary will be copied to the load address and started at the entry point. If you load an ELF file, the addresses are set automatically. If you load a raw binary, you have to enter the addresses manually.
- **User Data**: This field is ignored by the ROM loader. It is a possible place to store a serial number. You can enter any 32 bit value, which will be stored at position 0x34 (ulSerial) in the bootheader. If you leave this field blank, the value is set to zero.
- **Source device**: this is the memory device from which the image is read at boot time, i.e. non-volatile memory. To select a device, first make sure that the netX type is set. Choose the type of device (SPI serial flash, parallel flash etc.) from the left combo box. Then, select the device from the right combo box.
- **Destination device**: this is the device to which the executable binary is copied before running it. When you load an ELF file, the destination device type (internal RAM, SDRAM or external SRAM) will be suggested according to the load address.

Flasher parameters: These parameters are relevant to the flasher.

- **Area Offset**: The starting position of a flash operation in bytes as a decimal or hexadecimal number.
- **Area Size**: The size for a flash operation in bytes as a decimal or hexadecimal number. For write and verify, the length is set depending on the input file.
- **Area All**: Read or erase the whole flash. This option overrides the offset and length fields.

- **Flash type**: the type of flash device (serial/parallel) you want to access. If you are flashing a bootable image, it is set automatically according to the source device type stored in the boot header. However, you may set the flash type manually, which is necessary if you want to flash a file without boot header. You do not have to specify the exact device, the flasher will auto-detect it.
- **Buffer**: When writing to, verifying or reading from the flash memory, the data is buffered in the internal RAM or external SDRAM.
  By default, data is buffered in internal RAM, as it is available on all netX chips and does not require any configuration. However, large images are processed in chunks, which may slow down the process.
  Alternatively, you may select an SDRAM configuration from the destination devices in netx.xml or "Auto". If "Auto" is selected and the size of the data to be written to or read from the flash is much larger than the buffer in internal RAM, the security memory will be checked for SDRAM parameters. If found, the SDRAM will be configured and used as a buffer.

Important: If you select the wrong type of SDRAM, or "Auto" and the security memory contains incorrect SDRAM parameters, errors will occur.

Important: On the netX 50/51/52/10, it depends on the hardware design whether SDRAM and parallel flash can be used simultaneously.

When a boot image for the netX 50/51/52/10 is loaded which has parallel flash as the source device, the buffer device is set to internal RAM.

Output: These fields will be available only if you build or modify an image, or if you read from the flash memory of the netX device.

- **Output file**: If you are creating or modifying a bootable image, the output file is a bootable image. If you are reading from a flash device, it is a binary file.
- Click on **Save as** to choose the output file, or enter the file name manually.

5 Bootwizard Use Cases

5.1 Boot Image Functions

The boot image functions of the Bootwizard are relevant to design-in OEMs who want to create their own custom bootable firmware for a netX device from an ELF file or a raw application binary. In addition to this, the **Modify image** function of the Bootwizard allows OEMs to edit the boot block parameters (load address, entry point, user data, source and destination device) contained in the boot header of an already existing bootable image. The validity (presence of cookies and signatures, correct checksums) of existing bootable images can also be checked here.

OEMs using Hilscher standard loadable firmware in NXF format can use the Bootwizard to edit the boot block parameters of an NXF file.
Note, however, that NXF firmware is started by a special software module called **Second Stage Bootloader**, and SDRAM parameters are usually provided by the **Security Memory** (Sec Mem). If no Sec Mem is present on the device (i. e. on slave devices), SDRAM parameters will be read from the tag list of the Second Stage Bootloader. Only if the Second Stage Bootloader is not capable of providing these parameters will they be taken from the boot header of the NXF file.
Note also, that the checksums in the NXF common header will not be correct after you have used the Bootwizard to change the NXF file. In this case, use the **bootblocker.bat** command line tool (see section *Updating Checksums in NXF files* on page 52) or the Hilscher Tag List Editor (see Operating Instruction Manual *Tag List Editor*) to update the checksums.

5.1.1 Build a Bootable Image

This section describes how to create a bootable image from an ELF file or a raw application binary. For a more detailed description of the parameters and elements which can be set here, please refer to the *Elements in the Bootwizard GUI* section on page 23.

Prerequisites

If you want build a bootable image from an ELF file:

- ELF file
- The compiler/linker toolchain for ARM that was used to build the ELF file is installed on your PC.
- If other than a preconfigured toolchain is to be used (see section *Creating Bootable Images* on page 10):
  The script in the netx.xml file has been adapted to the toolchain.
  For more information, see *Editing the netx.xml File* chapter on page 56.
- The netX Bootwizard application is installed on your PC.

If you want build a bootable image from a raw binary file:

- Raw binary file
- The netX Bootwizard application is installed on your PC.

Step-by-step instructions

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

↬ The Bootwizard opens.

2. In the **Select task** area, click on **Build image** button.

↬ The GUI will look as follows:

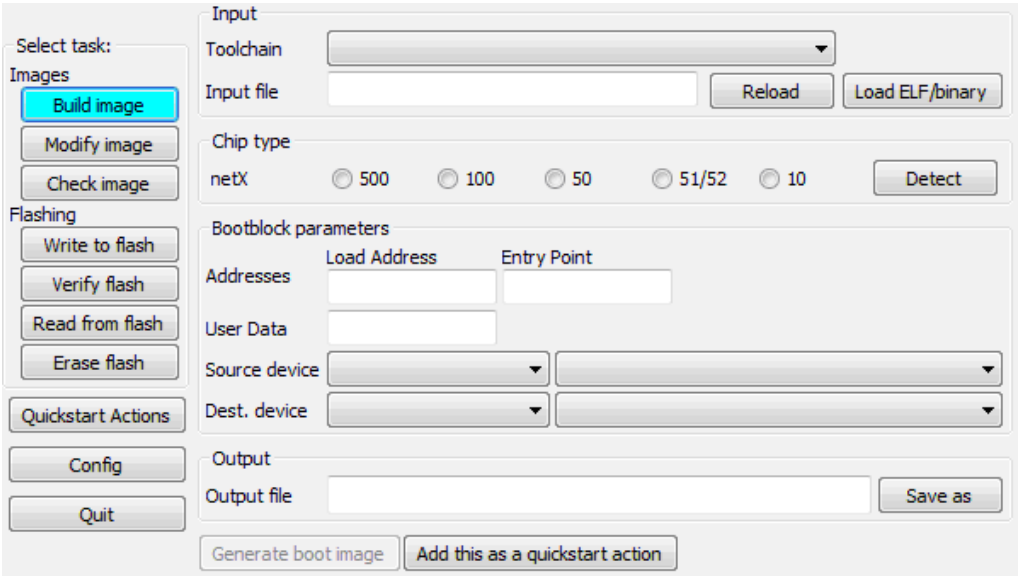

Figure 9: The "Create Bootimage" GUI

3. In the **Input** area, select the **Toolchain** (if you want build a bootable image from an ELF file).
4. Load the input file.

➢ Click on **Load ELF/binary** button and choose an ELF file or a raw binary.

5. Select the **Chip type**.
6. If you have loaded a raw binary file, enter the **Load Address** and the **Entry Point** in the **Bootblock parameters** section. (If you have loaded an ELF file, these parameters are preset by the values taken from the ELF file.)
7. In the **Bootblock parameters** section, select the **Source device** and the **Destination device** according to your hardware.
8. Specify the output file.

➢ In the **Output** section, click on **Save as** button to specify name and storage location of the output file in your file system.

➮ The GUI should now look similar to the following image, and the **Generate boot image** button should be enabled.

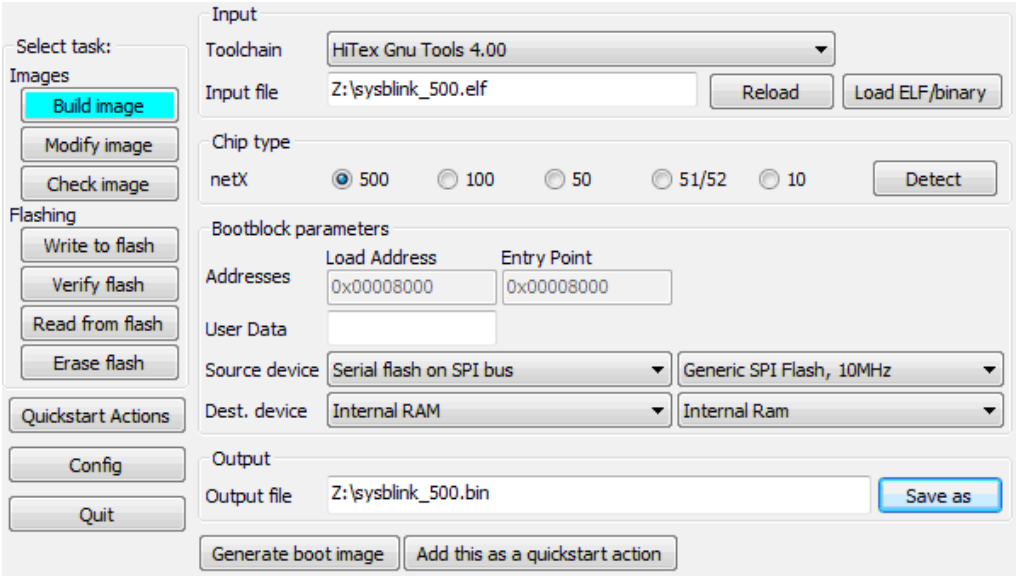

Figure 10: The "Create Bootimage" GUI after all selections have been made

9. Click on **Generate boot image**.

➮ The bootable image is created and written to the specified output file.

5.1.2 Modify a Bootable Image

This section describes how to make changes to the boot block parameters of an existing boot image or Hilscher NXF file.

Important: If you use the Bootwizard to modify the boot block parameters in the boot header of an NXF file, the checksums in the Common Header of the NXF file will be incorrect. In this case, use the Bootblocker command line tool to update the checksums (see chapter *Using the Command Line Bootblocker Script* on page 47). As an alternative, you can also use the Hilscher Tag List Editor to update the checksums (for details, please refer to the Operating Instruction Manual *Tag List Editor*).

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

↬ The Bootwizard opens.

2. Click on the **Modify image** task button.
3. Click on **Load boot image** button and select a bootable image.

Note: After loading a boot image, the devices shown in the **Bootblock parameters** may differ from those selected when the image was created. The Bootwizard tries to set the chip type, source and destination device automatically by matching the parameters in the boot header against the device parameters in netx.xml. This detection may return incorrect results, since the parameters are not unique between devices. Normally, this is not a problem, since the recognized devices have the same parameters. If you want to change the source or destination device, make sure the correct chip type is selected. The auto detection may also fail altogether, in which case you will have to set the chip type and source/destination devices manually.

4. Make the necessary changes to the **Bootblock parameters**.
5. Select an output file. This may be the same as the input file, or a different one.
6. Click on **Write modified boot image** button.

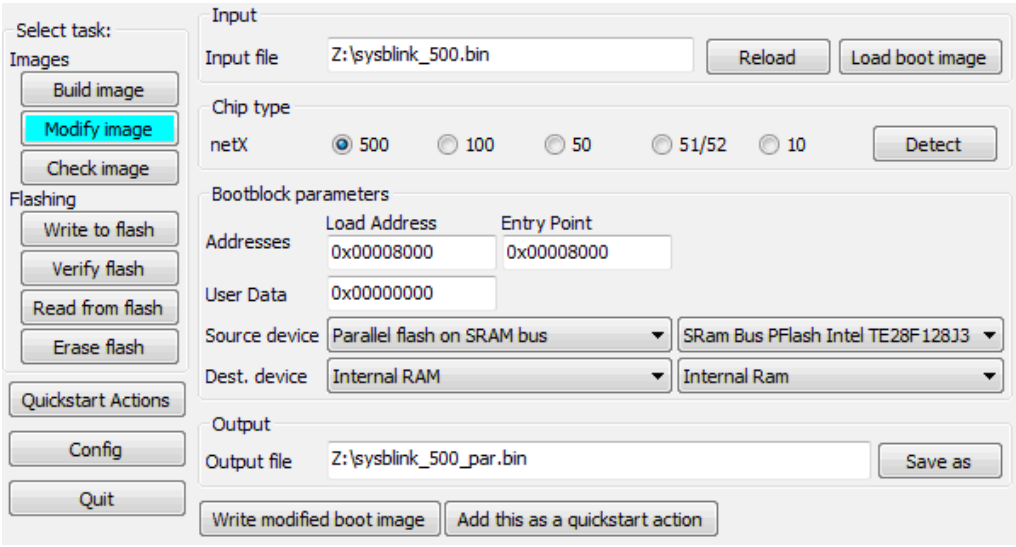

Figure 11: The "Modify Bootimage" GUI

5.1.3 Check a Bootable Image

You can check a boot image before downloading it to the flash memory of the netX device. In order to avoid boot problems, the **Check image** function of the Bootwizard will perform the following tasks:

- The presence of the magic constants (magic cookie and netX signature) is checked.
- The application and boot header checksums are re-calculated and compared with those in the boot header.
- The application size according to the length of the image is compared to the application size stored in the header.

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

↬ The Bootwizard opens.

2. Click on the **Check image** task button.
3. Select the image you want to check.

➢ In the Input area, click on **Load boot image** button to load the relevant file.

Note: After loading a boot image, the Bootwizard tries to set the chip type, source and destination device automatically. If it recognizes different chip types or devices, you can ignore these settings, since the check does not depend on them.

↬ If the image contains a valid header, the address and device information is shown:

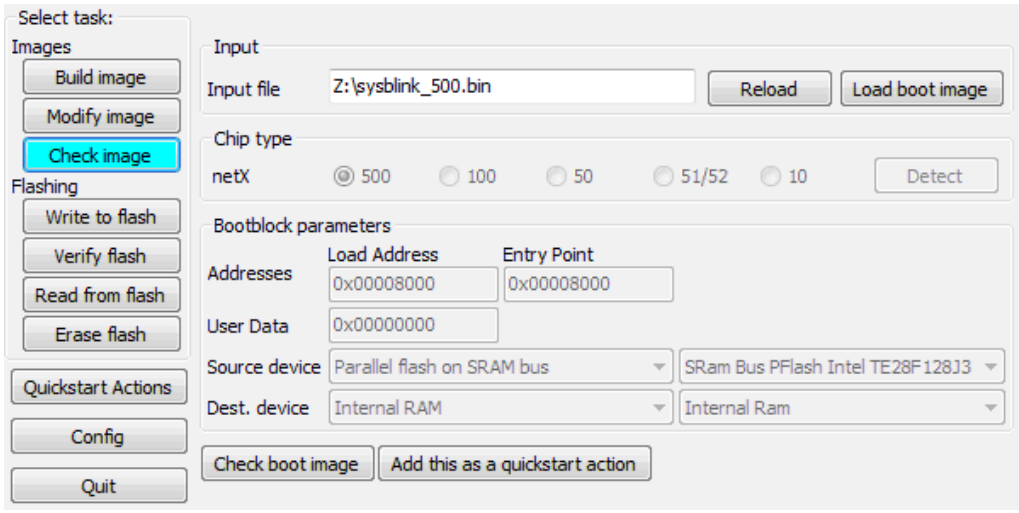

Figure 12: The "Check Image" GUI

4. Click on **Check boot image** button.

➪ First, the presence of a few magic bytes is checked (the magic cookie and netX signature). If these bytes are not found, an error message is shown and no further checks are performed. If they are found, the checksums over the boot header and the application data, and the size of the application data are re-calculated and compared to the information stored in the header.

➪ A window appears, showing the results.

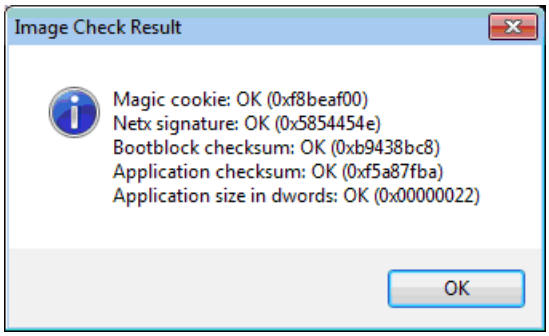

Figure 13: The "Check image“ Result Window

5. Click on **OK** button to close the result window.

5.2 Flasher Functions

The flasher functions of the Bootwizard are relevant to all users who need to download a bootable image for a netX controller from a PC to a flash memory connected to the netX. This can be design-in OEMs who have used the boot image functions of the Bootwizard to create their own custom bootable firmware for netX and now need to download it.
This can also be OEMs who use Hilscher Standard Loadable Firmware (LFW) in NXF format on their netX device (either on a NXHX Board or on a design-in netX device), which can not be directly booted by the ROM Loader of the netX, and therefore need to download the Second Stage Bootloader (SSBL) first, before they can download and operate the standard LFW on their device (the ROM Loader boots the SSBL, the SSBL then loads the LFW to the SDRAM).

5.2.1 Prerequisite: Connecting PC to netX Device

The flash functions and the "detect chip type" function require a connection from your PC to the netX chip. The Bootwizard supports the following interface types: USB, JTAG and Ethernet (Ethernet only on netX 51/52 controllers).

How to connect

1. Put your netX device into boot mode.

➢ In order to communicate with the netX chip via a serial, USB or Ethernet connection, you need to put it into boot mode.
Consult your device documentation to find out how to do so.

Note: When using a JTAG connection, this is not necessary.

2. Establish cable connection from PC to netX device.

➢ If not already installed, you now have to install the Windows drivers, respectively (i. e. under Windows XP), you now have to finish driver installation in the **Found New Hardware Wizard)**.

3. Select plugin.

➢ When you click on the **Flash**/**Verify**/**Read**/**Erase** button to start the corresponding action, the Bootwizard first scans for available interfaces. A window titled **Select the plugin** is opened and shows the recognized devices or interfaces:

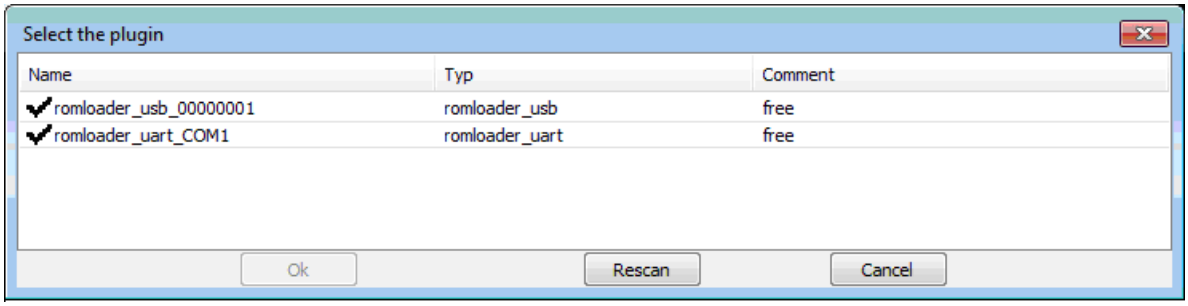

Figure 14: The Plugin Selector Dialog

Note: If the netX device is not shown, try clicking on **Rescan**.
If you want to abort the function, click **Cancel**.

The various netX devices and interfaces are identified as follows:

Device	Interface	Plugin name (example)	Plugin type
all netX	serial port	romloader_uart_COM1	romloader_uart
netX 500/100/10	USB	romloader_usb_00_00	romloader_usb
netX 50, 51, 52	USB virtual COM port	romloader_uart_COM42	romloader_uart
NXHX boards	onboard JTAG	romloader_openocd_NXHX500-RE	romloader_openocd
all netX	JTAGkey Tiny	romloader_openocd_netX500/100 JTAGkey	romloader_openocd
netX 51/52	Ethernet	romloader_eth_10.10.5.0	romloader_eth

Table 4: Appearance of netX chips and interfaces in the Plugin Selector Window

4. Execute flash function.

➢ In order to execute the desired flash function (flash, verify, read or erase flash memory) on your netX device, select the entry representing your device/interface in the **Select the plugin** window, then click **OK** button (alternatively, you can double-click on the entry).

⇨ The flash function is executed.

Notes on Connection Modes

Please note the following about the different connection modes:

UART

The romloader_uart plugin simply lists all available COM ports. It does not detect if a netX is listening on a port, since doing so could confuse other devices. netX 50, 51 and 52 provide a virtual COM port via USB, which is also handled by the UART plugin.

The UART plugin does not work with the netX51 Step A.

USB

The romloader_usb plugin scans for known netX USB IDs. It does not check whether the chip is listening on the port, whether it is busy or if it has crashed.

USB monitor update (applies to netX 500/100/10):

After a reset, the netX is running a UART/USB monitor routine contained in the ROM. When you execute a detect or flash operation, a new (faster) monitor routine is downloaded into the internal RAM and takes over from the ROM routine until the next reset or power cycle. It also sets a new USB ID. If you unplug and re-plug the USB cable while it is running, Windows will recognize the alternate USB ID as a new device and require the installation of a driver.

JTAG

The romloader_openocd plugin connects to JTAG interfaces based on the FTDI 2232 USB interface chip, which is used on NXHX boards and on the JTAGkey adapter.

For each JTAG configuration listed in the openocd.xml file, the plugin checks if a device with a matching USB ID and description string is present. If yes, it tries to attach and check the JTAG device ID.

If you connect to an NXHX board using the on-board USB-JTAG interface, an additional COM port may appear in the list, which is the secondary channel of the USB interface chip. Ignore this entry and select the entry starting with "romloader_openocd_".

Ethernet

In order to connect via Ethernet, a DHCP service must be present on the network or the IP parameters must be configured via the security memory.

General

If a netX is connected via USB/UART/Ethernet AND via JTAG at the same time, only the JTAG connection will work, because the netX – and thus the monitor routine – is halted by the JTAG plugin.

If a netX device is connected via two interfaces, e.g. UART and USB, and you execute a flash operation using one of them, the monitor routine will stop listening on the other interface. Both interfaces will continue to be listed but only one will work until the netX is reset.

The netX 51/52, when connected via USB, appears to the PC as a composite device consisting of two logical devices, a vendor-specific USB device and a virtual COM port. A driver must be installed for each logical device. However, only the virtual COM port is shown in the selection list.

5.2.2 Write Data to Flash Memory

Note: The Bootwizard supports flashing a binary image to a flash device on a connected netX. If this image should be executed automatically by the netX ROM loader on startup, it has to be patched before flashing it, i. e. a boot header has to be put in front of the executable (for more information, please see *Build a Bootable Image* section on page *25*).

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

The Bootwizard opens.

2. In the **Select task** area, click on **Write to flash** button.
3. In the **Input** area, click on **File to flash** button to open the file you want to download to the flash memory.

If the file is a bootable image with a valid header, the **Bootblock parameters** are displayed. The Bootwizard will also examine the header in order to set the flasher parameters automatically.
However, you must still check whether the settings are correct.

4. Select the **Chip type**.
5. If the file contains a boot header, you can change the **User Data** value. This affects only the copy which will be downloaded to the flash, not the original input file.
6. Set the **Flasher parameters**.

➢ Set the flash area **Offset**.

Note: If the device contains a flash device label and the data to be written overlaps the protected area containing the label, this area will not be written to. Any data outside the protected area will be written and the Bootwizard will notify you with a message. For more information on Flash Device Labels, see *Flash Device Label* chapter on page 66.

➢ Select the **Flash type**.

➢ Select the **Buffer**.

➢ The GUI should now look similar to this:

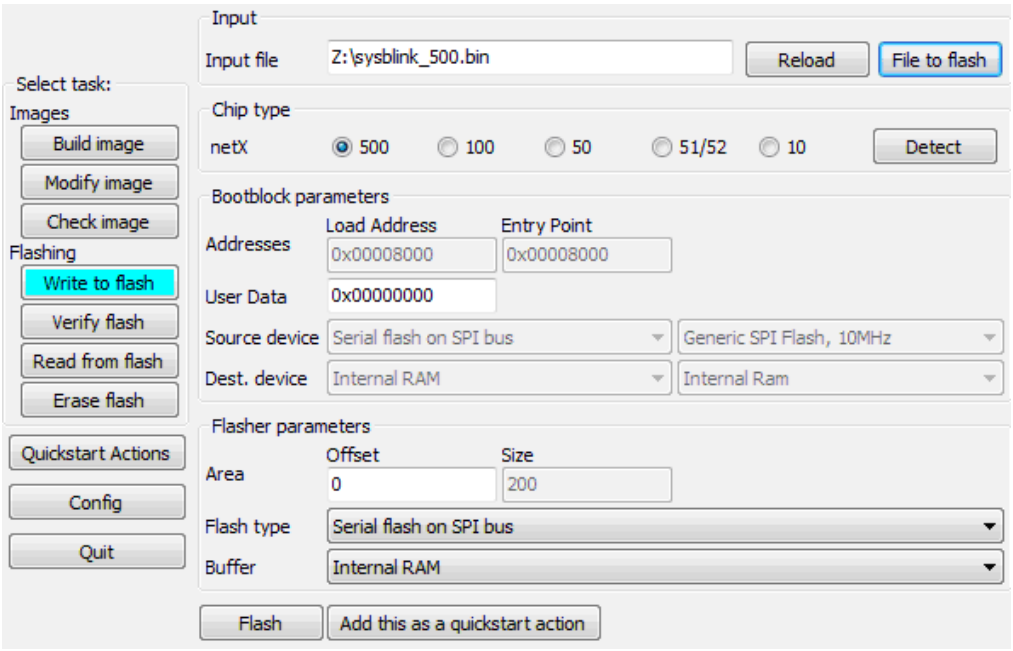

Figure 15: The "Write Flash" GUI

7. Put your netX device into serial or USB boot mode, if necessary. Consult your device documentation to find out how to do so.

➢ If successful, the SYS LED on your netX device blinks yellow, once per second.

8. Click on **Flash** button.

➢ The plugin selector opens:

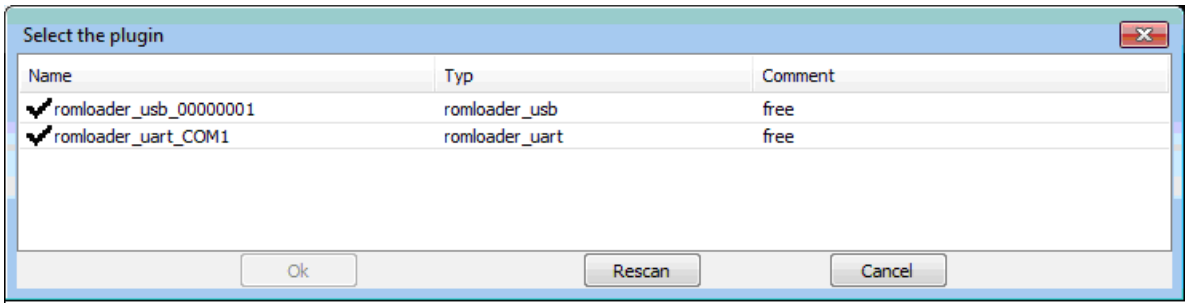

Figure 16: The Plugin Selector Dialog

9. In the **Select the plugin** window, select the entry representing your device/interface, then click **OK** button (alternatively, you can double-click on the entry).

The Bootwizard will now send the data file and the flasher program to the netX device and run the flasher. During this time, a progress bar is shown:

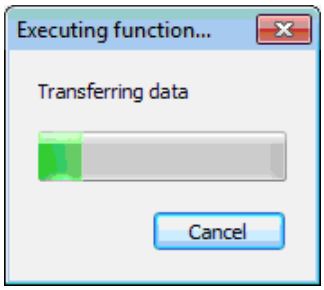

Figure 17: Progress Window

Note: After flashing, the written data is verified by reading it back and comparing it to the image in RAM. You do not have to verify explicitly.

Finally, a success message or an error message is shown. If a conflict with a Flash Device Label occurred, you will also be told about this in the message:

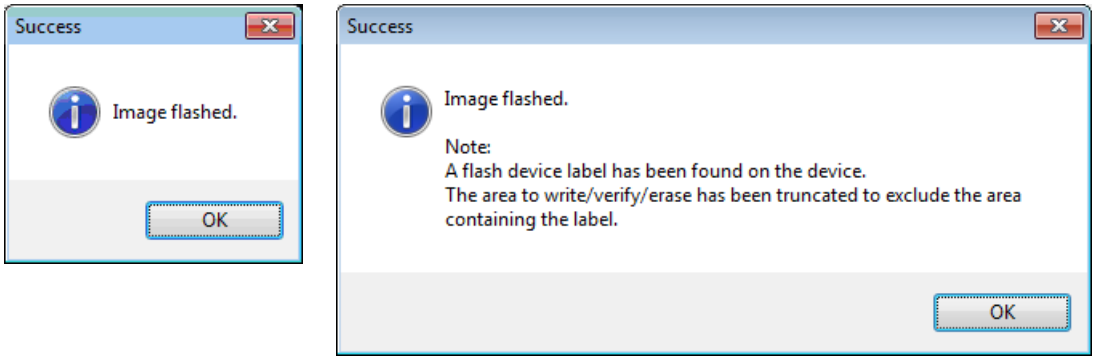

Figure 18: Flashing Messages

Please note the following:

- If a boot header has been recognized, you can edit the **User Data** field. When you start flashing, the value is read back from the GUI and the boot header checksum is updated. However, the internal checksums in NXF files are not updated.
  The boot header checksum is updated regardless of whether you actually edited the User Data value. The checksum is also updated if it was incorrect in the first place.
- Before writing data, the flasher checks if the area is already erased and erases it if not. Since the flash is comprised of erase blocks which can only be erased as a whole, the area actually erased may be larger than the specified offset/size.
- Areas containing a Flash Device Label will not be erased or overwritten if you are using a Bootwizard version ≥ 1.3.15584.0.
- If you want to write several separate blocks of data, either make sure that they do not occupy overlapping erase blocks, or erase the entire area first, then write the data blocks.
- In case of parallel flash, the offset and the size must be a multiple of the data width, i.e. 8, 16 or 32 bit. Otherwise, the write access will fail. For example, if you are using a paired 16 bit flash, the data width is 32 bit.

5.2.3 Verify Flash Memory

You can compare the contents of a flash device with any binary file.

The procedure is similar to the **Write to flash** function, except that no data is actually written to the flash and its content stays the same. Instead of flashing, the result of the comparison is shown afterwards.

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

➧ The Bootwizard opens.

2. In the **Select task** area, click **Verify flash** button.
3. In the **Input** area, click on **File to verify** button to open the file you want to compare with the contents of the flash memory.
4. Set the **Flasher parameters**.
5. Put your netX device into serial or USB boot mode, if necessary. Consult your board documentation to find out how to do so.

➧ If successful, the SYS LED on your netX device blinks yellow, once per second.

6. Click **Verify** button.

➧ The plugin selector opens:

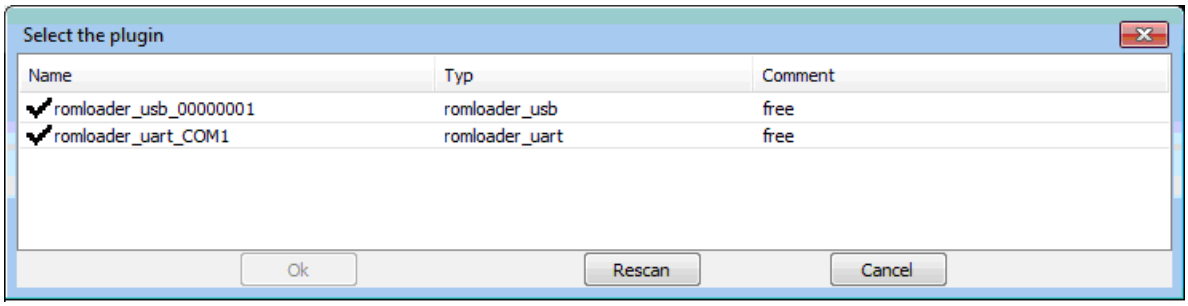

Figure 19: The Plugin Selector Dialog

7. In the **Select the plugin** window, select the entry representing your device/interface, then click **OK** button (alternatively, you can double-click on the entry).

➧ The Bootwizard now verifies the flash memory while a progress bar is displayed.

➧ After the flash memory has been verified, either a success or an error message is displayed. If an overlapping with a Flash Device Label occurred, you will also be told about this in the message.

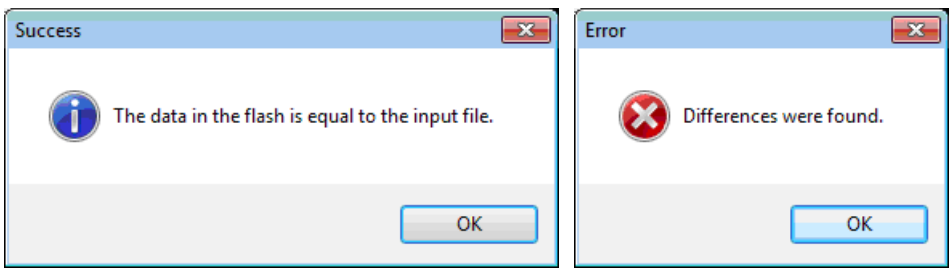

Figure 20: Verify Flash Results

5.2.4 Read from Flash Memory

You can read data from a flash device and save the output to a file.

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

➢ The Bootwizard opens.

2. In the **Select task** area, click **Read from flash** button.
3. Select the **Chip type**.
4. Set **Flasher parameters**.

➢ Enter the starting **Offset** and the **Size** of the area to read, or select **All** to read an image of the whole flash. **Offset** and **Size** are in bytes and decimal or hexadecimal.

Note: When reading from a device containing a Flash Device Label (like e. g. the Hilscher netRAPID Chip Carrier NRP 52-RE), you should be aware of the fact that the output file will be truncated if the area read from the device overlaps with the area containing the flash device label. The protected data in the area containing the flash device label will also be read by the Bootwizard, but it will be written to a separate file with the name extension "**_fdl**". The Bootwizard will notify you with a message after reading from flash if the task was affected by overlapping with the Flash Device Label.

➢ Select the **Flash type**.

➢ Select the **Buffer** device.

5. In the **Output** area, click on **Save as** button to specify name and storage location of the output file on your PC.

➪ The GUI should look similar to the following image:

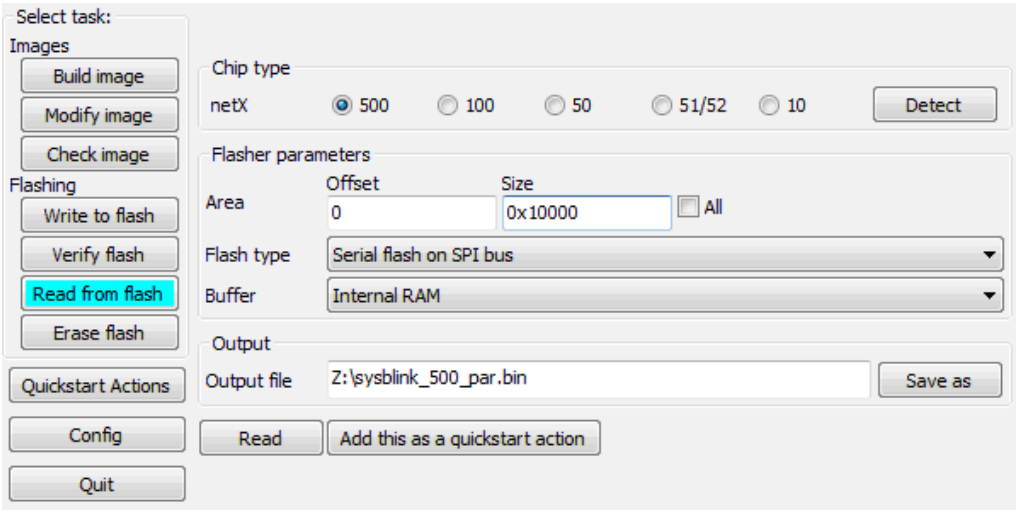

Figure 21: The "Read Flash" GUI

6. Put your netX device into serial or USB boot mode if necessary. Consult your device documentation to find out how to do so. If successful, the SYS LED blinks yellow, once per second.

7. Click on **Read** button.

➪ The plugin selector opens:

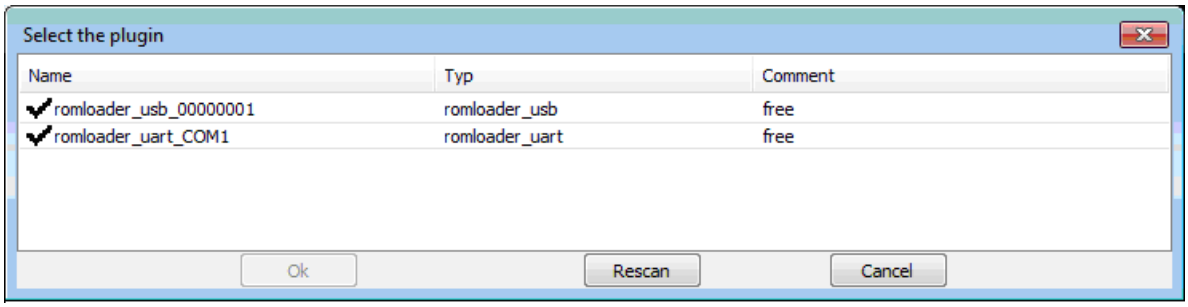

Figure 22: The Plugin Selector Dialog

8. In the **Select the plugin** window, select the entry representing your device/interface, then click **OK** button (alternatively, you can double-click on the entry).

- The flasher program is sent to the hardware and runs. The flash memory is read and buffered in RAM, then transferred to the PC. During the transfer, a progress bar is shown:

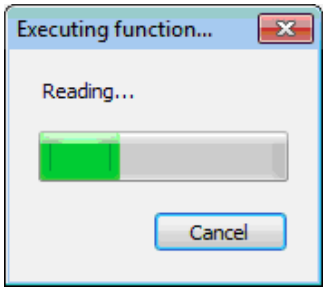

Figure 23: Progress Window

- Finally, a success message or an error message is shown. If an overlapping of the read area with a Flash Device Label occurred, you will also be told about this in the message:

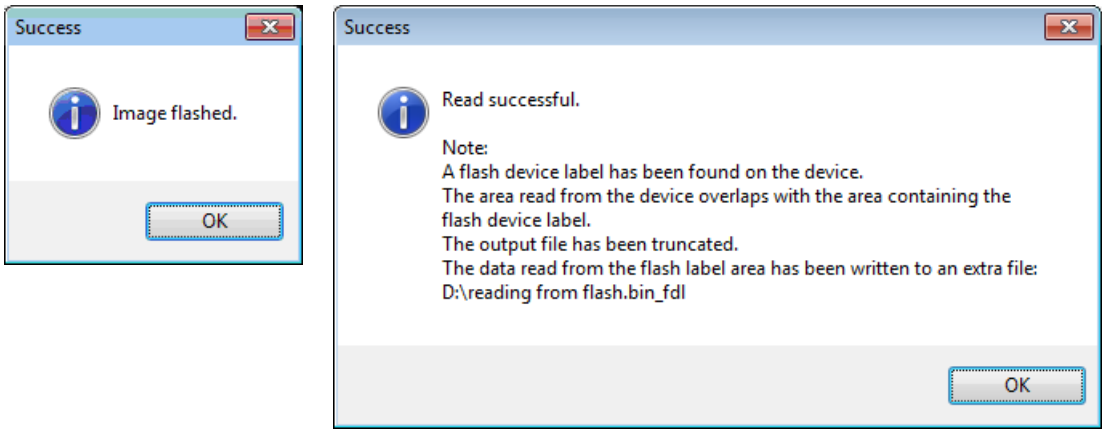

Figure 24: Flash read successful

5.2.5 Erase Flash Memory

1. Open the netX Bootwizard application on your PC.

➢ In the Windows **Start** menu, choose **All Programs** > **Hilscher GmbH** > **Bootwizard** > **Bootwizard**.

➪ The Bootwizard opens.

2. In the **Select task** area, click **Erase flash** button.
3. Select the **Chip type**.
4. Set the **Flasher parameters**.

➢ Enter the **Offset** and the **Size** of the area to erase, or select **All** to erase the whole chip. **Offset** and **Size** are in bytes and decimal or hexadecimal.

Note: Areas in the flash memory containing a **Flash Device Label** will not be erased. The Bootwizard will notify you with a message if the task was affected by a conflict with a Flash Device Label.

➢ Select the **Flash type** you want to erase

➪ The GUI should now look like this:

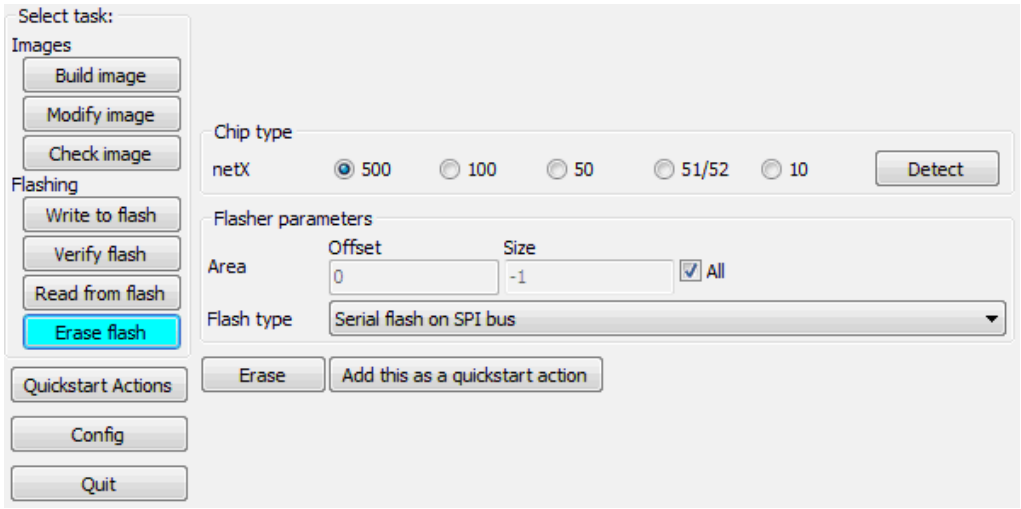

Figure 25: The "Erase Flash" GUI

5. Put your netX device into serial or USB boot mode, if necessary. Consult your device documentation to find out how to do so. If successful, the SYS LED blinks yellow, once per second.
6. Click on **Erase** button.

➮ The plugin selector opens:

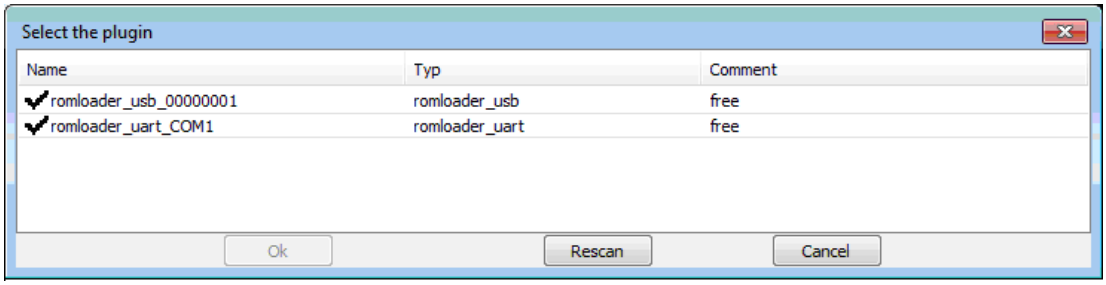

Figure 26: The Plugin Selector Dialog

7. Double-click on the device name.

➮ The flasher program is sent to the hardware and runs. The flasher checks if the requested area is already empty. If not, the area is erased and the following window is shown:

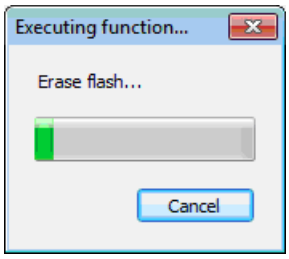

Figure 27: Erasing Memory

Note: The area actually erased may be larger than the area you have specified, since flash memory consists of erase blocks which can only be erased as a whole.

➮ Finally, a message window appears. If an area could not be erased due to protection by a Flash Device Label, this will also be mentioned in the message:

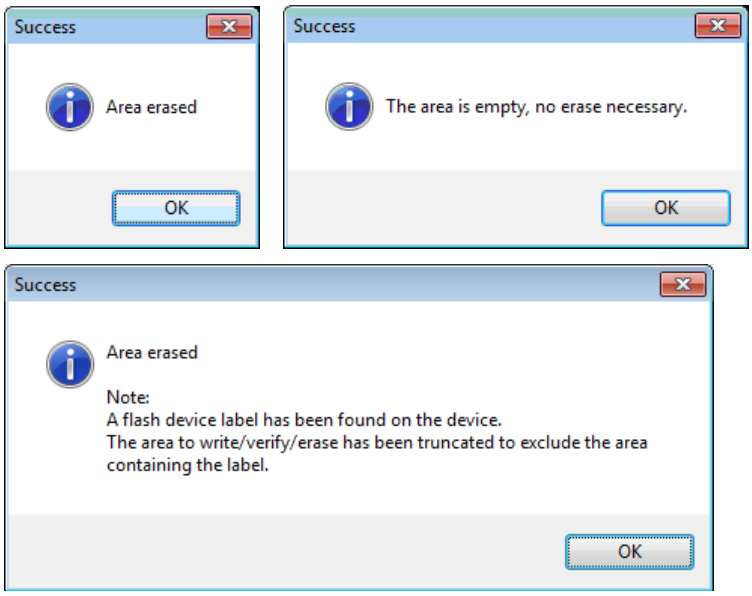

Figure 28: Area Erased Message

6 Customizing the Bootwizard

6.1 Quickstart Actions

A Quickstart action stores all the settings in the user interface. It can be used as a shortcut to frequently used actions, or as a template, i.e. some fields may be left blank. You can bring up the quickstart window at any time using the **Quickstart Actions** button on the left side.

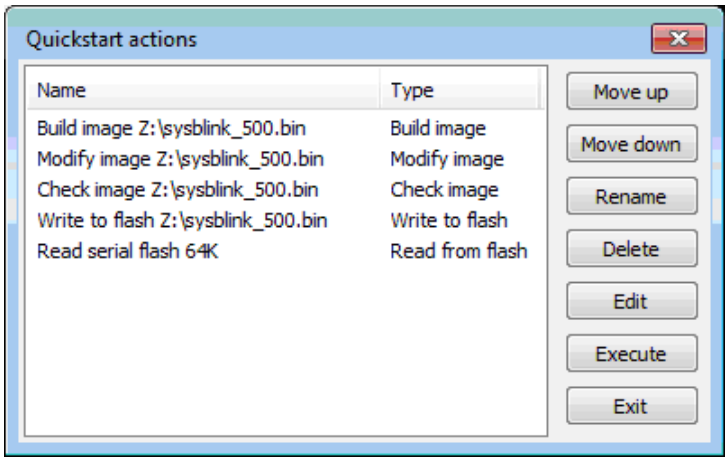

Figure 29: The Quickstart Window

Add a new Quickstart Action

To create a new quickstart action, use the button labeled **Add this as a quickstart action**. When you click this button, the current settings are saved to a new quickstart action, and the list of quickstarts is shown, with the new entry at the top of the list. You may want to select it and click on the **Rename button** to give it a more meaningful name.

Execute a Quickstart Action

A Quickstart action can be executed

- by double-clicking the action in the list
- by selecting the Quickstart action in the list and selecting **Quickstart Actions/Execute** from the menu

Execute means that the GUI is restored according to the information in the quickstart action, with one exception: The load and entry addresses are obtained from the input file, since that file may have changed. The addresses from the quickstart action are only used in the special case where you load a raw binary and set the addresses manually. The input file is read and the parameters are set, but the task is not executed yet. You can still change the parameters in the GUI. When you are done, click on the **Go** button (**Build image** / **Flash** etc.) to execute the task.

Edit a Quickstart Action

You can change the settings stored in a quickstart action by selecting it in the list and clicking on the **Edit** button. The GUI will be set up according to the settings stored in the action as if you had pressed the **Execute** button, but instead of the **Go** and **Add quickstart** button two other buttons will be shown:

- **Replace quickstart action** will save the current settings to the quickstart action, and bring up the quickstart list. Remember to rename the action to reflect its new settings, if necessary.
- **Cancel editing quickstart action** will exit the edit mode, and the normal **Go** and **add quickstart action** buttons will be shown instead.

Note: The ‘Edit’ operation also allows you to change the action of an item, e.g. from ‘build bootimage’ to ‘flash bootimage’. Please do not forget to rename the Quickstart action according to its new function.

Rename a Quickstart Action

To rename an action, select it and click on the **Rename** button. You can also select it and click on the name a second time after a short delay (i.e. do not double-click). Edit the name and press enter to confirm the change.

Delete a Quickstart Action

A quickstart action can be deleted by selecting it in the list and clicking the **Delete** button. Note that no confirmation dialog will be shown.

Move a Quickstart Action

After selecting a quickstart entry, you can move it up or down using the **Move up** and **Move down** buttons.

Exiting Quickstart List

Click **Exit** to leave the quickstart list.

6.2 Customizing the Bootwizard Configuration

The **Config** button opens the configuration window, which allows you to select some files used by the Bootwizard:

- The netx.xml file, which contains the Toolchain scripts and device descriptions
- The flasher binaries for the different chip types

For each file, there is a checkbox which tells the Bootwizard to use the built-in files, which is the default. If you uncheck the box, you can use the file requester to select a file which will be used instead of the built-in one.

If you click the **OK** button and any external files are selected, the existence of these files will be verified, and a message will appear if any of them do not exist. If the dialogue closes, the currently selected XML file will be loaded.

You can also discard any changes and leave the dialog by clicking **Cancel**.

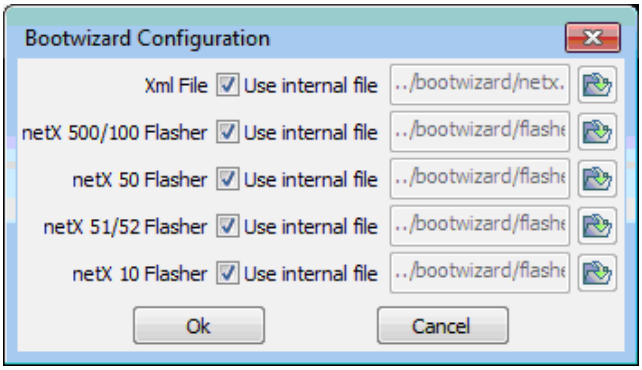

Figure 30: The Configuration Dialog

7 Using the Command Line Bootblocker Script

The **bootblocker.bat** is a command-line driven tool intended for the inclusion in build scripts. It can be used as a more potent alternative to the Bootwizard. The Bootblocker offers several functions:

- Generating bootable images (-bootimage)
- Modifying the source/destination devices of an image or the serial number/user data field of the boot header (-modify)
- Checking the validity of a bootable image and identifying the source and destination devices (-check)
- Listing information from the XML file (-listsrc, -listdst, -listtools)
- Extracting the load and execution addresses and the raw binary from an ELF file (-loadadr, -execadr, -extract)
- Building NXF files with common header V3/Tag list
- Updating checksums in NXF files

Note: bootblocker.bat requires the environment variable PATH_BOOTWIZARD to be set. The installer does this for you unless you de-select this option in the installer.

7.1 Syntax

The command line syntax is as follows:

```
bootblocker.bat [mode] [arguments/flags] [input file]
```

"mode" selects an operation, e.g. building a bootable image. It may be placed between the other arguments.

Arguments can be written in two notations: with a short argument identifier and the value separated by a space:

```
-o firmware.bin
```

or with a long identifier and in the form of an assignment:

```
--output=firmware.bin
```

For flags, the short notation is:

```
-uh
```

and the long notation is:

```
--updateCommonHeader
```

If an input file is required, it is put as the last element in the command line.

7.2 Arguments and Flags

Short name	Long name	Description
-x	--xmlfile	Path to and name of the netx.xml file. **Default value: <install path>\bootwizard\netx.xml**
-t	--toolset	The id of the toolset to use for extraction of the binary and load/entry addresses.
-s	--source	Source device id
-d	--destination	Destination device id
-ct	--chiptype	The chip type is required to select the correct device parameters. Possible values: “netx500”, “netx100”, “netx50”, "netx 10" or "netx 51_52" **Default value: netX 500**
-ud	--userdata	The userdata parameter is a 32 bit unsigned number which is stored in the ulSerial field of the boot header. The parameter may be in decimal or hexadecimal notation.
-tl	--taglist	Assume that the binary begins with a common header V3, include a binary file as a tag list and update the common header V3.
-uh	--updateCommonHeader	Assume that the output file has a common header V3 and update it.
-o	--output	Name of the output file
-v	--verbose	This flag enables more comprehensive output.

Table 5: Bootblocker Command Line Arguments and Flags

The following table lists the arguments/flags which are required or optional for the different modes. "if" stands for the input file, which is the last command line argument. Additionally, the "verbose" (-v) flag may be used.

Mode	XML file	Chip type	Tool set name	Source device	Dest. device	User data	Tag list	Update common header	Output file	Input file
-boot image	(-x)	(-ct)	-t	-s	-d	(-ud)	(-tl)	(-uh)	-o	if
-check	(-x)									if
-modify	(-x)	(-ct)		(-s)	(-d)	(-ud)		(-uh)	-o	if
-listsrc	(-x)									
-listdst	(-x)									
-listtools	(-x)									
-extract	(-x)		-t						-o	if
-loadadr	(-x)		-t						(-o)	if
-execadr	(-x)		-t						(-o)	if

Table 6: Mandatory and optional arguments/flags by mode. Arguments/flags in parentheses are optional.

7.3 Operation Modes

-bootimage: Generates a bootable image from an ELF file. This is the default if no mode is given on the command line.

Required arguments: toolset, source device, destination device, input file, output file

Optional arguments/flags: XML file, chip type, user data, taglist, updateCommonHeader

Example:

```
bootblocker.bat -bootimage --xmlfile=netx.xml --toolkit=HITEX_ARM_WIN_4_00
--source=SPI_NXSB100 --destination=SD_MT48LC2M32B2 --–chiptype=netx500
–-output=myfile.bin myfile.elf
```

Using the short forms of the arguments and omitting the arguments which have default values, this can be abbreviated to the following:

```
bootblocker.bat -t HITEX_ARM_WIN_4_00 -s SPI_NXSB100 –d SD_MT48LC2M32B2
–o myfile.bin myfile.elf
```

-modify: Changes the source and/or destination device parameters in a bootable image.

Required arguments: input file, output file

Optional arguments/flags: XML file, chip type, source device, user data, destination device, updateCommonHeader

Example:

```
bootblocker.bat -modify -x netx.xml –ct netx500 –s SRB_PF_TE28F128J3
–o myfile_par.bin myfile.bin
```

-check: Checks the validity of a bootable image, and show its source/destination device.

This command checks the presence of the magic cookie and netX signature, the application length, the application checksum and the boot header checksum. Then, it tries to re-identify the source and destination device using the device parameters and the load address of the application. Since it is usually not possible to detect exactly those device names which were used to generate the image, all matching devices are displayed.

Required arguments: input file

Optional arguments: XML file

Example:

```
C:\Programme\Hilscher GmbH\Bootwizard_test\bootwizard>bootblocker.bat -check
d:\projekt\netx-testimages\nx50dps.bin

 0 0xf8beaf00 MagCok
 1 0x00040000 Speed/Res0/MemCtrl
 2 0x800080b8 AppEnt
 3 0xbd5f0f01 AppChksm
 4 0x0000c459 AppFilSiz
 5 0x80008000 AppSrtAdd
 6 0x5854454e Signt
 7 0x030d0001 SdramGeneralCtrl/SramCtrl/ExpBusReg
 8 0x00a12151 SdramTimingCtrl/IoRegMode0
 9 0x00000000 IoRegMode1/Res1
10 0x00000000 IfConf0/Res2
11 0x00000000 IfConf1/Res3
12 0x00000001 MiscAsicCtrl
13 0x00000000 Serial
14 0x00000002 SrcType
15 0xedda164b BootChksm

check result:
Magic cookie: OK (0xf8beaf00)
Netx signature: OK (0x5854454e)
Bootblock checksum: OK (0xedda164b)
Application checksum: OK (0xbd5f0f01)
Application size in dwords: OK (0x0000c459)

Device identification

matching source devices for netx 50:

SPI_AT45DB321D: 'Atmel AT45DB321D SPI Flash', 0x00420000 bytes
SPI_gen_25: 'Generic SPI Flash netX50, 25MHz'

matching destination devices for netx 50:

SD_MT48LC2M32B2: 'SDRam MT48LC2M32B2', 0x00800000 bytes
```

-extract: Extracts the binary from an ELF file.

Required arguments: toolset, input file, output file

Optional arguments: XML file

-loadadr/execadr: Extracts the load/execution address from an ELF file. If an output file is given, the address is written to the file as a hexadecimal string, otherwise it is printed to the screen.

Required arguments: toolset, input file

Optional arguments: XML file, output file

-listsrc, -listdst: Show the names, ids and sizes of source or destination devices contained in the XML file. If the verbose flag is present, the parameters of these devices will be shown, too.

Optional arguments: XML file, verbose

-listtools: Show the names and ids of the toolsets (i.e. the Lua extraction scripts) in the XML file.

Optional arguments: XML file

-help: print the help text.

Arguments: none

7.4 Exit Codes

bootblocker.bat exits with %ERRORLEVEL% = 0 if the operation finished successfully and with %ERRORLEVEL% = 1 if any error occurred.

7.5 Adapting Build Scripts to the New Bootblocker Version

Although the command line syntax is close to that of the old bootblocker.exe, some changes are necessary.

For netX 500 build configurations:

- Change calls to bootblocker.exe to ‘bootblocker.bat’. If the call is made from another batch file, use “call bootblocker.bat”

For netX 50 build configurations:

- Change calls to bootblocker.exe to ‘bootblocker.bat’. If the call is made from another batch file, use “call bootblocker.bat”
- Specify the chip type on the command line (-ct netx50)
- Check the device identifiers: in the current netx.xml file, the same device ids are used for netX 500 and netX 50. In previous versions, different ids were used.

7.6 Building NXF Files With Common Header V3/Tag List

NXF firmware files have the following structure:

- Boot header: Information required by the ROM code to load and start the image. If the image is used with the Second Stage Bootloader, the header may be empty except for an initial NXF marker.
- Common header V3: Describes the structure of the file, the location and size of the headers, application binary and tag list. Also contains two additional checksums.
- Other headers: Other information for firmware validation: hardware options, licenses etc.
- Application binary: the actual code
- Tag list (optional): Configuration data, e.g., task and interrupt priorities, xC unit number or UART configuration.

The Bootwizard application itself is not aware of this structure. It cannot insert a tag list from a binary file and does not update the offset/size fields and the additional checksums in the common header. Use the bootblocker in order to build an NXF file.

There are three possible scenarios:

1. There is no tag list, but the common header checksums must be updated (-uh on the command line).

2. A binary file is to be included as a tag list (-tl <filename).

3. A tag list is contained in the ELF file (-uh).

The following table describes which fields are set by which tool, depending on the scenario.

Header field	Set by
HeaderVersion	These fields must be set by the compiler/linker.
HeaderLength	
numModuleInfos	
DataStartOffset	
DataSize	▪ **No tag list or tag list included from external file**: The field may be set by the compiler/linker, or, if its value is 0, the bootblocker sets it to 64 + size of the binary extracted from the ELF file – DataStartOffset. ▪ **Tag list included in ELF file**: The field must be set by the compiler/linker.
TagListStartOffset	▪ **No tag list**: Set TagListSize to 0 in the source. ▪ **Tag list from external file** (-tl): The external tag list is appended and both fields are set accordingly by bootblocker. ▪ **Tag list included in ELF file**: These fields must be set by the compiler/linker
TagListSize	
TagListSizeMax	This field must be set by the compiler/linker. Set to 0 or the maximum allowed tag list size.
MD5 checksum	The checksums are updated by the bootblocker, if a tag list is inserted from an external file (–tl) or the -uh flag is specified.
CRC checksum	

Table 7: Header Field/Tool Matrix

7.7 Updating Checksums in NXF files

After an NXF file with the common header is built, you may open it in Bootwizard to change boot header parameters. After you make any changes and save the file, the checksums will be incorrect. Use the following command to update the checksums:

```
bootblocker –modify –uh –o myfile.nxf myfile.nxf
```

8 Using the Command Line Flash Script

The **flash.bat** command line flasher is a batch file which allows you to run the Bootwizard and automatically flash a file, verify the flash against a file, read data from the flash or erase the flash. It can be used as an alternative to the Bootwizard. With respect to Flash Device Labels, the command line flasher script behaves in the same way as the Bootwizard (see *Flash Device Label* chapter on page 66).

Note: In order to use the script, the environment variable PATH_BOOTWIZARD must be set. The installer does this for you unless you de-select this option in the installer.

8.1 Syntax

The script is located in the directory

```
<installdir>\bootwizard
```

e.g.,

```
C:\Programme\Hilscher GmbH\Bootwizard\bootwizard
```

The general syntax is as follows:

```
flash.bat [command] [interface_name] [chip_type] [flash_type] [buffer] [offset] [size]
[image_path]
```

The specific arguments for flashing, verifying, reading and erasing are:

```
flash.bat        interface_name chip_type flash_type buffer offset      image_path
flash.bat flash  interface_name chip_type flash_type buffer offset      image_path
flash.bat verify interface_name chip_type flash_type buffer offset      image_path
flash.bat read   interface_name chip_type flash_type buffer offset size image_path
flash.bat erase  interface_name chip_type flash_type        offset size
```

The arguments must be specified in the order shown above.

The command

```
flash.bat list_interfaces
```

will list the currently available interfaces.

The Bootwizard window opens and the specified command is carried out.

The netx.xml file and the flasher binary files are used as specified in the Bootwizard configuration. This configuration may be changed using the configuration window in the Bootwizard GUI.

8.2 Arguments

Argument	Description
command	Specifies the operation: flash (default) loads the image file and flashes it verify verifies the flash contents against a file read reads data from the flash and write to file erase erases the flash list_interfaces lists the currently available interfaces.
interface_name	The name of the interface which shall be used to access the netX, as shown by the list_interfaces command. If the name contains any spaces, it must be enclosed in double quotes. Examples: romloader_uart_COM1 romloader_usb_00_00 "romloader_openocd_NXHX 500-ETM"
chip_type	The chip type of the netX device: “netx500”, “netx100”,“netx50”, "netx 10" or "netx 51_52"
flash_type	Indicates whether the flash type is serial or parallel. Possible values are SRB Parallel flash on sram bus SPI Serial flash on SPI bus EXT Parallel flash on extension bus
buffer	The buffer memory to use for flashing, verifying or reading. Possible values are AUTO: configures the SDRAM using parameters from the security memory, if available INTRAM: uses internal RAM the id attribute of an SDRAM device entry in netx.xml, e.g. SD_ MT48LC2M32B2
offset	The start offset in the flash memory
size	The number of bytes to read or erase or "ALL". Setting the size to "ALL" overrides the offset and reads or erases the whole flash memory.
image_path	The file to be flashed/verified or the output file when reading from the flash

Table 8: Flash.bat Command Line Arguments

8.3 Examples

```
flash.bat flash romloader_usb_00_00 netx100 SPI SD_MT48LC2M32B2 0
D:\netX100\UART_Demo_netX100.bin
```

Writes to SPI flash on a netX100 via USB, starting at offset 0 and buffering in SDRAM.

```
flash.bat romloader_usb_00_00 netx100 SPI SD_MT48LC2M32B2 0
D:\netX100\UART_Demo_netX100.bin
```

This is equivalent to the above command since the flash command may be left out.

```
flash.bat romloader_usb_00_00 netx100 SPI INTRAM 0
D:\netX100\UART_Demo_netX100.bin
```

This uses the internal RAM as a buffer.

```
flash.bat flash romloader_usb_00_00 netx100 SRB SD_MT48LC2M32B2 0x40000
test.nxf
```

Writes an NXF file to offset 0x40000 of the parallel flash of a netX100 via USB, buffering in SDRAM.

```
flash.bat verify romloader_usb_00_00 netx100 SPI SD_MT48LC2M32B2
D:\netX100\UART_Demo_netX100.bin
```

Verifies the file against the SPI flash on a netX100 via USB, starting at offset 0 and buffering in SDRAM.

```
flash.bat read "romloader_openocd_NXHX 50-ETM" netx50 AUTO 0 0x100000 D:\readout.bin
```

Reads the first 1 MB from the parallel flash on a netX50 via JTAG and writes the data to a file. If a security memory is present and contains SDRAM parameters, SDRAM is used for buffering. Otherwise, internal RAM is used.

```
flash.bat erase romloader_uart_COM1 netx50 SPI 0 ALL
```

Erases the whole SPI flash on a netX50 via a serial connection

8.4 Exit Codes

flash.bat exits with %ERRORLEVEL% = 0 if the operation finished successfully and with %ERRORLEVEL% = 1 if any error occurred. In case of an error, the message is contained in the file %TEMP%\flasherror.txt .

9 Editing the netx.xml File

The Bootwizard comes with a default XML file, which contains presets for Hilscher boards and the Hitex toolchain. It also includes several common settings which might not be sufficient when used with custom hardware or toolchains. The following chapters describe how to modify the "*netx.xml*" file to support other memory devices and toolchains. They are intended for users who have experience with the XML syntax.

The overall structure of the netx.xml file is as shown below.

The entries under **Toolset** contain scripts which are used to extract binary images from ELF files.
The entries under **src** and **dst** tags contain descriptions of the devices which are selectable as source and destination devices in the user interface. They also contain the parameters for these devices which are put into the boot block when a boot image is built. Since the parameters differ between chip types, there are several lists of device entries.

```
<netX>
     <bin>
          <Toolset id="ARM_WINHITEX_4_03" name="ARM Hitex Gcc 4.0.3 ">
               <lua>
                    function (strElfName, strBinName)
                         -- in case of failure
                         return false, “An error has occurred.”
                         -- in case of success
                         return true, ulLoadAddr, ulEntryAddr
                    end
               </lua>
          </Toolset>
     </bin>

     <src netx_version="netx 500">
          <Device id="SD_MT48LC2M32B2_netx50" name="SDRam MT48LC2M32B2"
                    size="0x00800000" type="SDRAM">
               <Param name="SdramGeneralCtrl">
                    0x010D0001
               </Param>
               <Param name="SdramTimingCtrl">
                    0x00A12151
               </Param>
          </Device>
          …
     </src>
     <src netx_version="netx 50">
          …
     </src>
     <src netx_version="netx 10">
          …
     </src>
     <src netx_version="netx 51_52">
          …
     </src>
     <dst netx_version="netx 500">
          …
     </dst>
     <dst netx_version="netx 50">
          …
     </dst>
     <dst netx_version="netx 10">
          …
     </dst>
     <dst netx_version="netx 51_52">
          …
     </dst>
</netX>
```

All objects inside the xml file need to implement the following attributes:

Attribute	Description
id	Unique identifier without spaces
name	Human readable name displayed in the dropdown boxes

Table 9: Common XML Configuration Object Attributes

Important: All IDs inside this file must be unique across all objects or the Bootwizard may confuse device descriptions or toolchain scripts. There is one exception: Entries for the same memory device and the same category (src/dst) but for different netX types should have the same ID.

Note: If you update the XML file while Bootwizard is running, enter the Config menu and click on **OK**. This will cause the Bootwizard to re-read the XML file.

9.1 Toolset Entries

A toolset entry is necessary to build a bootable image from an ELF file. It is used to

- extract the startup address,
- extract the main entry point address and
- create a raw binary image from the ELF file.

The Bootwizard uses Lua scripts (www.lua.org) to extract this information. These scripts are contained in the netx.xml file, under <netX><bin>. Each script is embedded in a <Toolset> tag, which contains a <lua> tag, which in turn contains the Lua script:

```
<netX>
     <bin>
          <Toolset id="ARM_WINHITEX_4_03" name="ARM Hitex Gcc 4.0.3 ">
               <lua>
                    function (strElfName, strBinName)
                         -- in case of failure
                         return false, “An error has occurred.”
                         -- in case of success
                         return true, ulLoadAddr, ulEntryAddr
                    end
               </lua>
          </Toolset>
     </bin>
</netX>
```

Function Parameters	
strElfName	The filename of the ELF file to process. The GUI has already checked that the file exists.
strBinName	The name of the binary file to which the application binary should be written
Return values if successful	
true	Boolean true, indicates success
ulLoadAddress	The load address (unsigned long)
ulEntryAddress	The entry/run address (unsigned long)
Return values if not successful	
false	Boolean false, indicating failure
strMessage	An error message string

Table 10: Function Parameters Toolset Entries

Example Entry:

The following example is the script using the Hitex GNU ARM toolchain. It consists of four sections: Initialization, extracting the raw binary, extracting the load address and extracting the entry address. The addresses are extracted from the output of the objdump tool using regular expressions. You may have to adapt these expressions if you use a different toolchain, or a different version of the toolchain.

```
<bin>
<Toolset id="HITEX_ARM_WIN_4_00" name="HiTex Gnu Tools 4.00">
<lua>
function (strElfName, strBinName)
     print("toolset", strElfName, strBinName)
     require("utils")

     -- setup the paths to objcopy/readelf and the command strings
     local strPath = os.getenv("PATH_GNU_ARM")
     if not strPath or strPath:len()==0 then
          return false, "PATH_GNU_ARM not set"
     end

     local strObjcopyCmd = string.format(
          [["%s\bin\arm-hitex-elf-objcopy.exe" -O binary "%s" "%s"]],
          strPath, strElfName, strBinName)
     local strObjdumpCmd = string.format(
          [["%s\bin\arm-hitex-elf-objdump.exe" -f -h "%s"]],
          strPath, strElfName)

     local ulLoadAddr = nil
     local ulEntryAddr = nil

     -- build raw binary
     local iRes, strOutput = utils.runcommand(strObjcopyCmd)
     if iRes ~= 0 then
          return false, "Can't extract binary"
     end

     -- extract load/entry address
     iRes, strOutput = utils.runcommand(strObjdumpCmd)
     if iRes ~= 0 then
          return false, "objdump failed"
     end

     -- parse loadable sections
     -- 1 .itcm_data    000013f8  00000100  8000c000  00000100  2**2
     --                  CONTENTS, ALLOC, LOAD, READONLY, CODE
     local lineend =
          "[^"..string.char(13,10).."]*" .. "["..string.char(13,10).."]+"
     local section_pattern=
          " +(%d+) +([^ ]+) +"--number/name
          .."(%x+) +(%x+) +(%x+) +(%x+)"..lineend -- size, vma, lma, offset
          .."[%u ,]+LOAD"..lineend

     local function printf(strFormat, ...)
          return print(string.format(strFormat, ...))
     end
     local strFormat = "%3d %-20s 0x%08s 0x%08s 0x%08s 0x%08s"

     -- 1 .itcm_data    000013f8  00000100  8000c000  00000100  2**2
     print(" # Name    Size       VMA        LMA        Offset")
     for number, name, size, vma, lma, offset in
               string.gmatch(strOutput, section_pattern) do
          printf(strFormat, number, name, size, vma, lma, offset)
          lma = tonumber(lma, 16)
          if not ulLoadAddr or ulLoadAddr > lma then ulLoadAddr = lma end
     end
```

```
     if not ulLoadAddr then
          return false, "Can't extract load address from objdump output"
     end

     local strEntryAddr = strOutput:match("start address (0x%x+)")
     if not strEntryAddr then
          return false, "Can't extract entry address from objdump output"
     end

     ulEntryAddr = tonumber(strEntryAddr)
     if not ulEntryAddr then
          return false, "Can't parse entry address: " .. strEntryAddr
     end

     printf("load addr: %08x  entry addr: %08x", ulLoadAddr, ulEntryAddr)
     return true, ulLoadAddr, ulEntryAddr
end
</lua>
</Toolset>
</bin>
```

9.2 Device Entries

To include custom source and target devices you need to add *"<Device>"* nodes under the *"netX\dst"* or *"netX\src"* node depending on the type of device (source/destination). Since parameters can differ for the different variants of the netX, the “src” and “dst” nodes have a “netx_version” attribute indicating to which chip the device parameters apply:

Chip type	netx_version value
netX 500/100	netx 500
netX 50	netx 50
netX 10	netx 10
netX 51/52	netx 51_52

Table 11: Values of the netx_version attribute. "netx" is followed by a space

```
<netX>
     <dst netx_version="netx 50">
          <Device id="SD_MT48LC2M32B2_netx50" name="SDRam MT48LC2M32B2"
                    size="0x00800000" type="SDRAM">
               <Param name="SdramGeneralCtrl">
                    0x010D0001
               </Param>
               <Param name="SdramTimingCtrl">
                    0x00A12151
               </Param>
          </Device>
     </dst>
</netX>
```

A device should have the following attributes:

Element	Attribute	Mandatory	Description
Device	id	yes	Defines a unique identifier across all elements
	name	yes	Name to be displayed
	type	yes	Type of the device. The following devices are available: SPI Serial flash I2C I2C EEprom SRB Parallel flash on SRAM Bus EXT Parallel flash on Extension Bus INTRAM Internal netX memory SDRAM External SDRAM
	size	no	Optional: size of the device

Table 12: Base Device Configuration Parameters (XML)

Note: Generally, all IDs have to be unique. However, if you create entries for the same device, under the same category (source or destination device) for different netX types, these entries should have the same ID, as this allows the Bootwizard to find the appropriate device parameters when you change the netX type in the GUI.

To allow patching every element of the boot header each device can have several <Param> subnodes containing a value for a boot header element. Each parameter is identified by the name attribute of the node.

The CDATA portion contains the data being patched and must be a hexadecimal value prefixed by "0x".

The following table shows all patchable boot header entries:

Attribute (name)	Header Offset	Description
MagCok	0x00	Magic cookie
MemCtrl	0x04	Parallel Flash on SRAM Bus Memory control value / SPI Speed
Speed		
AppEnt	0x08	Application entry point
AppChksm	0x0C	Application checksum (Is automatically calculated and should not be used in XML file)
AppFilSiz	0x10	Application size in DWORDs (Is automatically inserted and should not be used in XML file)
AppSrtAdd	0x14	Application relocation start (Automatically extracted from ELF file. Should not be used in XML file)
Signt	0x18	'NETX' Signature. Should not be changed, as it is required by the netX ROM loader to start the image.
SdramGeneralCtrl	0x1C	SDRAM General Control Value
ExpBusReg		Extension bus configuration
SramCtrl		SRAM Control Value (netX 50 / HBoot only)
SdramTimingCtrl	0x20	SDRAM Timing Value
IoRegMode0		Extension bus configuration
SdramMR	0x24	SDRAM Mode Register Value (only supported by netX 10, 51, 52)
IoRegMode1		Extension bus configuration
IfConf0	0x28	Extension bus configuration
IfConf1	0x2C	Extension bus configuration
MiscAsicCtrl	0x30	Asic control setting
Serial	0x34	Serial number or user parameter (aulRes[0])
SrcType	0x38	Source Type (aulRes[1])
BootChksm	0x3C	Bootheader checksum (Automatically calculated and should not be used in XML file)

Table 13: Device Configuration Bootheader Parameters (XML)

Important: If source and destination device modify the same parameter of the boot header (e.g. both modify the SDRAM Parameters), Bootwizard will refuse to patch the file.

Note: Please note that adding device entries for other flash chips to netx.xml only affects which devices are shown in the GUI and the generation of boot images. You will be able to generate boot images for the newly added devices, but the flasher may not be able to access the device. The flasher requires a much larger set of device parameters, which are compiled into the flasher binary. It also includes its own detection functions.

10 netX Bootblock

The netX bootblock is a 64 bytes header used by the netX ROM code to recognize, load and start a binary image.

If the image is to be started by the ROM code, it must be preceded by this header.

If the image is to be started by another software, for instance, the Second Stage Bootloader, this software may or may not evaluate the boot header.

The following C-Structure defines the netX bootblock:

```
typedef struct NETX_BOOTBLOCK_Ttag {
     UINT32 ulMagCok;     // magic cookie
     union {
          UINT32 ulMemCtrl;
          UINT32 ulSpeed;    // source device spi and i2c: speed
          UINT32 ulRes;      // reserved field
     } unCtrl;
     UINT32 ulAppEnt;     // application execution address
     UINT32 ulAppChksm;   // application checksum
     UINT32 ulAppFilSiz;  // application filesize in dwords
     UINT32 ulAppSrtAdd;  // application load address
     UINT32 ulSignt;      // bootblock signature
     union {
          UINT32 ulSdramGeneralCtrl; // only for sdram: general control
          UINT32 ulExpBusReg;
          UINT32 ulSramCtrl; // netx50 only
     } unCtrl0;
     union {
          UINT32 ulSdramTimingCtrl; // only for sdram: timing control
          UINT32 ulIoRegMode0;
     } unCtrl1;
     union {
          UINT32 ulSdramMR; // only for sdram: mode register (netX10, 51, 52)
          UINT32 ulIoRegMode1;
          UINT32 ulRes0;     // reserved field
     } unCtrl2;
     union {
          UINT32 ulIfConf0;
          UINT32 ulRes0;     // reserved field
     } unCtrl3;
     union {
          UINT32 ulIfConf1;
          UINT32 ulRes0;     // reserved field
     } unCtrl4;
     UINT32 ulMiscAsicCtrl;
     UINT32 ulSerial;
     UINT32 ulSrcType;
     UINT32 ulBootChksm;    // bootblock checksum
} NETX_BOOTBLOCK_T;
```

Figure 31: netX Bootblock (C-Typedef)

Note: All reserved parameters should be set to zero for maximum compatibility.

The following table describes the header elements:

Offset	Name	Description
0x00	ulMagCok	Identifies a valid bootblock (also used for detecting bus width on parallel flashes) **Valid Entries:** 0xF8BEAF00 Bootheader for a source device other than parallel flash 0xF8BEAF08 Bootheader for 8-bit parallel flash 0xF8BEAF16 Bootheader for 16-bit parallel flash 0xF8BEAF32 Bootheader for 32-bit parallel flash
0x04	unCtrl	Depends on where the image is booted from: **Parallel Flash:** The SRAM Bus timing value (see MEMSR0_CR Register in netX Manual) **SPI Flash:** SPI clock settings (see netX Manual) netX 500/100: bits 0-3 are the value of the field CR_speed in register SPI_CR. netX 50: this DWORD contains the value of register spi_cr0. Only the fields slave_sig_early (bit 28), filter_in (bit 27) and sck_muladd (bits 19-8) are used, all other bits are masked out. **Otherwise:** Reserved, set to zero
0x08	ulAppEnt	Application entry point. This is the address the netX will start executing the binary image.
0x0C	ulAppChksum	Checksum over the application data. Generated by summing up all application data dwords.
0x10	ulAppFilSiz	Application size (in DWORDs)
0x14	ulAppSrtAdd	Physical start of the binary image. This is the address to which the netX will load the binary image
0x18	ulSignt	Bootblock signature (must be 'NETX', 0x5854454E)
0x1C	unCtrl0	Depends on destination device **SDRAM:** Register value for SDRAM General control register (see netX Manual) **Extension Bus:** Register value for EXT_CONFIG_CS0 **SRAM Bus:** Register value for EXT_SRAMn_CTRL (HBOOT/netX 50 only. The bootloader will configure one of the external SRAM blocks according to ulAppSrtAdd) **Otherwise:** Reserved, set to zero
0x20	unCtrl1	Depends on destination device **SDRAM:** Register value for SDRAM Timing register (see netX Manual) **Extension Bus:** Register value for DPMAS_IO_MODE0 (see netX Manual) **Otherwise:** Reserved, set to zero

Offset	Name	Description
0x24	unCtrl2	Depends on destination device **SDRAM:** Register value for SDRAM Mode register (see netX Manual, supported only by netX 10, 51, 52 ROM loader, ignored on netX 100, 500 and 50) **Extension Bus:** Register value for DPMAS_IO_MODE1 (see netX Manual) **Otherwise:** Reserved, set to zero
0x28	unCtrl3	Depends on destination device **Extension Bus:** Register value for DPMAS_IF_CONF0 (see netX Manual) **Otherwise:** Reserved, set to zero
0x2C	unCtrl4	Depends on destination device **Extension Bus:** Register value for DPMAS_IF_CONF1 (see netX Manual) **Otherwise:** Reserved, set to zero
0x30	ulMiscAsicCtrl	Internal ASIC control register value (set to 1)
0x34	ulSerial	Serial number or user parameter
0x38	ulSrcType	Source device type **Valid entries:** 1 parallel flash on SRAM bus 2 serial flash on SPI bus 3 serial eeprom on I2C bus 4 boot image on MMC/SD card 5 DPM boot mode 6 extended DPM boot mode 7 parallel flash on extension bus
0x3C	ulBootChksum	Checksum over the bootblock. Generated by summing up all DWORDs and multiplying the result by -1

Table 14: netX Bootblock Description

11 Flash Device Label

Flash Device Label

A Flash Device Label (in the following called "label") is a data structure stored in the flash memory on some devices, like e. g. on the Hilscher netRAPID Chip Carrier NRP 52-RE.

It contains

- identifying information about the device, like device and serial numbers
- hardware parameters, like SDRAM configuration parameters
- other information, like a MAC address.

The label is located at the very end of the flash memory and should not be erased or overwritten. Doing so may render the device useless because the firmware then may be unable to run correctly.

In order to prevent accidental erasure or overwriting of the label, Bootwizard versions 1.3.15584.0 and above check for a label before performing any flash operation. The detection is active on all netX chips and on both serial and parallel flash.

If a label is found, the area of the flash memory which contains the label is excluded from the flash operation ("protected area"). The protected area is larger than the label itself, because a flash memory device consists of a number of sections ("erase blocks") which can only be erased as a whole.

Handling of Flash Device Labels

The Bootwizard handles flash operations affecting the protected area as follows:

- If you try to write to, verify or erase an area which overlaps with the protected area, only the non-protected area will be written to, verified or erased.
- If you try to write to, verify or erase an area which is entirely contained in the protected area, the flash operation will be rejected.
- If you try to read from an area which overlaps with the protected area, two files will be written: your specified <output file>, containing the data from the non-protected area, and an additional <output file>_fdl (same name but appended with the suffix _**fdl**), which contains the remaining data from the protected area.
- If you try to read from an area which is entirely inside the protected area, only the additional output file with the suffix _**fdl** will be written.

In any of the above mentioned cases you will get notified by the Bootwizard.

Example

- The size of the flash is 0x420000 bytes.
- The size of the flash label is 194 bytes.
- The label is located in the last erase block of the flash, from 0x41ef80 – 0x41ffff. This is the protected area.

➢ If you try to erase the whole flash device, only the area from 0x0 – 0x41ef7f will be erased.

➢ If you try to erase the area from 0x41f000 to 0x41f100, the operation will be rejected.

➢ If you try to create an image of the whole flash device and specify “image.bin” as the output file name, the data from 0x0 – 0x41ef7f will be written to the “image.bin” file and the data from 0x41ef80 – 0x41ffff will be written to the “image.bin_fdl” file.

➢ If you try to read the area between 0x41ef80 – 0x41ffff and specify “image.bin” as the output file name, the data will be written to the “image.bin_fdl” file.

12 Troubleshooting

12.1 Failure to Open XML/Flasher File

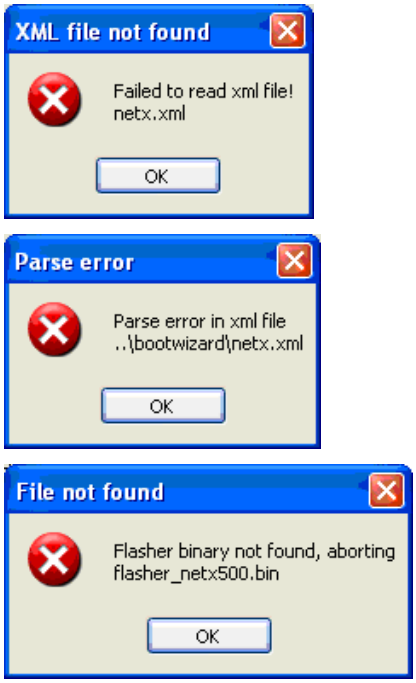

Figure 32: Bootwizard Error Opening Files

Bootwizard was not able to open (or parse) the netx.xml file or the flasher binary could not be found. Please check the configuration (see section *Customizing the Bootwizard Configuration* on page 46). If the problem persists, uninstall and re-install the Bootwizard.

12.2 Error Messages from Toolchain Scripts

The scripts in netx.xml may return error messages. The following ones may be returned by the script using the Hitex toolchain which is included in the standard distribution:

Error Message	Description
PATH_... not set	The environment variable pointing to your toolchain (e.g. PATH_GNU_ARM) is not set. Your toolchain may not be installed correctly. You may have to set this variable manually.
Can't extract binary	The call to objcopy to generate the raw binary has failed.
objdump failed	The call to objdump to extract the load and entry addresses has failed.
Can't extract load address from objdump output	The call to objdump has succeeded, but no load address was found in the output.
Can't extract entry address from objdump output	The call to objdump has succeeded, but no entry address was found in the output.
Can't parse entry address	The entry address could not be parsed.

Table 15 Error Messages from Toolchain Scripts

These errors occur when trying to generate a bootable image using a toolchain which is not installed or an ELF file which does not contain a valid start label.

Please check the following:

- Is the input file a valid ELF file?
- Did you select the correct toolchain?
- Is the selected toolchain correctly installed?
- Is a start label globally defined in your initialization code of the image?

12.3 Extracted Binary Is Too Large

Bootwizard refuses to load the image extracted from an ELF file if the image is overly large (>256 MB). Check if the ELF file contains loadable sections located in two different memory areas.

For example, if one section is assigned to the internal RAM at address 0 and another to the SDRAM at 0x80000000, the memory dump generated by objcopy will be 2 GB in size.

This problem is most likely caused by an incorrect linker configuration or an incomplete linker configuration which does not assign all sections to a memory area. The second case happens frequently in C++ projects.

12.4 Error Messages When Accessing Flash

The following messages may occur during flash operations. The table is not exhaustive.

Error Message	Possible Causes, Suggestions
Could not identify the netX chip	You are trying to use a netX type which is not supported by this version of Bootwizard.
You've selected the wrong chip type. Selected: x Detected: y	You have selected the wrong chip type.
Operation cancelled by user.	You clicked on the Cancel button during a flash operation.
Unknown/unsupported flash type	The selected memory interface does not exist on the chip type.
Failed to load flasher binary	The flasher binary could not be found or could not be loaded. Open the config window. If the internal flasher is selected for the respective chip type, re-install Bootwizard. If you have selected another flasher file, check if the file is present.
Error while downloading flasher binary	There may be a hardware problem with the PC-netX connection, with your netX hardware or with the driver installation on the PC side.
Failed to get a device description	The flasher cannot identify the flash device. The flash device or some other part of the hardware is defective. No flash device is connected to the selected memory interface.
Other messages	There may be a hardware problem with the PC-netX connection, with your netX hardware or with the driver installation on the PC side.

Table 16 Error Messages During Flash Access

General suggestions:

- Flash detection: Repeat the operation using the debug flasher
- Driver / connection problems: Try using a different interface (COM, USB, JTAG)
- Hardware problems: Try a second hardware of the same kind or a standard netX hardware (e.g. NXHX)

12.5 Using the Debug flashers

There is a version of each flasher which produces extended output.

- Open the config window.
- Uncheck **Use internal file**.
- Click the Browse button, navigate to the Bootwizard installation, enter the subdirectory "bootwizard" and select the appropriate debug flasher binary.
- Click the **Ok** button.

In order to return to the normal flasher, check **Use internal file**.

12.6 Saving the Message Log

The bottom area of the Bootwizard window, which is titled **Message Log** contains log information.

- If this area is turned off, enable it by opening the **View** menu and checking **View message log**.
- Right-click in the log area and choose **Select all** from the menu.
- Right-click again and choose **copy**.
- Open a text editor, paste the text and save it to a file.

12.7 Collision Between SPI Flash and MMC/SD Card

If you encounter any problem accessing SPI flash memory and your board has an MMC/SD card connector, check if a card is inserted. In this case, remove the card.

13 Appendix

13.1 List of Tables

13.2 List of Figures

13.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
New Delhi - 110 065
Phone: +91 11 43055431
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Suwon, Gyeonggi, 443-734
Phone: +82 (0) 31-695-5515
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com