NAND FLASH Programming User´s Guide

©1989-2014 Lauterbach GmbH

©1989-2014 Lauterbach GmbH

©1989-2014 Lauterbach GmbH

Version 11-Nov-2014

Introduction

This manual describes the basic concept of NAND and OneNAND Flash programming.

There are many similarities between NAND Flash programming and OneNAND Flash, but also important differences. For reasons of clarity and user-friendliness, this manual covers NAND Flash programming and OneNAND Flash programming in separate chapters.

How This Manual is Organized

- **Background Information**: Provides information about important terms in NAND Flash programming, including the different types of NAND Flash controllers (NFC).
- **Standard Approach**: Describes the fastest way to get started with NAND Flash programming. All you need to do is to identify and run the correct script.

  Demo scripts for NAND Flash programming are provided on the TRACE32 software CD:

  `demo/<architecture>/flash/<cpu_name>-<nand_flash_code>.cmm`

  e.g. `omap3430-nand.cmm`, `imx31-nand2g08.cmm` …
- **Scripts for NAND Flash Programming**: Describes how you can create a script if there is no demo script for the NFC type you are using.
- **About OneNAND Flash Devices**: Explains the difference between OneNAND Flash and NAND Flash.
- **Scripts for OneNAND Flash Devices**: Describes how you can create scripts for OneNAND Flash programming based on the template script provided by Lauterbach.
- **Appendix**: Provides information about ECC (error correction code) and spare area schemes.

©1989-2014 Lauterbach GmbH

Related Documents

A complete description of all NAND Flash programming commands can be found in chapter **"FLASHFILE"** in **"General Commands Reference Guide F"** (general_ref_f.pdf).

The manual **"List of Supported FLASH Devices"** (flashlist.pdf) provides the following information:

- A list of the supported NAND and OneNAND Flash devices.
- A list of the supported on-chip NAND Flash controllers.

The Lauterbach home page provides an up-to-date list of

- Supported NAND and OneNAND Flash devices under:
  http://www.lauterbach.com/ylist.html
- Supported on-chip NAND Flash controllers under:
  http://www.lauterbach.com/ylistnand.html

Contacting Support

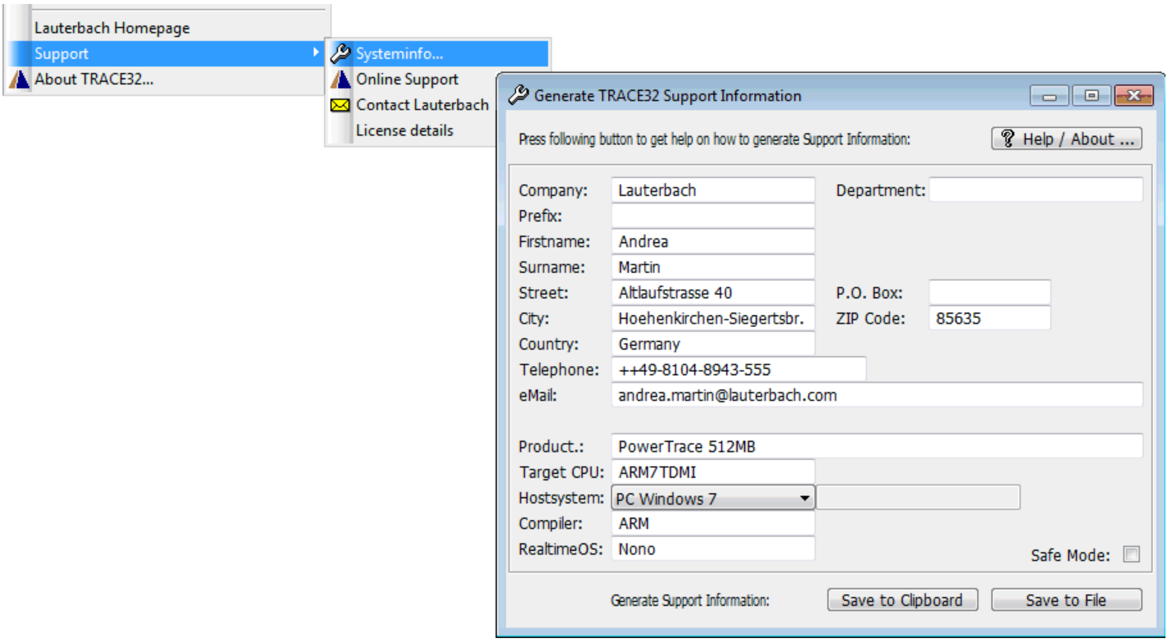
LAUTERBACH GmbH
Altlaufstrasse 40
85635 Hoehenkirchen-Siegertsbrunn
Germany

Phone	(+49) 8102-9876-555
Fax	(+49) 8102-9876-187
Internet	http://www.lauterbach.com/tsupport.html or http://www.lauterbach.com/report.html Here you'll find local and special support addresses.
E-mail	support@lauterbach.com General support address where your request will be answered within a short time if it is a basic support request or redirected to the appropriate address.

Be sure to include detailed system information about your TRACE32 configuration.

1. To generate a system information report, choose **TRACE32** > **Help** > **Support** > **Systeminfo**.

©1989-2014 Lauterbach GmbH



NOTE:	Please help to speed up processing of your support request. By filling out the system information form completely and with correct data, you minimize the number of additional questions and clarification request e-mails we need to resolve your problem.

2. Preferred: click **Save to File**, and send the system information as an attachment to your e-mail.

3. Click **Save to Clipboard**, and then paste the system information into your e-mail.

NOTE:	In case of missing script files (`*.cmm`), please proceed as requested in “**If There is No Script**”.

©1989-2014 Lauterbach GmbH

List of Abbreviations

ALE	Address latch enable
CLE	Command latch enable
CS	Chip selection
ECC	Error correction code
NFC	NAND Flash controller
SP	Spare area

Background Information

This chapter of the manual is aimed at users who are new to NAND Flash programming; it does not address experts with many years of expertise in this area. This chapter gives you a brief overview of important terms in NAND Flash programming, such as NAND Flash device, block, page, main area, spare area, bad block marker, generic NFC, CPU-specific NFC.

What is a NAND Flash Device?

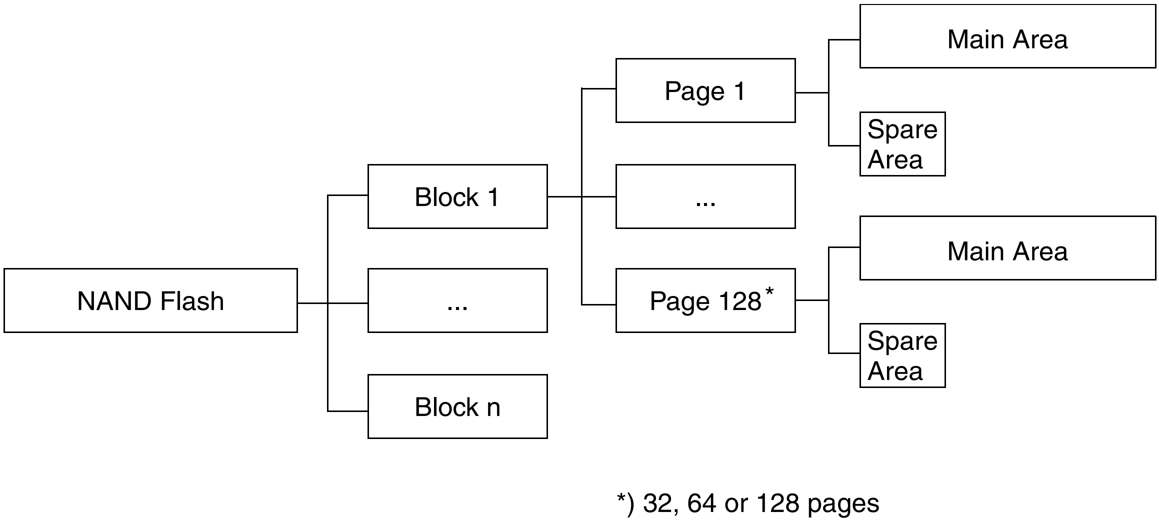
A NAND Flash device (short: NAND Flash) is a non-volatile storage chip that can be electrically erased and reprogrammed. It is used in data-storage applications such as cell phones and multi-media devices. Reasons why NAND Flash devices have become widespread include:

- Smaller interface pins than NOR Flash
- High density at low-cost per bit
- Faster than NOR Flash

©1989-2014 Lauterbach GmbH

About Blocks, Pages, Main Area, and Spare Area

A NAND Flash consists of blocks. Each block is subdivided into 32, 64, or 128 pages, and each page has a main and a spare area; see example diagram below.



Block A block is the minimum size unit for erasing.

Page A page is the minimum size unit for reading and writing.
There are two types of pages:

- Small pages
- Large pages

Type	Main Area*	Spare Area*	Total*
Small Page	256	8	264
	512	16	528
Large Page	2048	64	2112
	4096	128	4224
	*) in Bytes		

Main area The main area of each page can have a size of 512, 2048, or 4096 Bytes and contains the real code or data.

Spare area The spare area of each page can have a size of 16, 32, 64, or 128 Bytes and contains the following:

- Bad block marker for a bad block (mandatory)
- ECC codes (optional)
- User-specific metadata (optional)

©1989-2014 Lauterbach GmbH

About Bad Block Markers

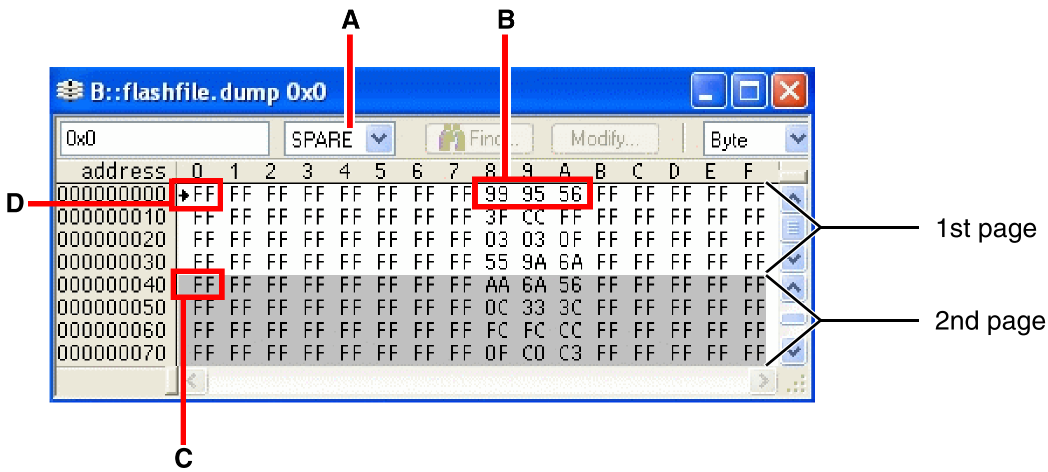
If a block is bad, then data cannot be erased or read from or written to the bad block. To flag a bad block, one or two bad block markers are used:

- The 1st marker is located in the spare area of the 1st page.
- The 2nd marker is located in the spare area of the 2nd page.

Bad block markers are stored in different byte positions, depending on the type of page (large or small):

- **Large page NAND**: The bad block marker is stored in the 1st byte.
- **Small page NAND**: The bad block marker is stored in the 6th byte.

The figure below shows the 64-byte spare areas of the first two pages of a large page NAND. The **FLASHFILE.DUMP** window visualizes the individual pages using alternating colors for pages - white and gray.



A	Spare area of a large page NAND
B	ECC code
C, **D**	• FF = The block that these first two pages (white and gray) belong to is intact. • If (**C**) or (**D**) or both do *not* read FF, as shown above, then the system considers the block to be bad. Byte position of a 1st bad block marker in the 1st page = (**D**). Byte position of a 2nd bad block marker in the 2nd page = (**C**).

NOTE:	The **/EraseBadBlocks** option of the **FLASHFILE.Erase** command can only erase faked bad blocks, but not real bad blocks. A faked bad block is a block where the user has modified an FF value to a non-FF value in the byte position (**C**) or (**D**) or in both byte positions.

©1989-2014 Lauterbach GmbH

About NAND Flash Controllers

Access to the NAND Flash is performed by an on-chip NAND Flash controller. There are two types of NAND Flash controllers (NFC):

- **Generic NAND Flash controllers**
  These NFC types are typically manufactured by Samsung Semiconductor, Atmel Corporation, STMicroelectronics, Marvell, Inc., and Texas Instruments.

- **CPU-specific NAND Flash controllers**
  These NFC types are typically manufactured by Qualcomm, Freescale Semiconductor, NVIDIA Corporation, and Renesas Technology, Corp.

The architecture of systems featuring generic NFCs is shown in the block diagram below.

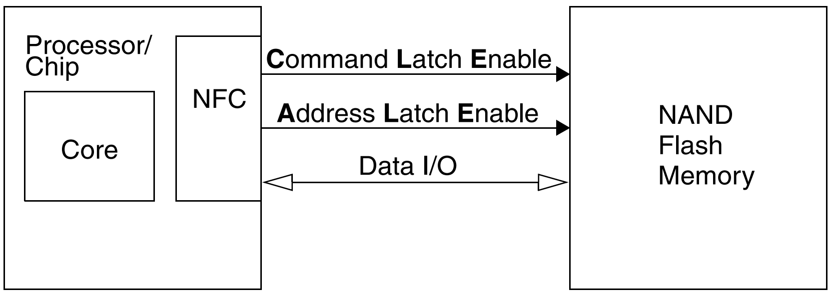


Figure: System with a Generic NAND Flash Controller (NFC)

The architectures of systems featuring CPU-specific NFCs may vary considerably. The following block diagram illustrates an example of a typical architecture. Data from/to the NAND Flash is buffered in a data buffer.

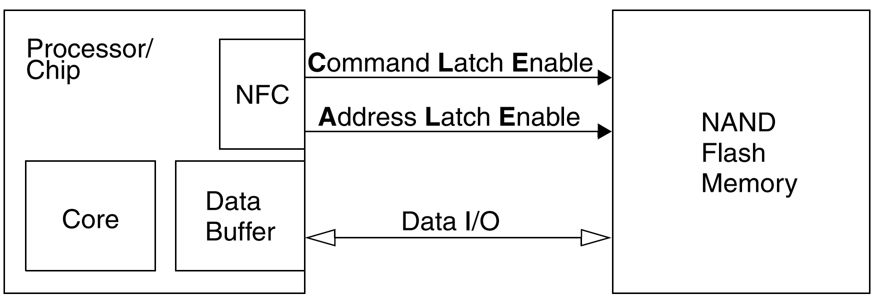


Figure: Example of a System with a CPU-specific NAND Flash Controller (NFC)

©1989-2014 Lauterbach GmbH

Standard Approach

Standard Approach provides a compact description of the steps required to program NAND Flash memory. This description is intentionally restricted to standard use cases.

A detailed description of the NAND Flash programming concepts is given in“**Scripts for NAND Flash Programming**”.

Identifying and Running Scripts for NAND Flash Programming

Demo scripts (*.cmm) for NAND Flash programming are provided by Lauterbach. They can be found in the TRACE32 installation directory, which contains scripts developed for generic and CPU-specific NFC types.

Path and file name convention of scripts for generic and CPU-specific NFC types:

```
<TRACE32_installation_directory>/
       demo/<architecture>/flash/<cpu_name>-<nand_flash_code>.cmm
```

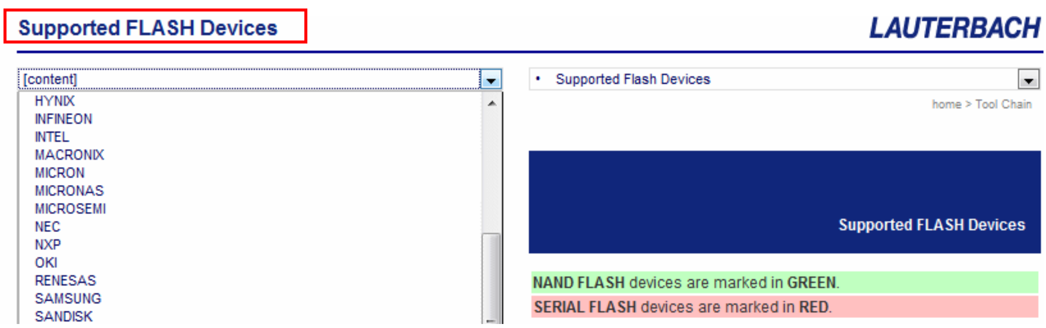
To identify and run the required script:

1. Make a note of the **<cpu_name>** printed on the CPU; for example, **at91sam9xe**

2. For information about supported Flash devices, access the **Lauterbach** website.

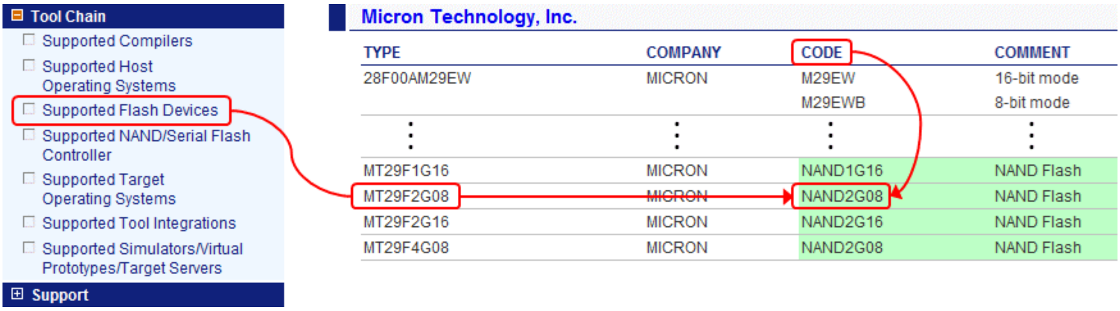
3. Click the **+** drill-down button next to **Tool Chain**, and then click **Supported Flash Devices** (**http://www.lauterbach.com/ylist.html**).

4. On the **Supported Flash Devices** page, select the required company from the drop-down list.



5. Use the type printed on the Flash device to retrieve the **<nand_flash_code>** from the web page.

   For example, NAND Flash type = MT29F2G08



   Result: **<nand_flash_code>** = nand2g08

6. Put the **<cpu_name>** and the **<nand_flash_code>** together to form the script name: **at91sam9xe-nand2g08**.cmm

   The script file resides in this folder: ~~/demo/**arm**/flash/**at91sam9xe-nand2g08**.cmm

   Whereas ~~ is expanded to the <TRACE32_installation_directory>, which is c:/T32 by default.

   If the folder does not contain the script you are looking for, see "**If There Is No Script**" on **page 13**.

7. Run the script in TRACE32 by doing one of the following:

   - Choose **File** > **Run Batch File** <script name>
   - In the command line, type DO <path and script name>

NOTE:	Each script (*.cmm) includes a reference to the required NAND Flash programming algorithm (***.bin**). You do not need to program or select the algorithm.

Example

```
;                <code_range>      <data_range>      <algorithm file>
FLASHFILE.TARGET 0x80008000++0x3fff 0x8000C000++0x4FFF
                                 ~~/demo/arm/flash/byte/nand2g08_imx.bin
```

©1989-2014 Lauterbach GmbH

If There Is No Script

If there is no script for your device in this directory (<TRACE32_installation_directory>/demo/ **<architecture>**/flash/), then please send a request to **nandflash-support@lauterbach.com** using the e-mail template below.

E-Mail Template:

Chip name: ______________________

Name of NAND Flash device: ________

Provide the CPU datasheet for us: ___________

Lend the target board to us by sending it to the address given in “**Contacting Support**”: ________

`<SystemInformation>`

Be sure to include detailed system information about your TRACE32 configuration. For information about how to create a system information report, see “**Contacting Support**”.

Normally we can provide support for a new device in two weeks.

If our support cannot provide you with a script, you will have to create your own script (`*.cmm`).

For more information, see “**Scripts for NAND Flash Programming**” on **page 14**.

Scripts for NAND Flash Programming

This chapter describes how you can create your own scripts for chips that are equipped with generic or CPU-specific NAND Flash controllers.

The steps and the framework (see below) provide an overview of the process. Both, steps and framework, are described in detail in the following sections.

The following steps are necessary to create a new script:

1. Establish communication between debugger and target CPU.
2. Configure the NAND Flash controller.
3. Reset the NAND Flash environment in TRACE32 to its default values.
4. Identify the type of NAND Flash controller using the Lauterbach website.
5. Inform TRACE32 about the NAND Flash register addresses (Flash declaration).
6. Inform TRACE32 about the NAND Flash programming algorithm.
7. Check the identification from the NAND Flash device.
8. Erase the NAND Flash device.
9. Program the NAND Flash device.

©1989-2014 Lauterbach GmbH

The following framework can be used as base for NAND Flash programming:

```
                                  ; Establish the communication
                                  ; between the target CPU and the
                                  ; TRACE32 debugger.

                                  ; Configure the NAND Flash
                                  ; controller.

FLASHFILE.RESet                   ; Reset the NAND Flash environment
                                  ; in TRACE32 to its default values.

FLASHFILE.CONFIG …                ; Inform TRACE32 about the
                                  ; NAND Flash register addresses.

FLASHFILE.TARGET …                ; Specify the NAND Flash
                                  ; programming algorithm and where
                                  ; it runs in the target RAM.

FLASHFILE.GETID                   ; Get the ID values of the NAND
                                  ; Flash device.

FLASHFILE.Erase …                 ; Erase the NAND Flash.

FLASHFILE.LOAD <main_file> …      ; Program the file to the NAND
                                  ; Flash (main area).
```

An ellipsis (…) in the framework indicates that command parameters have been omitted here for space economy.

NOTE:	The parametrization of **FLASHFILE.CONFIG** and **FLASHFILE.TARGET** requires expert knowledge.

©1989-2014 Lauterbach GmbH

Establishing Communication between Debugger and Target CPU

NAND Flash programming with TRACE32 requires that the communication between the debugger and the target CPU is established. The following commands are available to set up this communication:

SYStem.CPU *<cpu>*	Specify your target CPU.
SYStem.Up	Establish the communication between the debugger and the target CPU.

```
SYStem.CPU OMAP3430      ; Select OMAP3430 as the target CPU.

SYStem.Up                ; Establish the communication between the
                         ; debugger and the target CPU.
```

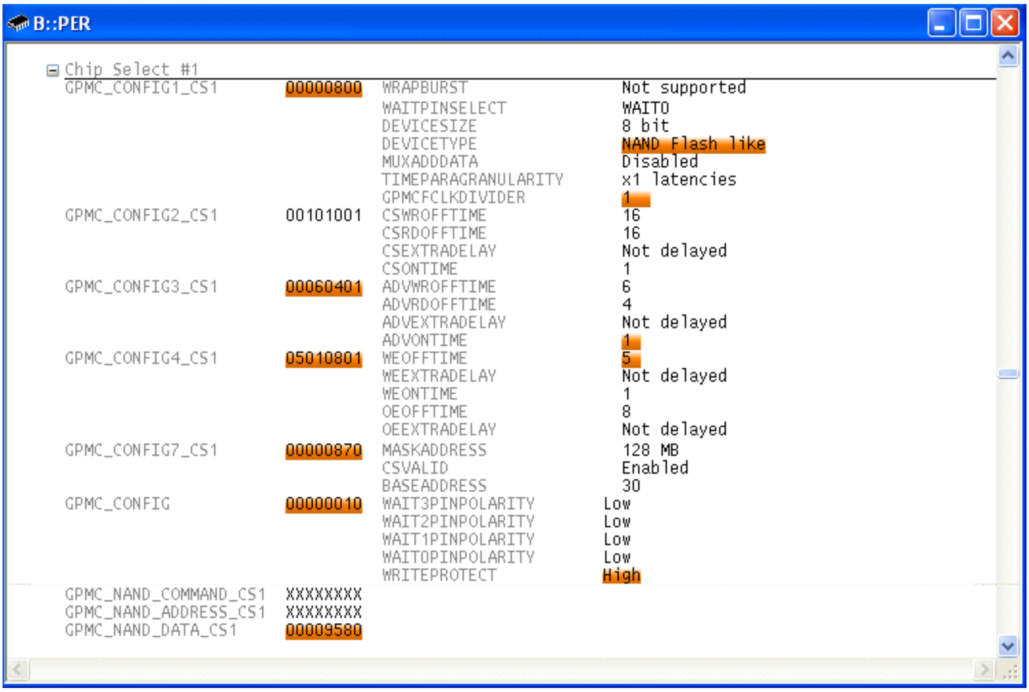
Configuring the NAND Flash Controller

Programming a NAND Flash device requires a proper initialization of the NAND Flash controller. The following settings might be necessary:

- Enable the NAND Flash controller or bus.
- Configure the communication signals (clock, timing, etc.).
- Inform the NAND Flash controller about the NAND Flash device (large/small page, ECC, spare, etc.).
- Configure the NAND Flash pins if they are muxed with other functions of the CPU.
- Disable the write protection for the NAND Flash.

©1989-2014 Lauterbach GmbH

Use the **PER.view** command to check the settings for the NAND Flash controller.



Example: NAND Flash controller configuration for the OMAP3430.

```
PER.Set SD:0x6E0000A8 %LE %Long 0x870          ; Enable CS1 and define
                                               ; the base address of
                                               ; CS1(NAND Flash).
                                               ; LE = little endian

PER.Set SD:0x6E000098 %LE %Long 0x60401        ; Define the NAND Flash
PER.Set SD:0x6E00009C %LE %Long 0x5010801      ; access timing.

PER.Set SD:0x6E000090 %LE %Long 0x0800         ; Define CS1 for 8 bit
                                               ; NAND Flash.

PER.Set SD:0x6E000050 %LE %Long 0x10           ; Disable the write
                                               ; protection of the NAND
                                               ; Flash device.
```

©1989-2014 Lauterbach GmbH

Resetting Default Values

The following command is used to reset the NAND Flash environment in TRACE32 to its default values.

FLASHFILE.RESet	Reset the NAND Flash environment in TRACE32 to its default values.

©1989-2014 Lauterbach GmbH

Identifying the Type of Controller

You need to know which NFC type you are dealing with because NAND Flash programming differs for the NFC types:

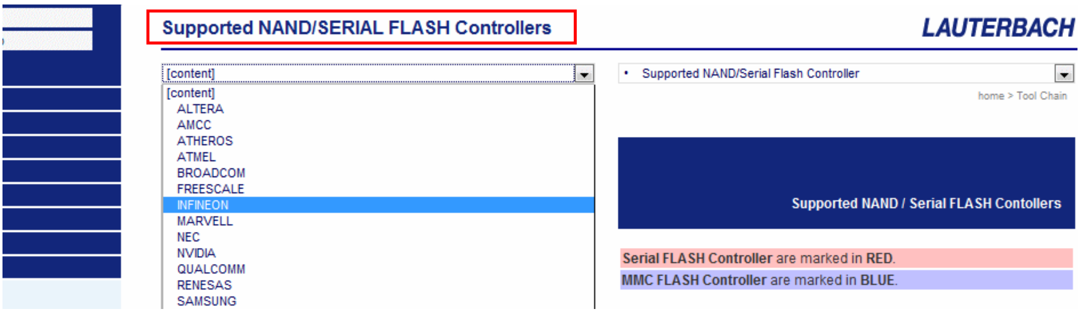
- Generic NAND Flash controllers
- CPU-specific NAND Flash controllers

To identify the type of controller:

1. Access the **Lauterbach** website.

2. Click the **+** drill-down button next to **Tool Chain**, and then click **Supported NAND/Serial Flash Controller**.

3. Select the required company from the drop-down list.

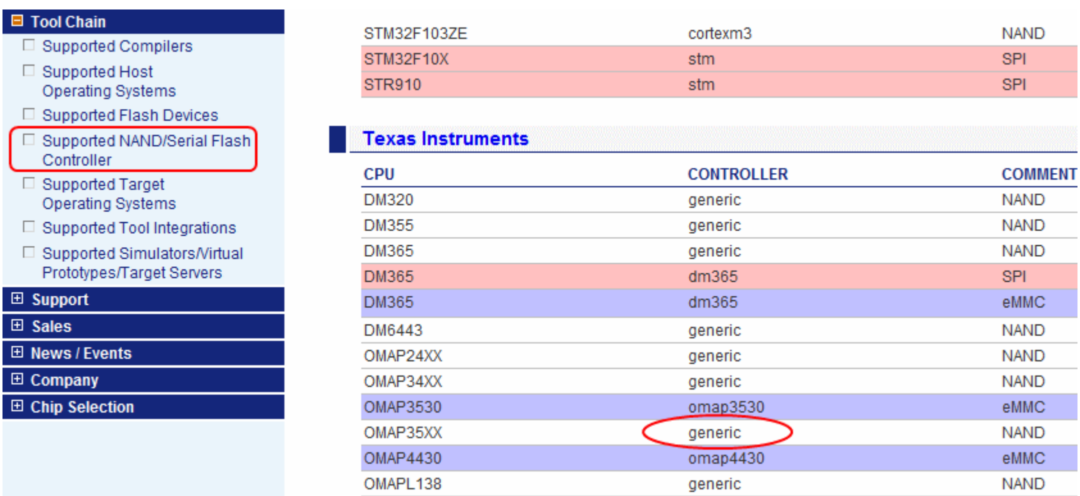


4. Locate the desired CPU.

   The **Controller** column indicates whether the NFC type is generic or CPU-specific or a hybrid. The following three examples cover all possible options.

   Example 1: CPU = OMAP3530

   The entry in the **Controller** column reads *generic*, and the entry in the **Comment** column reads *NAND*. That means that this CPU is equipped with a generic NAND Flash controller.

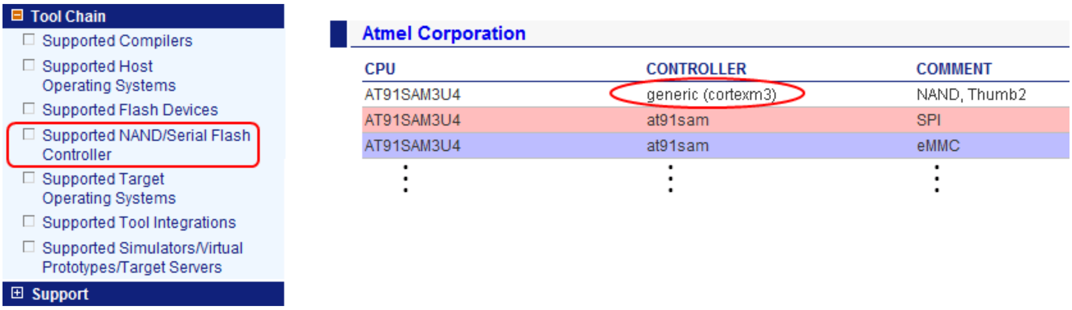


©1989-2014 Lauterbach GmbH

Example 2: CPU = AT91SAM3U4

The entry in the **Controller** column reads *generic (cortexm3)*, and the entry in the **Comment** column reads *NAND, Thumb2*.

That means that this CPU is equipped with a generic NAND Flash controller, too. The term in parentheses tells you the architecture of the processor core, here *(cortexm3)*. This processor core requires that the NAND Flash driver binary file is compiled using a special instruction set, here *Thumb2*.



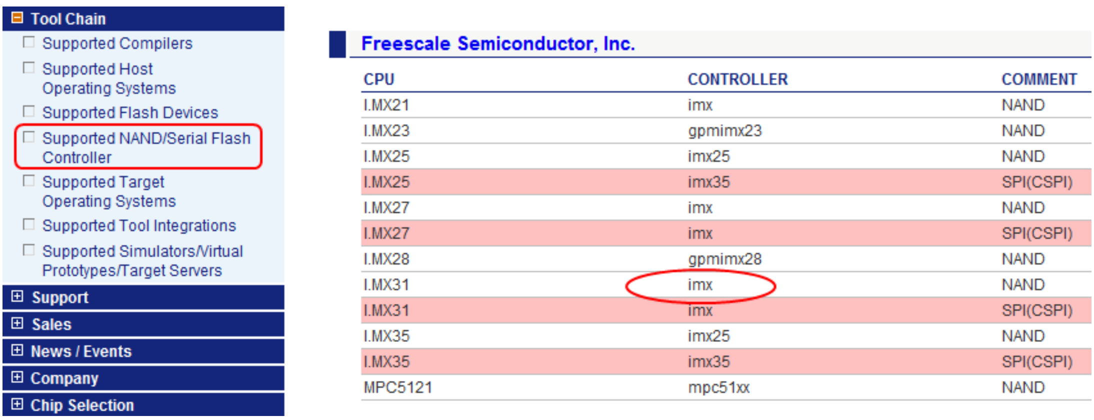
- Tool Chain
  - Supported Compilers
  - Supported Host Operating Systems
  - Supported Flash Devices
  - Supported NAND/Serial Flash Controller
  - Supported Target Operating Systems
  - Supported Tool Integrations
  - Supported Simulators/Virtual Prototypes/Target Servers
- Support

Atmel Corporation

CPU	CONTROLLER	COMMENT
AT91SAM3U4	generic (cortexm3)	NAND, Thumb2
AT91SAM3U4	at91sam	SPI
AT91SAM3U4	at91sam	eMMC
⋮	⋮	⋮

Example 3: CPU = I.MX31

The entry in the **Controller** column contains the controller name (*imx*), and the entry in the **Comment** column reads *NAND*. That means that this CPU is equipped with a CPU-specific NAND Flash controller.



- Tool Chain
  - Supported Compilers
  - Supported Host Operating Systems
  - Supported Flash Devices
  - Supported NAND/Serial Flash Controller
  - Supported Target Operating Systems
  - Supported Tool Integrations
  - Supported Simulators/Virtual Prototypes/Target Servers
- Support
- Sales
- News / Events
- Company
- Chip Selection

Freescale Semiconductor, Inc.

CPU	CONTROLLER	COMMENT
I.MX21	imx	NAND
I.MX23	gpmimx23	NAND
I.MX25	imx25	NAND
I.MX25	imx35	SPI(CSPI)
I.MX27	imx	NAND
I.MX27	imx	SPI(CSPI)
I.MX28	gpmimx28	NAND
I.MX31	imx	NAND
I.MX31	imx	SPI(CSPI)
I.MX35	imx25	NAND
I.MX35	imx35	SPI(CSPI)
MPC5121	mpc51xx	NAND

©1989-2014 Lauterbach GmbH

Informing TRACE32 about the NAND Flash Register Addresses

The parametrization of **FLASHFILE.CONFIG** differs for generic and CPU-specific NFCs.

In the case of generic NAND Flash controllers:

The NAND Flash device can be programmed by operating the command, address, and I/O registers. As a result:

1. A generic NAND Flash programming driver can be used.
2. The command **FLASHFILE.CONFIG** always requires the parameters *<cmd_reg> <addr_reg> <io_reg>*

FLASHFILE.CONFIG *<cmd_reg> <addr_reg> <io_reg>*	Inform TRACE32 about the NAND Flash register addresses.

Parameters for FLASHFILE.CONFIG command – generic NAND Flash programming	
<cmd_reg>	Register address of the command register
<addr_reg>	Register address of the address register
<io_reg>	Register address of the data I/O register

For information about the register addresses of the command, address, and data I/O register, refer to the manufacturer's processor manual.

Example 1

```
; Register addresses of the generic NAND Flash controller in the OMAP3530
FLASHFILE.CONFIG 0x6E00007C 0x6E000080 0x6E000084
```

Example 2

```
; Register addresses of the generic NAND Flash controller in the OMAP3430
FLASHFILE.CONFIG 0x6E0000AC 0x6E0000B0 0x6E0000B4
```

In the case of CPU-specific NAND Flash controllers:

FLASHFILE.CONFIG *<NFC_base_address> , ,*	Specify the start address of the NAND Flash base register. , represents don't-care parameters.

For information about the NAND Flash base register, refer to the manufacturer's processor manual.

©1989-2014 Lauterbach GmbH

Example

```
; NFC base address of the CPU-specific NAND Flash controller
; in the i.MX31.
FLASHFILE.CONFIG 0xB8000000 , ,
```

Informing TRACE32 about the NAND Flash Programming Algorithm

The following command is available to inform TRACE32 about the NAND Flash programming algorithm:

FLASHFILE.TARGET *<code_range> <data_range> <file>*	Specify the NAND Flash programming algorithm and where it runs in the target RAM.

Parameters

- *<code_range>*

  Define an address range in the target´s RAM to which the NAND Flash programming algorithm is loaded.

Flash algorithm
32 byte

  Figure: Memory mapping for the *<code_range>*

  Required size for the code is `size_of(`*<file>*`) + 32 byte`

©1989-2014 Lauterbach GmbH

- *<data_range>*

  Define the address range in the target´s RAM where the programming data is buffered for the programming algorithm.

64 byte argument buffer
buffer for programming data
256 byte stack

Figure: Memory mapping for the *<data_range>*

The argument buffer used for the communication between the TRACE32 software and the programming algorithm is located at the first 64 bytes of *<data_range>*. The 256 byte stack is located at the end of *<data_range>*.

```
<buffer_size> =
size_of(<data_range>) - 64 byte argument buffer - 256 byte stack
```

<buffer_size> is the maximum number of bytes that are transferred from the TRACE32 software to the NAND Flash programming algorithm in one call.

- *<file>*

  Lauterbach provides ready-to-run driver binary files for NAND Flash programming. They are located in the TRACE32 installation directory:

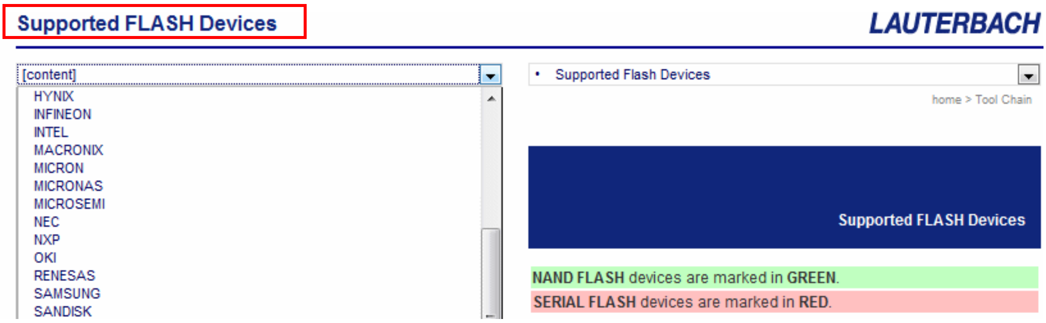
  ~~/demo/<architecture>/flash/<bus_width>/

  Whereas ~~ is expanded to the <TRACE32_installation_directory>, which is c:/T32 by default.

  For detailed information about how to determine the *<file>* parameter, see "**Identifying the Correct Driver Binary File for a NAND Flash Device**" on **page 24**.

Identifying the Correct Driver Binary File for a NAND Flash Device

1. For information about supported Flash devices, access the **Lauterbach** website.
2. Click the **+** drill-down button next to **Tool Chain**, and then click **Supported NAND/Serial Flash Controller** (**http://www.lauterbach.com/ylistnand.html**).
3. Open **Supported Flash Devices** in a separate window or tab (**http://www.lauterbach.com/ylist.html**).
4. On the **Supported Flash Devices** page, select the required company from the drop-down list.



5. Locate the desired Flash device.

   You need the name of the Flash device to be able to identify the correct driver binary file.

   The file name convention for driver binary files is explained below. In addition, a few examples illustrate how to apply the file name convention in practice.

File Name Convention for NAND Flash Drivers

The NAND Flash drivers for the various NFC types use the following file name convention:

Page Size (bytes)		Block Size	Device Size	Bus Width	File Name
Main area	Spare area				
512	16	32 pages	<= 2048 blocks	8	Nand5608.bin
				16	Nand5616.bin
512	16	32 pages	> 2048 blocks	8	Nand1208.bin
				16	Nand1216.bin
2048	64	64 pages	<= 1024 blocks	8	Nand1g08.bin
				16	Nand1g16.bin
2048	64	64 pages	> 1024 blocks	8	Nand2g08.bin
				16	Nand2g16.bin
2048	64	128 pages	> 1024 blocks	8	NandLAg08.bin
4096	128	64 pages	> 1024 blocks	8	Nand8g08.bin
4096	218				Nand8g08xs.bin
4096	128	128 pages	> 1024 blocks	8	NandLBg08.bin
4096	218				NandLBg08xs.bin

“xs” = eXtra spare area

Examples for Generic NFCs

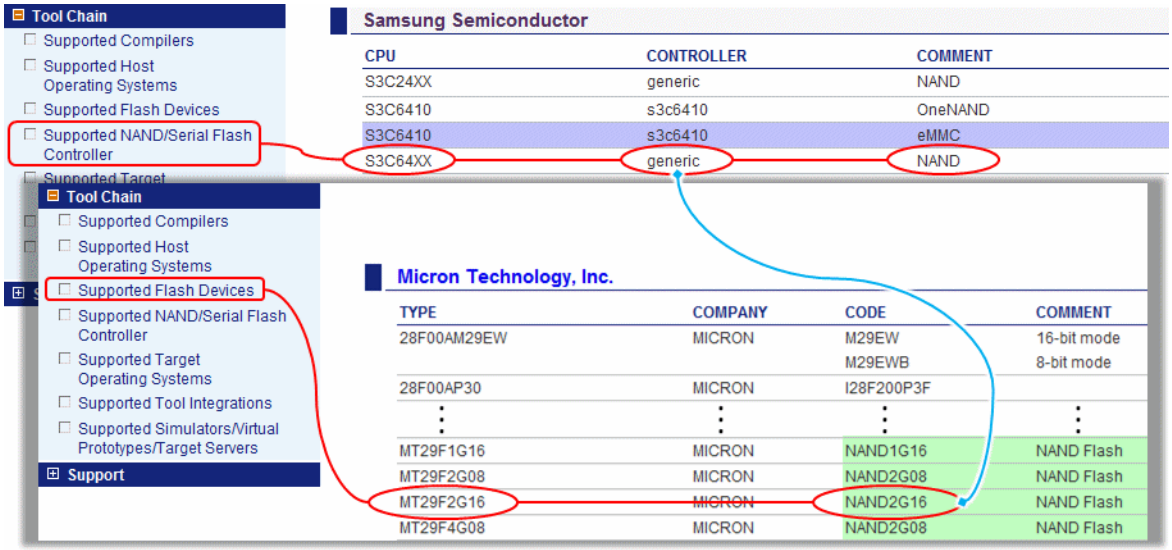
The names of the required NAND Flash driver binary files consist of information from the **Controller** and/or **Code** columns. The following example illustrate how you can combine this information from the Lauterbach website to form the correct file name.

Example 1 – target:

- CPU **S3C6410** with generic NFC
- NAND Flash device **MT29F2G16**

©1989-2014 Lauterbach GmbH

The **Code** column identifies the name of the NAND Flash driver binary file: `nand2g16.bin`.
Note that the information in the **Controller** column is *not* part of the file name in this case.



The number **`16`** in the file name indicates the bus width and the folder where the file resides, i.e. in the **`word`** folder.

The binary file resides in this folder: **`~~/demo/arm/flash/word`**

Whereas `~~` is expanded to the <TRACE32_installation_directory>, which is `c:/T32` by default.

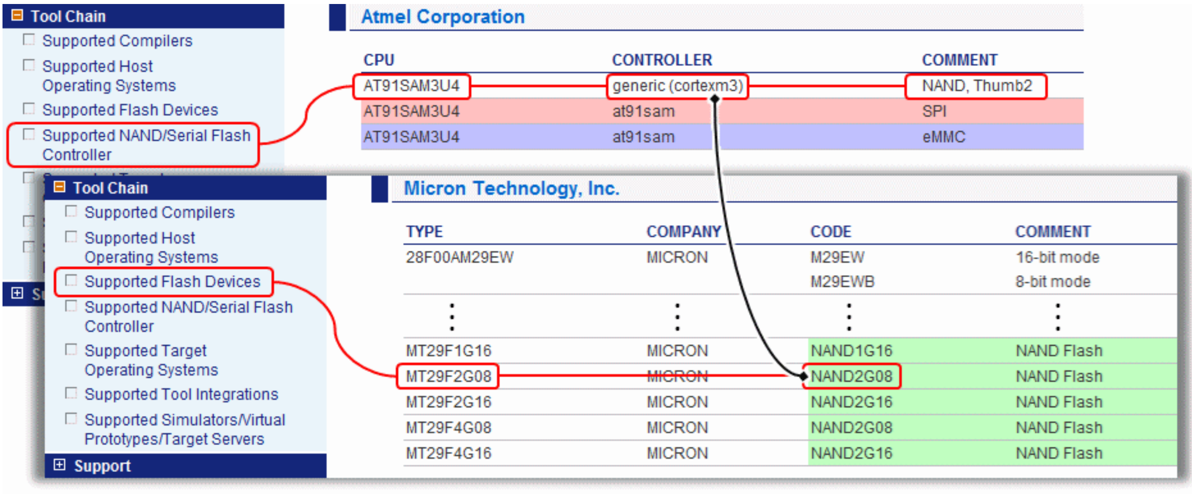
Example 2 – target:

- CPU **AT91SAM3U4** with generic (cortexm3) NFC.

  Remember that NFCs flagged like this in the **Controller** column—*generic (name)*—require binary files that are compiled with a special instruction set, here *Thumb2*; see figure below.

- NAND Flash device **MT29F2G08**

Taken together, the **Code** column and the **Controller** column make up the file name of this particular NAND Flash driver binary file: **`nand2g08_cortexm3.bin`**



The number **`8`** in the file name indicates the bus width and the folder where the file resides, i.e. in the **`word`** folder.

The binary file resides in this folder: **`~~/demo/arm/flash/byte`**

Whereas `~~` is expanded to the `<TRACE32_installation_directory>`, which is `c:/T32` by default.

This results in the following command line:

```
; Specify the NAND Flash programming algorithm and where it runs in
; the target RAM.    <code_range>       <data_range>           <file>
FLASHFILE.TARGET 0x20000000+0x1fff 0x20002000++0x1fff
                                   ~~/demo/arm/flash/byte/nand2g08_cortexm3.bin
```

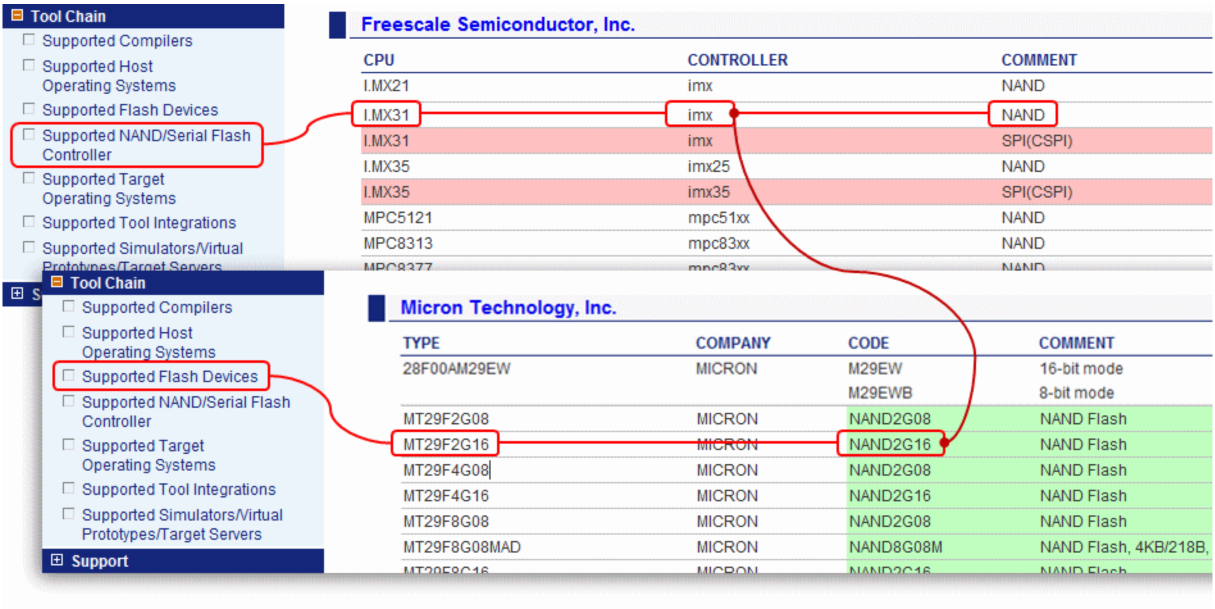
Example for CPU-Specific NFCs

Target:

- CPU **i.MX31** with a CPU-specific controller
- NAND Flash device **MT29F2G16**

©1989-2014 Lauterbach GmbH

Taken together, the **Code** column and the **Controller** column make up the file name of the NAND Flash driver binary file: **nand2g16_imx.bin**



The number **16** indicates the bus width and the folder where the file resides, i.e. in the **word** folder.

The file resides in this folder: **~~/demo/arm/flash/word**

Whereas ~~ is expanded to the <TRACE32_installation_directory>, which is c:/T32 by default.

Checking the Identification from the NAND Flash Device

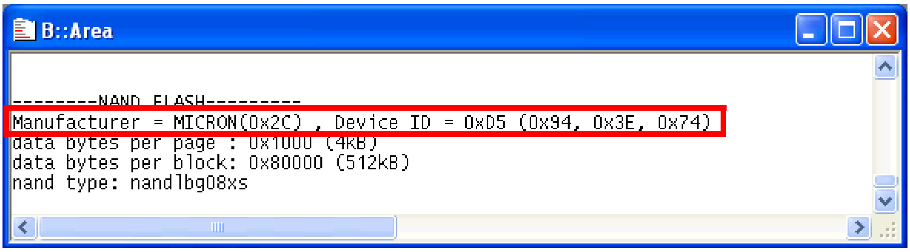
The following command can be used to check if TRACE32 can access the NAND Flash device:

FLASHFILE.GETID	Get the ID values, page size, block size, and the NAND Flash code from the NAND Flash device.

```
; Open the TRACE32 AREA window.
AREA.view

; Get the ID values, page size, block size, and the NAND Flash code
; from the NAND Flash device.
FLASHFILE.GETID
```

©1989-2014 Lauterbach GmbH


B::Area
--------NAND FLASH---------
Manufacturer = MICRON(0x2C) , Device ID = 0xD5 (0x94, 0x3E, 0x74)
data bytes per page : 0x1000 (4kB)
data bytes per block: 0x80000 (512kB)
nand type: nand1bg08xs

Erasing the NAND Flash Device

The following commands are available to erase NAND Flash devices:

FLASHFILE.Erase *<range>*	Erase NAND Flash except bad blocks.

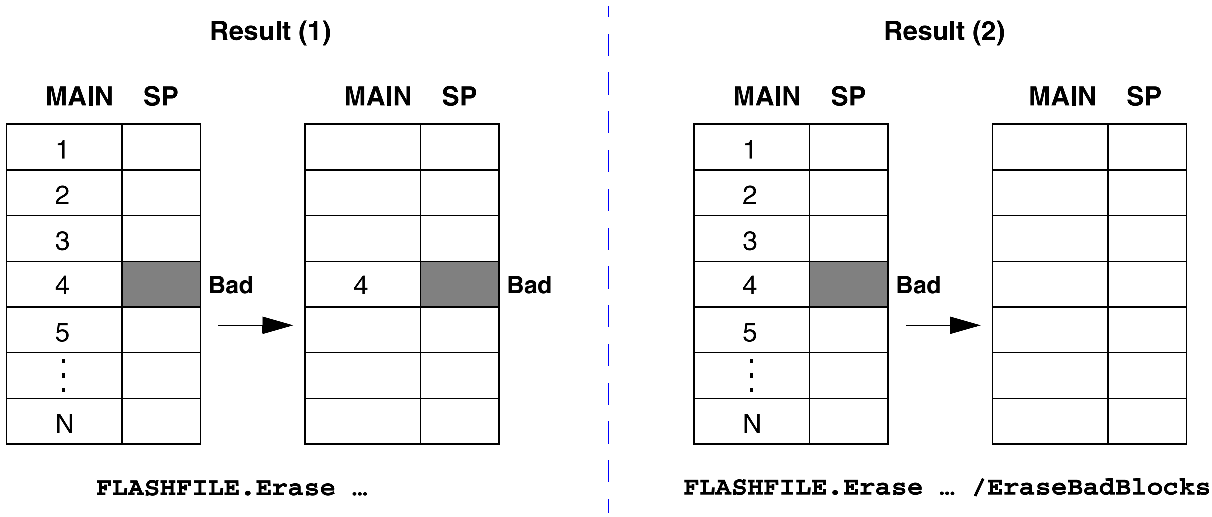
FLASHFILE.Erase *<range>* **/EraseBadBlocks**	Erase NAND Flash including bad blocks.

Example 1

```
; Erase 1MB starting from 0x0 except bad blocks.
FLASHFILE.Erase 0x0--0xFFFFF
```

Example 2

```
; Erase 1MB starting from 0x0 including bad blocks.
; Afterwards all bad block data is erased.
FLASHFILE.Erase 0x0--0xFFFFF /EraseBadBlocks
```



©1989-2014 Lauterbach GmbH

In a NAND Flash device, each page consists of two areas:

- The **main area** contains the data which is accessed by the CPU.
- The **spare area** contains the bad block information and the ECC data.
  For background information about ECC, see “**Appendix: ECC (Error Correction Code)** on **page 75**.

The main and spare area are programmed independently.

All CPU-specific NAND Flash controllers generate the ECC data automatically when data is programmed to the main area. Therefore, the spare area does not need to be programmed explicitly.

©1989-2014 Lauterbach GmbH

The following commands are available to program the NAND Flash main area:

FLASHFILE.LOAD *<file>* [*<address>* | *<range>*]	Program NAND Flash except bad blocks.

FLASHFILE.LOAD *<file>* [*<address>* | *<range>*] **/WriteBadBlocks**	Program NAND Flash including bad blocks.

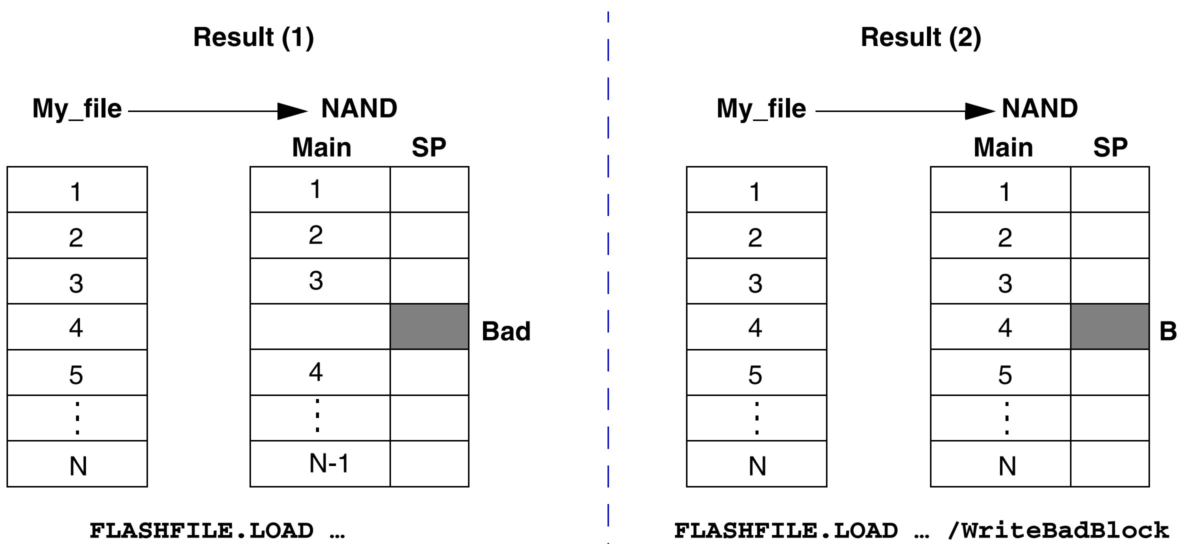
The data from *<file>* is written to the address range specified by *<range>*. If no *<range>* or *<address>* is specified, programming starts at address 0x0. Currently only binary files can be programmed.

Example 1

```
; Program contents of my_file.bin to NAND Flash main area starting at
; address 0x0.
; If a block is bad, the data is programmed to the next valid block.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF
```

Example 2

```
; Program contents of my_file.bin to NAND Flash main area starting
; at address 0x0.
; Even if a block is bad, data will be programmed.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF /WriteBadBlock
```



©1989-2014 Lauterbach GmbH

Verifying the Main Area

The following command is used to compare the NAND Flash main area with the specified target file:

FLASHFILE.LOAD *<file>* [*<address>* | *<range>*] **/ComPare**

The data from *<file>* is compared to the address range specified by *<range>*. If no *<range>* or *<address>* is specified, comparing starts at address 0x0.

Example 1

```
; Verify the contents of my_file.bin against the NAND Flash main area,
; starting at address 0x0.
; If a block is bad, then the data in the file is verified against
; the next valid block up to the end of the specified range.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF /ComPare
```

Example 2

```
; Verify the contents of my_file.bin against NAND Flash main area,
; starting at address 0x0.
; Even if a block is bad, the data will be verified against the bad block
; data.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF /WriteBadBlock /ComPare
```

©1989-2014 Lauterbach GmbH

Other Useful Commands (NAND)

Writing Other File Formats to the Main Area

The following commands are available to load IntelHex and S-Record files:

FLASHFILE.LOAD.IntelHex *<file>*	Program an intelhex file to the NAND Flash.

FLASHFILE.LOAD.S1record *<file>* **FLASHFILE.LOAD.S2record** *<file>* **FLASHFILE.LOAD.S3record** *<file>*	Program an S-record file to the NAND Flash.

Modifying the Main Area

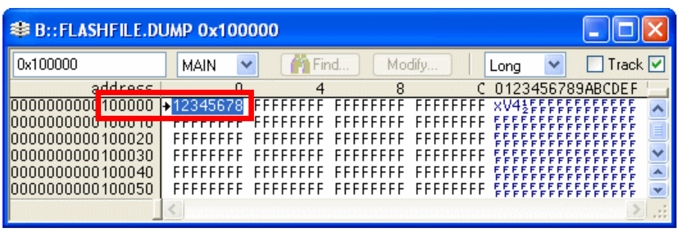
The following command is available to modify the contents of the NAND Flash memory. The maximum range that one **FLASHFILE.Set** command can modify is only one block of the Flash memory. If you want to modify three blocks, you need three FLASHFILE.Set commands, etc. See below for an example.

FLASHFILE.Set [*<address>* | *<range>*] *%<format> <data>*	Modify the contents of the NAND Flash.

Example 1

```
; Write 4 bytes of data 0x12345678 to the address 0x100000.
; LE = little endian
FLASHFILE.Set 0x100000 %LE %Long 0x12345678
```

Result (1)

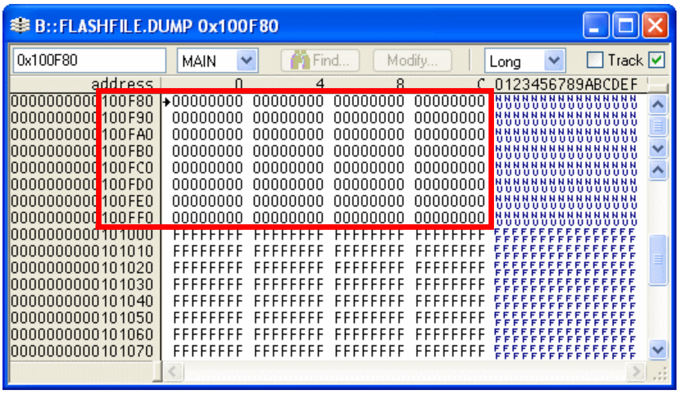


Example 2

```
; Write data 0x0 to the address range 0x100000++0xFFF.
FLASHFILE.Set 0x100000++0xFFF %Byte 0x0
```

©1989-2014 Lauterbach GmbH

Result (2)



Example 3

```
; A NAND Flash has 128KB per block (0x20000).
; Write data 0x0 from 0x100000 to 0x15FFFF in the NAND Flash.
FLASHFILE.Set 0x100000++0x1ffff %Byte 0x0
FLASHFILE.Set 0x120000++0x1ffff %Byte 0x0
FLASHFILE.Set 0x140000++0x1ffff %Byte 0x0
```

Copying the Main Area

The following command is available to copy:

- Any data from any CPU memory area to the NAND Flash, or
- Any data from one address range of the NAND Flash to another address range within the same NAND Flash; for example, for backup purposes.

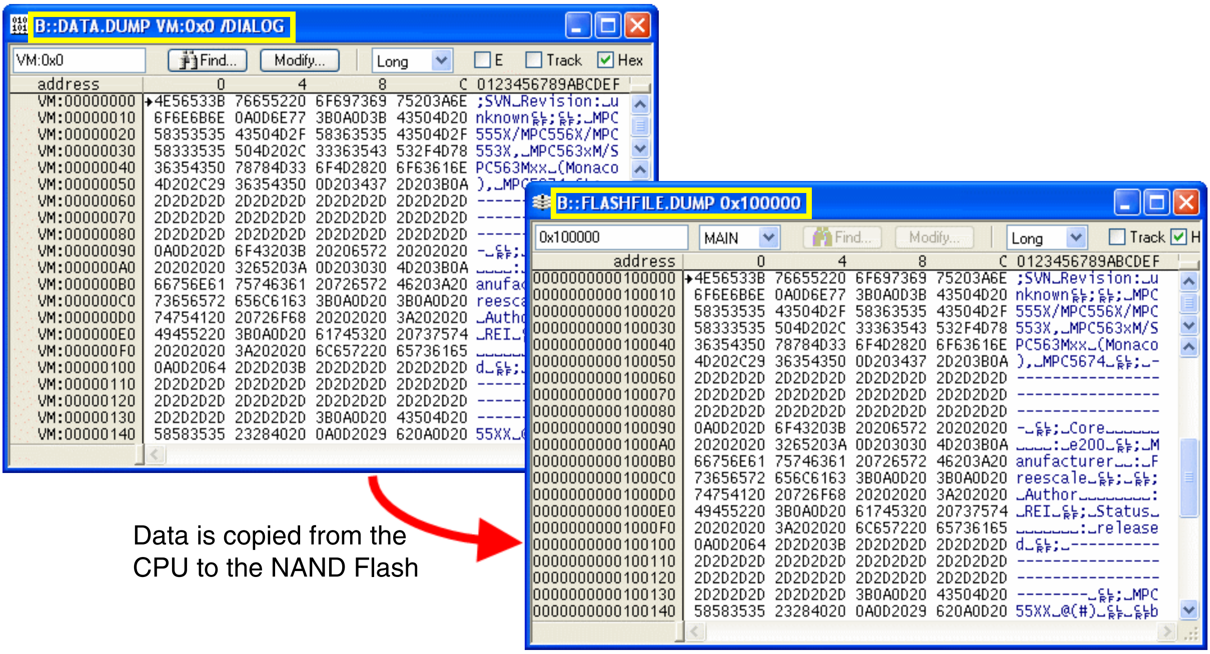
FLASHFILE.COPY *<source range> <target addr>*	Copy data from the source range to the defined address of the NAND Flash.
FLASHFILE.COPY *<source range> <target addr>* **/ComPare**	Verify the source range data against the target range data.

Example 1

```
; Copy the 2MB virtual memory data at 0x0 to the NAND Flash address
; at 0x100000.
; Bad blocks are skipped, data is written to the next valid block.
; VM: The virtual memory of the TRACE32 software.
FLASHFILE.COPY VM:0x0--0x1FFFFF 0x100000
```

©1989-2014 Lauterbach GmbH

Result (1)



Example 2

```
; Verify the data between virtual memory and NAND Flash.
FLASHFILE.COPY VM:0x0--0x1FFFFF 0x100000 /ComPare
```

Example 3

```
; Copy the 4MB NAND Flash data at 0x0 to the NAND Flash
; at 0x800000.
; Bad blocks are skipped, data is written to the next valid block.
FLASHFILE.COPY 0x0--0x3FFFFF 0x800000

; Verify the 4MB NAND Flash data between 0x0 and 0x800000.
FLASHFILE.COPY 0x0--0x3FFFFF 0x800000 /ComPare
```

The following commands are available to write a bad block marker, ECC codes, and special customer data to the NAND Flash spare area:

> Program the NAND Flash spare area except bad blocks.
>
> **FLASHFILE.LOADSPARE** *<file>* [*<address>* | *<range>*]

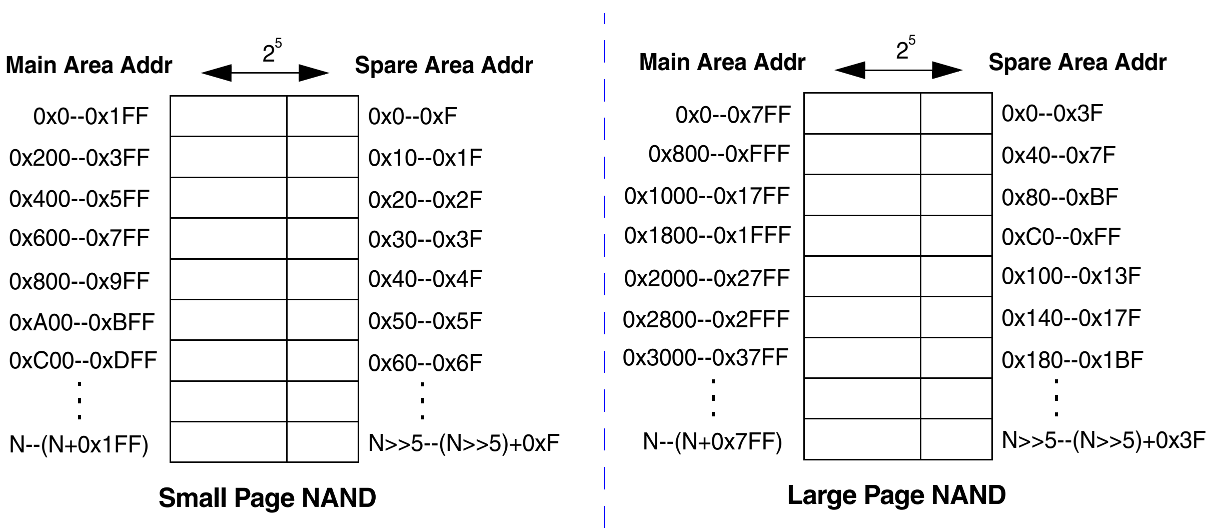
> Program the NAND Flash spare area including bad blocks.
>
> **FLASHFILE.LOADSPARE** *<file>* [*<address>* | *<range>*] **/WriteBadBlocks**

> Compare the NAND Flash spare area except bad blocks.
>
> **FLASHFILE.LOADSPARE** *<file>* [*<address>* | *<range>*] **/ComPare**

> Compare the NAND Flash spare area including bad blocks.
>
> **FLASHFILE.LOADSPARE** *<file>* [*<address>* | *<range>*] **/WriteBadBlocks /ComPare**

The data from *<file>* is written to the address range specified by *<range>*. If no *<range>* or *<address>* is specified, programming starts at address 0x0. Currently only binary files can be programmed.

NOTE:	• You need a third-party tool to create the spare file (*<file>*). • Be careful when you specify *<range>*: You should input *<range>* in the spare area address format, not in the main area format (see figure below).



©1989-2014 Lauterbach GmbH

Example 1

```
; Write my_spare.bin to the NAND Flash spare area.
; Start at the address 0x0 of the spare area.
; The bad blocks of my_spare.bin are excluded.
FLASHFILE.LOADSPARE my_spare.bin 0x0
```

Example 2: When specifying the address range, remember to use the address format of the spare area.

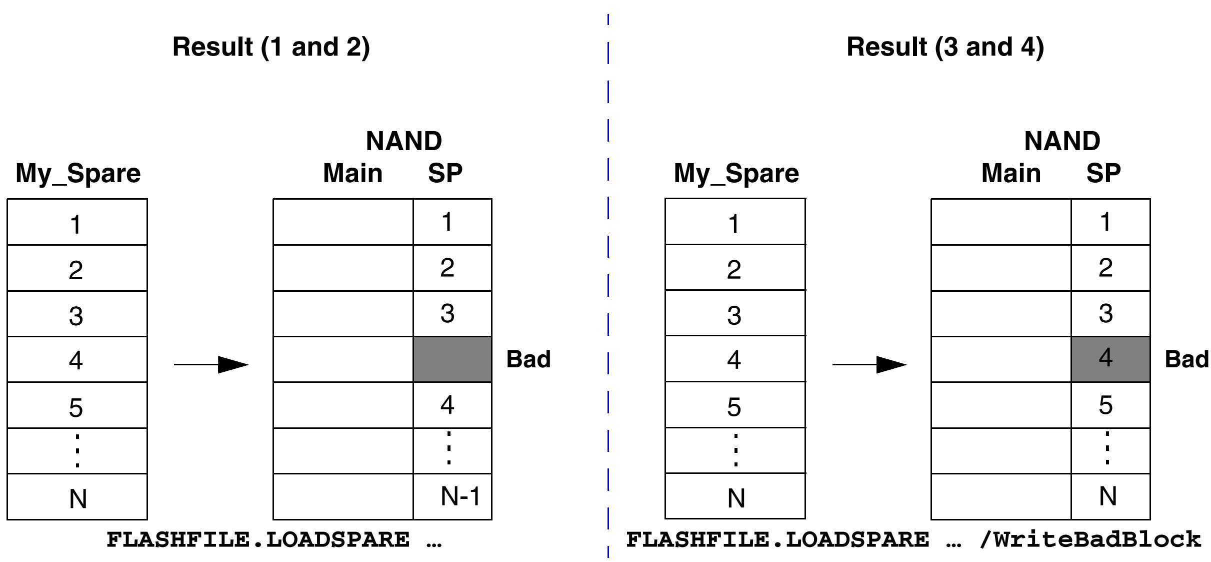
```
; Write 32KB of my_spare.bin to the specified address range
; of the spare area.
; The bad blocks of my_spare.bin are excluded.
FLASHFILE.LOADSPARE my_spare.bin 0x0--0x7FFF
```

Example 3

```
; Write my_spare.bin to the spare area.
; Start at the address 0x0 of the spare area.
; Include the bad blocks of my_spare.bin.
FLASHFILE.LOADSPARE my_spare.bin 0x0 /WriteBadBlock
```

Example 4

```
; Write 32KB of my_spare.bin to the spare area.
; Start at the address 0x0 of the spare area.
; Include the bad blocks of my_spare.bin.
FLASHFILE.LOADSPARE my_spare.bin 0x0--0x7FFF /WriteBadBlock
```



Example 5

```
; Verify the entire file my_spare.bin against the spare area.
; Start at the address 0x0 of the spare area.
FLASHFILE.LOADSPARE my_spare.bin 0x0 /ComPare
```

©1989-2014 Lauterbach GmbH

Programming the ECC Code to the Spare Area

tbd.

©1989-2014 Lauterbach GmbH

Reading/Saving the NAND Flash Device

The CPU cannot read NAND Flash devices directly. But TRACE32 provides special commands for reading NAND Flash memories. The contents of the NAND Flash are displayed in a window.

Reading the Main/Spare Area

The following commands are provided to read the NAND Flash areas.

FLASHFILE.DUMP [*<address>*] [**/***<format>*]	Display a hex-dump of the NAND Flash main area.

FLASHFILE.DUMP [*<address>* **/SPARE** [**/Track**]	Display a hex-dump of the NAND Flash spare area.

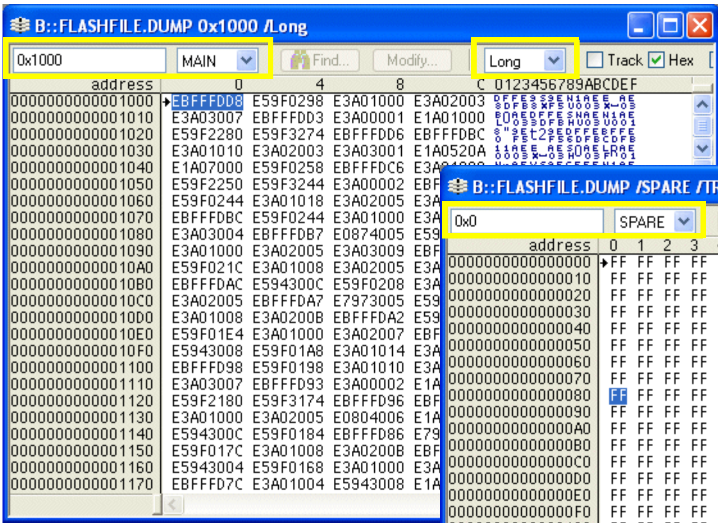
Example 1

```
; Display a hex-dump of the NAND Flash main area starting at 0x1000.
; Display the information in a 32-bit format (Long option).
FLASHFILE.DUMP 0x1000 /Long
```

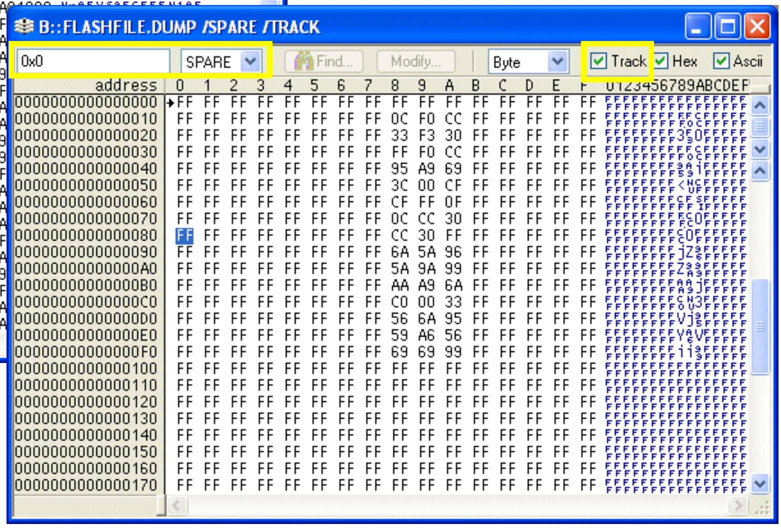
Example 2

```
; Display a hex-dump of the NAND Flash spare area.
; The cursor in the spare area display follows the cursor movements in
; the main area display (Track option).
FLASHFILE.DUMP /SPARE /Track.
```

Result (1)



Result (2)



Saving the Main Area

The following commands are available to save the contents of the NAND Flash main area to a file.

FLASHFILE.SAVE *<file> <range>*	Save the contents of the NAND Flash main area into *<file>*, bad blocks are saved.
FLASHFILE.SAVE *<file> <range>* **/SkipBadBlocks**	Save the contents of the NAND Flash main area into *<file>*, bad blocks are skipped.

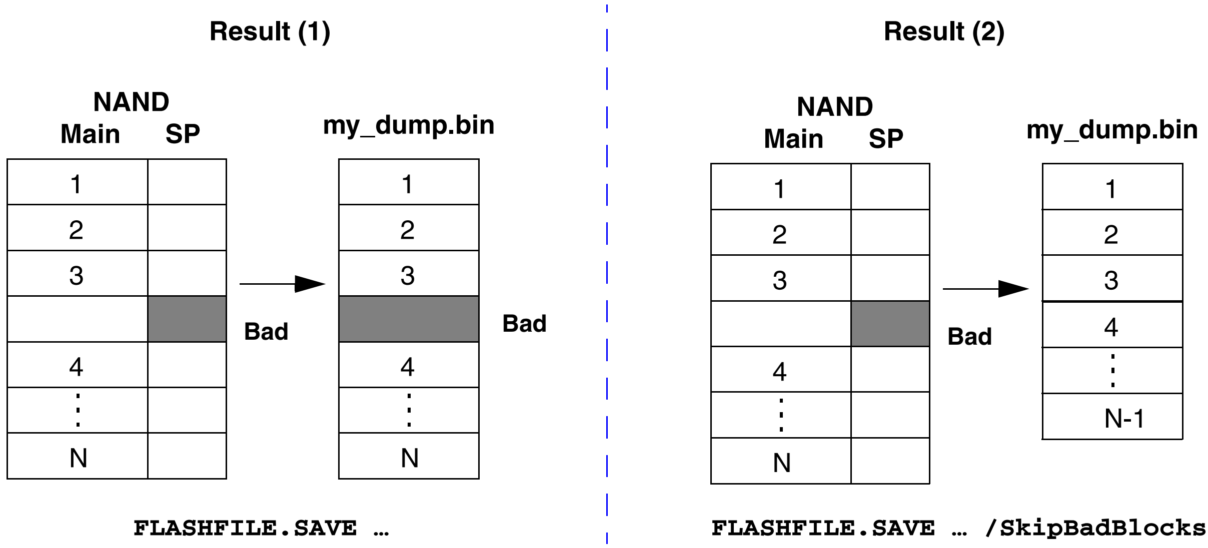
Example 1

```
; Save 1MB of the NAND Flash main area starting at 0x0 to the file
; my_dump.bin.
; The contents of bad block are also saved.
FLASHFILE.SAVE my_dump.bin 0x0--0xFFFFF
```

©1989-2014 Lauterbach GmbH

Example 2

```
; Save 1MB of the NAND Flash main area starting at 0x0 to the file
; my_dump.bin.
; The contents of bad block are skipped.
FLASHFILE.SAVE my_dump.bin 0x0--0xFFFFF /SkipBadBlocks
```



Saving the Spare Area

The following commands are available to save the contents of the NAND Flash spare area to a file.

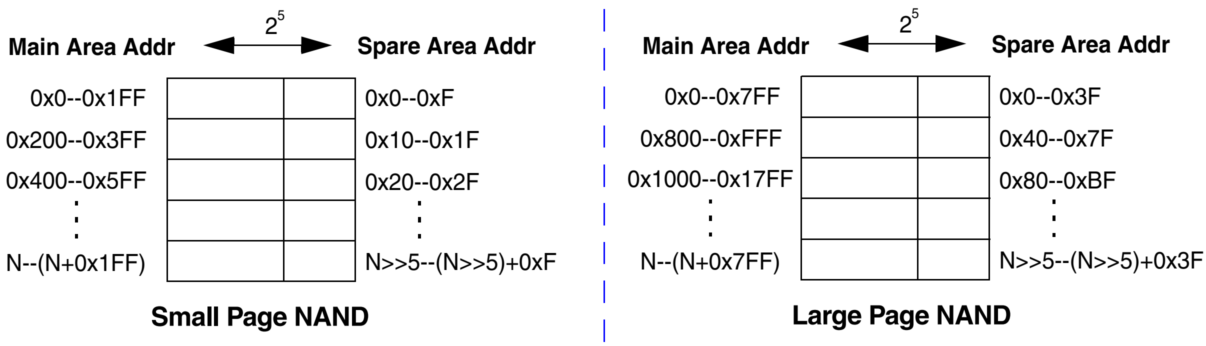
FLASHFILE.SAVESPARE *<file> <range>*	Save the contents of the NAND Flash spare area into *<file>*, bad blocks are saved.

FLASHFILE.SAVESPARE *<file> <range>* **/SkipBadBlocks**	Save the contents of the NAND Flash spare area into *<file>*, bad blocks are skipped.

Please be careful when you specify *<range>*:

©1989-2014 Lauterbach GmbH

You should input *<range>* in the spare area address format, not in the main area format (see figure below).

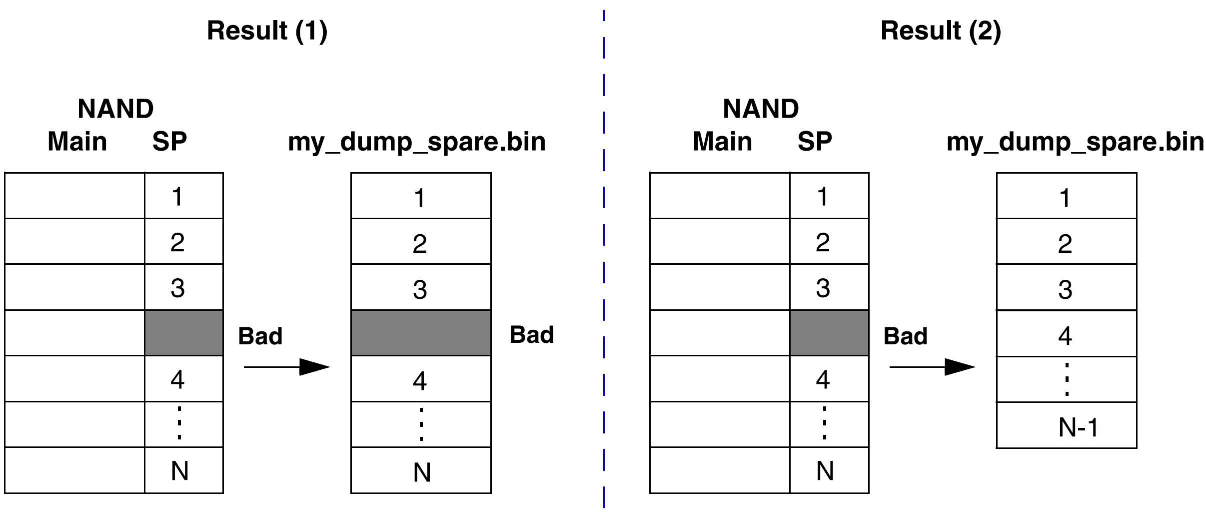


Example 1

```
; Save 32KB of the NAND Flash spare area starting at 0x0 to the file
; my_dump_spare.bin.
; The contents of bad block are also saved.
FLASHFILE.SAVESPARE my_dump_spare.bin 0x0--0x7FFF
```

Example 2

```
; Save 32KB of the NAND Flash spare area starting at 0x0 to the file
; my_dump_spare.bin.
; The contents of bad block are skipped.
FLASHFILE.SAVESPARE my_dump_spare.bin 0x0--0x7FFF /SkipBadBlocks
```



FLASHFILE.SAVESPARE ...

FLASHFILE.SAVESPARE ... **/SkipBadBlocks**

©1989-2014 Lauterbach GmbH

Example 1

CPU:	OMAP3430 (Texas Instruments) based on an ARM11 core.
NAND Flash:	MT29F1G08ABA (Micron)

NAND FLASH connected to the CS1 (Chip Selection 1) pin

Internal SRAM:	0x40200000
<cmd_reg>:	0x6E0000AC
<addr_reg>:	0x6E0000B0
<io_reg>:	0x6E0000B4

```
; Select OMAP3430 as target CPU.
SYStem.CPU OMAP3430

; Establish the communication between the debugger and the target CPU.
SYStem.Up

; Disable watchdog.
DO disable_watchdog.cmm

; Enable CS1 and define the base address of CS1(NAND Flash).
; LE = little endian
PER.Set SD:0x6E0000A8 %LE %Long 0x870

; Define the NAND Flash access timing.
PER.Set SD:0x6E000098 %LE %Long 0x60401
PER.Set SD:0x6E00009C %LE %Long 0x05010801

; Define CS1 for 8 bit NAND Flash.
PER.Set SD:0x6E000090 %LE %Long 0x0800 ; GPMC_CONFIG1_1

; Disable write protection for the NAND Flash device.
PER.Set SD:0x6E000050 %LE %Long 0x10 ; GPMC_CONFIG

; Reset the Flash declaration within TRACE32.
FLASHFILE.RESet

; Inform TRACE32 about the NAND Flash register addresses.
FLASHFILE.Config 0x6E0000AC 0x6E0000B0 0x6E0000B4

; Specify the NAND Flash programming algorithm and where it runs in the
; target RAM.
FLASHFILE.TARGET 0x40200000++0x3fff 0x40204000++0x3fff
                                     ~~/demo/arm/flash/byte/nand1g08.bin

; Check NAND Flash ID value.
FLASHFILE.GETID
```

©1989-2014 Lauterbach GmbH

```
; Erase NAND Flash including bad block.
FLASHFILE.Erase 0x0--0xFFFFF /EraseBadBlocks

; Program my_file.bin to NAND Flash main area.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF

ENDDO
```

©1989-2014 Lauterbach GmbH

CPU:	The STM32F103 is based on a Cortex-M3 core, which only runs Thumb-2 code. For this reason, a NAND Flash programming driver in thumb code is required.
NAND Flash:	Numonyx NAND512W3A2C (512 bytes per page), lock supported

NAND Flash connect to FSMC_NCE2, NAND Flash I/O

<cmd_reg>:	0x70020000
<addr_reg>:	0x70010000
<io_reg>:	0x70000000
Target RAM:	20 KB SRAM at 0x20000000

©1989-2014 Lauterbach GmbH

```
; Select STM32F103 as target CPU.
SYStem.CPU STM32F103ZE

; Establish the communication between the debugger and the target CPU.
SYStem.Up

; Clock enable to use FSMC and GPIO group related with NAND Flash.
PER.Set SD:0x40021014 %Long 0x114      ; FSCM clock enable
PER.Set SD:0x40021018 %Long 000001E0  ; GPIOD, GPIOE, GPIOF, GPIOG enable
; GPIO configuration CLE, ALE, D0->D3, NOE, NWE and NCE2
; (Output 50Mhz AF_PP), NWAIT((input pull-up) NAND pin configuration
PER.Set SD:0x40011400 %Long 0xB8BB44BB  ; GPIOD_CRL
PER.Set SD:0x40011404 %Long 0xBB4BB444  ; GPIOD_CRH
PER.Set SD:0x4001140C %Long 0x00000040  ; GPIOD_ODR pin6
; D4->D7 NAND pin configuration (output 50Mhz AF_PP)
PER.Set SD:0x40011800 %Long 0xB4444444  ; GPIOE
PER.Set SD:0x40011804 %Long 0x44444BBB  ; GPIOE
; INT2 NAND pin configuration (input pull-up)
PER.Set SD:0x40012000 %Long 0x48444444  ; GPIOG pin6
PER.Set SD:0x4001200C %Long 0x00000040  ; GPIOG_ODR pin6
; memory timing register
PER.Set SD:0xA0000068 %Long 0x01020301  ; FSMC_PMEM2
PER.Set SD:0xA000006C %Long 0x01020301  ; FSMC_PATT2
; Define & enable NAND Flash, 512 byte per page, ECC enable,
; 8 bit data width.
PER.Set SD:0xA0000060 %Long 0x0002004E  ;FSMC_PCR2

; Declarations for NAND Flash programming
FLASHFILE.RESet
FLASHFILE.CONFIG 0x70020000 0x70010000 0x70000000
FLASHFILE.TARGET 0x20000000++0x1fff 0x20002000++0x1fff
                 ~~/demo/arm/flash/byte/nand1208_cortexm3.bin

; Unlock, erase and program.
FLASHFILE.GETID
FLASHFILE.UNLOCK 0x000000++0xFFFFFF
FLASHFILE.Erase 0x00000++0xFFFFFF /EraseBadBlocks
FLASHFILE.LOAD  my_main_file.bin
ENDDO
```

©1989-2014 Lauterbach GmbH

Full Example: CPU-Specific NAND Flash Programming

CPU:	i.MX31 (Freescale)
NAND Flash:	MT29F1G08 (Micron)

NAND Flash connected to the NFCE (Flash Chip Enable) pin

<base_address>:	0xB8000000
Target RAM:	16KB SRAM at 0x1FFFC000

```
; Select i.MX31 as target CPU and establish communication between
; debugger and i.MX31.
SYStem.RESet
SYStem.CPU MCIMX31
SYStem.Option ResBreak OFF
SYStem.JtagClock RTCK
SYStem.Up

; Declare the NAND Flash Controller.
&nand_ctrl_base_addr=0xB8000000
FLASHFILE.RESet
FLASHFILE.CONFIG &nand_ctrl_base_addr , ,
FLASHFILE.TARGET 0x1FFFC000++0x1FFF 0x1FFFE000++0x1FFF
                ~~/demo/arm/flash/byte/nand1g08_imx.bin

; Erase and program.
FLASHFILE.GETID
FLASHFILE.Erase 0x0++0xFFFFF /EraseBadBlocks
FLASHFILE.LOAD C:\T32\my_file.bin 0x0++0xFFFFF

ENDDO
```

©1989-2014 Lauterbach GmbH

About OneNAND Flash Devices

A OneNAND Flash is a special NAND Flash type:

- A OneNAND Flash has a NOR Flash programming interface between the CPU and the OneNAND.
- The NAND Flash controller logic is part of the OneNAND Flash, so the target CPU does not need an integrated NAND Flash controller.

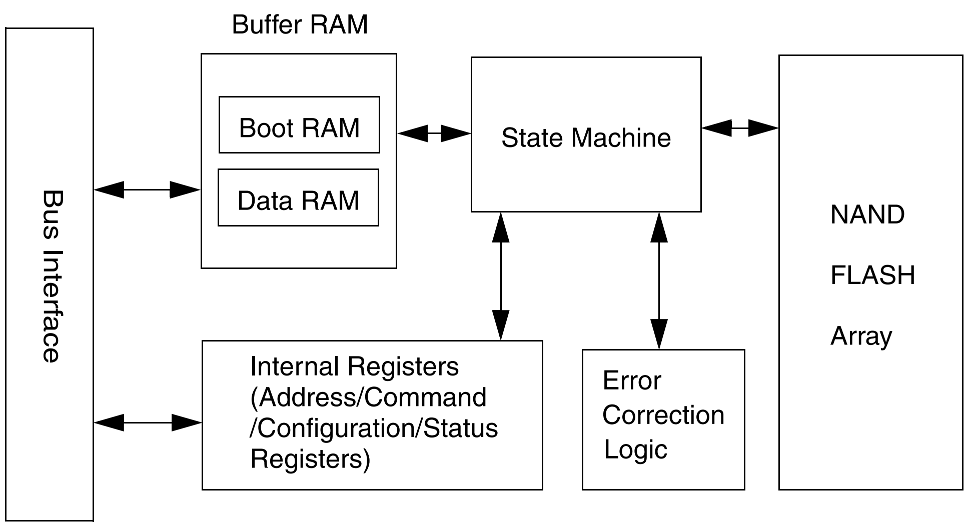


Figure: OneNAND Flash Block Diagram

©1989-2014 Lauterbach GmbH

Scripts for OneNAND Flash Devices

This chapter describes how to create scripts for OneNAND Flash programming.

The steps and the framework (see below) provide an overview of the process. They are described in detail in the following sections.

The following steps are necessary to create a new script:

1. Establish communication between debugger and target CPU.
2. Configure the OneNAND Flash bus.
3. Reset the NAND Flash environment in TRACE32 to its default values.
4. Inform TRACE32 about the OneNAND Flash address (Flash declaration).
5. Inform TRACE32 about the OneNAND Flash programming algorithm.
6. Check the identification from the OneNAND Flash device.
7. Erase the OneNAND Flash device.
8. Program the OneNAND Flash device.

©1989-2014 Lauterbach GmbH

The following framework can be used as base for OneNAND Flash programming:

```
                                        ; Establish the communication
                                        ; between the CPU and the TRACE32
                                        ; debugger.

                                        ; Configure the OneNAND Flash
                                        ; controller.

FLASHFILE.RESet                         ; Reset the OneNAND Flash
                                        ; declaration within TRACE32.

FLASHFILE.CONFIG …                      ; Inform TRACE32 about the
                                        ; OneNAND Flash register addresses.

FLASHFILE.TARGET …                      ; Specify the OneNAND Flash
                                        ; programming algorithm and where
                                        ; it runs in target RAM.

FLASHFILE.GETID                         ; Get the ID values of the OneNAND
                                        ; Flash.

FLASHFILE.Erase …                       ; Erase the OneNAND Flash.

FLASHFILE.LOAD <main_file> …            ; Program the file to the OneNAND
                                        ; Flash (main area).
```

An ellipsis (…) in the framework indicates that command parameters have been omitted here for space economy.

NOTE:	The parametrization of **FLASHFILE.CONFIG** and **FLASHFILE.TARGET** requires expert knowledge.

A template script (*.cmm) for OneNAND Flash programming is provided by Lauterbach. It can be found in the TRACE32 installation directory.

`<TRACE32_installation_directory>/demo/<architecture>/flash/onenand.cmm`

Whereas ~~ is expanded to the `<TRACE32_installation_directory>`, which is `c:/T32` by default.

©1989-2014 Lauterbach GmbH

Establishing Communication between Debugger and Target CPU

OneNAND Flash programming with TRACE32 requires that the communication between the debugger and the target CPU is established. The following commands are available to set up this communication:

SYStem.CPU *<cpu>*	Specify your target CPU.
SYStem.Up	Establish the communication between the debugger and the target CPU.

```
SYStem.CPU OMAP3430      ; Select OMAP3430 as target CPU.

SYStem.Up                ; Establish the communication between the
                         ; debugger and the target CPU.
```

Configuring the OneNAND Flash Bus

Programming an off-chip OneNAND Flash devices requires a proper initialization of the external bus interface. The following settings in the bus configuration might be necessary:

- Definition of the base address of the OneNAND Flash devices
- Definition of the size of the OneNAND Flash devices
- Definition of the data bus width that is used to access the OneNAND Flash devices
- Definition of the timing (number of wait states for the access to the OneNAND Flash devices)
- Definition of the bus type of the OneNAND Flash devices (for example, muxed mode)

Example: Define the bus configuration registers for the OneNAND Flash device.

```
PER.Set SD:0x6E0000D8 %Long           ; Enable chip selection and define
0x8000080                             ; 128MB OneNAND Flash size and the
                                      ; base address is 0x8000000.

PER.Set SD:0x6E0000C0 %Long 0x1200    ; Define chip selection for 16 bit
                                      ; muxed (address & data) for
                                      ; OneNAND Flash.
```

©1989-2014 Lauterbach GmbH

Resetting Default Values

The following command is used to reset the OneNAND Flash environment in TRACE32 to its default values.

FLASHFILE.RESet	Reset the OneNAND Flash environment in TRACE32 to its default values.

Informing TRACE32 about the OneNAND Flash Address

The following command is available to inform TRACE32 about the start address of the OneNAND Flash base register.

FLASHFILE.CONFIG *<base_address> , ,*	Inform TRACE32 about the start address of the OneNAND Flash base register. , represents don't-care parameters.

For information about the OneNAND Flash base register, refer to the manufacturer's processor manual.

Example: base address of the OneNAND Flash controller in the OMAP3430 as target CPU:

```
; Inform TRACE32 about the start address of the OneNAND Flash
; base register.
FLASHFILE.Config 0x08000000 , ,
```

©1989-2014 Lauterbach GmbH

The following command is available to inform TRACE32 about the OneNAND Flash device to be programmed:

FLASHFILE.TARGET *<code_range> <data_range> <file>*	Specify the OneNAND Flash programming algorithm and where it runs in the target RAM.

Parameters

- *<code_range>*

  Define an address range in the target´s RAM to which the OneNAND Flash programming algorithm is loaded.

Flash algorithm
32 byte

Figure: Memory mapping for the *<code_range>*

Required size for the code is `size_of(`*<file>*`) + 32 byte`

- *<data_range>*

  Define the address range in the target´s RAM where the programming data is buffered for the programming algorithm.

64 byte argument buffer
buffer for programming data
256 byte stack

Figure: Memory mapping for the *<data_range>*

The argument buffer used for the communication between the TRACE32 software and the programming algorithm is located at the first 64 bytes of *<data_range>*. The 256 byte stack is located at the end of *<data_range>*.

```
<buffer_size> =
size_of(<data_range>) - 64 byte argument buffer - 256 byte stack
```

`<buffer_size>` is the maximum number of bytes that are transferred from the TRACE32 software to the OneNAND programming algorithm in one call.

- *<file>*

  Lauterbach provides ready-to-run driver binary files for OneNAND Flash programming. They are located in the TRACE32 installation directory:

  They are located in the TRACE32 installation directory:

  `~~/demo/<architecture>/flash/<bus_width>/`

  Whereas `~~` is expanded to the `<TRACE32_installation_directory>`, which is `c:/T32` by default.

  The manual **”List of Supported FLASH Devices”** (flashlist.pdf) provides a list of the supported OneNAND Flash devices and the appropriate programming driver name.

  The Lauterbach home page provides the same information and is updated more often: **http://www.lauterbach.com/ylist.html.**

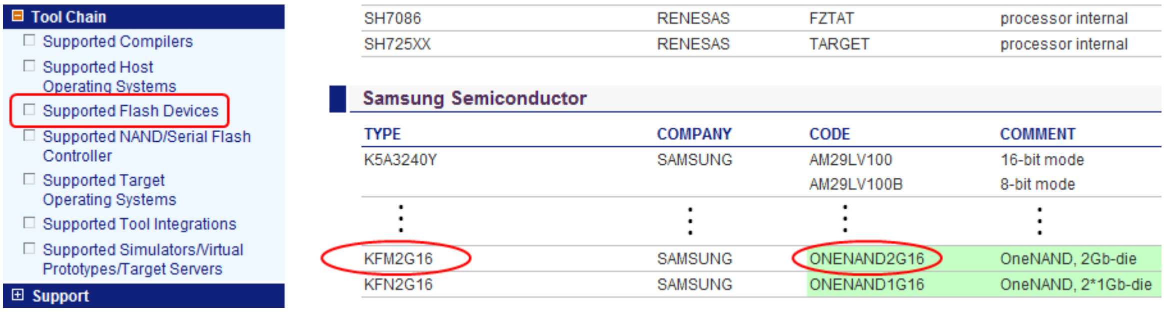
  For detailed information about how to determine the *<file>* parameter, see “**Identifying the Correct OneNAND Flash Driver for a OneNAND Device**” on **page 57**.

©1989-2014 Lauterbach GmbH

Identifying the Correct OneNAND Flash Driver for a OneNAND Device

1. For information about supported Flash devices, access the Lauterbach website: **http://www.lauterbach.com/ylist.html.**
2. Click the **+** drill-down button next to **Tool Chain**, and then click **Supported Flash Devices**.
3. Scroll through the list to locate the desired OneNAND Flash device.

   Based on the name of the Flash device, you can identify the correct OneNAND Flash driver binary file.



The **Code** column identifies the OneNAND Flash driver binary file.

The file **`onenand2g16.bin`** resides in this folder **`~~/T32/demo/arm/flash/word`**

Whereas `~~` is expanded to the <TRACE32_installation_directory>, which is `c:/T32` by default.

The number **`16`** indicates the bus width and the folder where the file resides, i.e. in the **`word`** folder.

Naming Convention for OneNAND Flash Drivers

The name of the OneNAND programming driver depends on:

1. The bus width between the CPU and the OneNAND Flash device.
2. The die, which describes the internal organization of the OneNAND Flash device

A 2 GByte OneNAND Flash, for example, can consist of a single 2 GByte die or of two 1 GByte dies.

Please refer to the datasheet of your OneNAND Flash device to get this information.

Naming examples are given in the table below:

OneNAND Flash	Bus Width	Die	Driver
KFG1G16	16	1 GByte	onenand1g16.bin
KFH2G16	16	1 GByte	onenand1g16.bin
KFM1G16	16	1 GByte	onenand1g16.bin

©1989-2014 Lauterbach GmbH

OneNAND Flash	Bus Width	Die	Driver
KFN2G16	16	1 GByte	onenand1g16.bin
KFG2G16	16	2 GByte	onenand2g16.bin
KFH4G16	16	2 GByte	onenand2g16.bin
KFM2G16	16	2 GByte	onenand2g16.bin
KFN4G16	16	2 GByte	onenand2g16.bin

©1989-2014 Lauterbach GmbH

Checking the Identification from the OneNAND Flash Device

The following command can be used to check if TRACE32 can access the OneNAND Flash device:

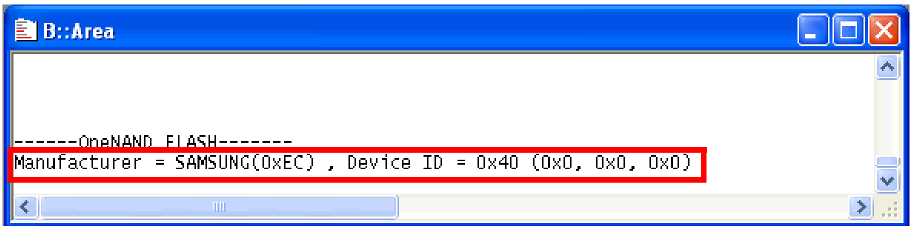
FLASHFILE.GETID	Get the ID values for OneNAND Flash.

Example

```
; Open the TRACE32 AREA window.
AREA.view

; Check the access to the OneNAND Flash device
; by getting the manufacturer ID and the device ID.
FLASHFILE.GETID
```

Manufacturer ID: Samsung
Device ID: KFM2G162M



Erasing the OneNAND Flash Device

The following command is used to erase OneNAND Flash devices:

FLASHFILE.Erase <*range*>	Erase OneNAND Flash except bad blocks.

FLASHFILE.Erase <*range*> **/EraseBadBlocks**	Erase OneNAND Flash including bad blocks.

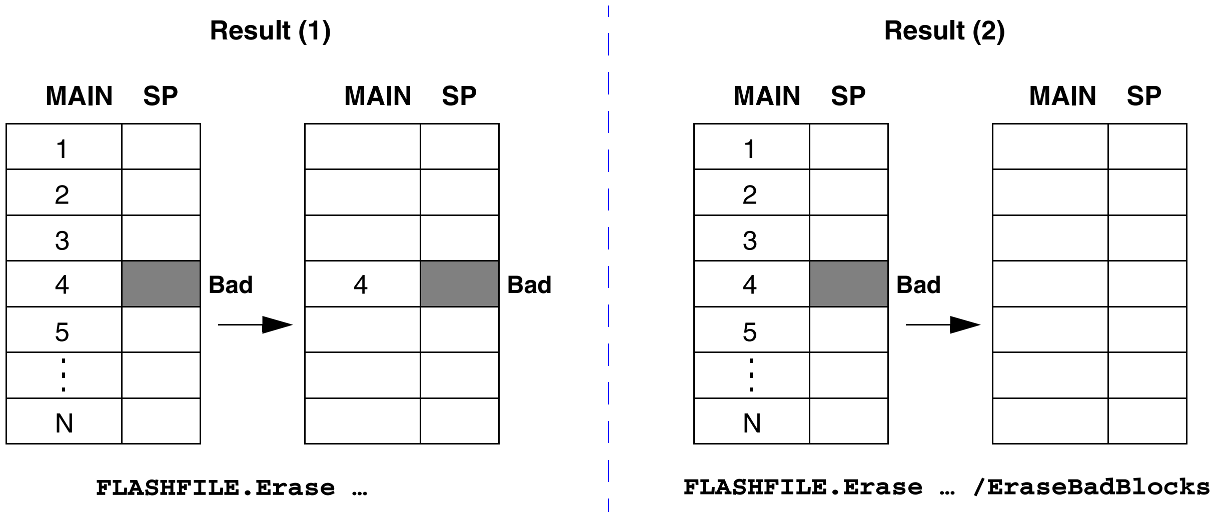
Example 1

```
; Erase 1MB starting from 0x0 except bad blocks.
FLASHFILE.Erase 0x0--0xFFFFF
```

©1989-2014 Lauterbach GmbH

Example 2

```
; Erase 1MB starting from 0x0 including bad blocks.
; Afterwards all bad block information is erased.
FLASHFILE.Erase 0x0--0xFFFFF /EraseBadBlocks
```



Programming the OneNAND Flash Device

OneNAND Flash devices consist of two areas:

- The **main area** contains the data which is accessed by the CPU.
- The **spare area** contains the bad block information and the ECC data.
  For background information about ECC, see "**Appendix: ECC (Error Correction Code)** on **page 75**.

The **FLASHFILE** commands allow to program the main and spare area independently.

©1989-2014 Lauterbach GmbH

Programming the Main Area (OneNAND)

The following commands are available to program the OneNAND Flash main area:

FLASHFILE.LOAD *<file>* [*<address>* | *<range>*]	Program OneNAND Flash except bad blocks.

FLASHFILE.LOAD *<file>* [*<address>* | *<range>*] **/WriteBadBlocks**	Program OneNAND Flash including bad blocks.

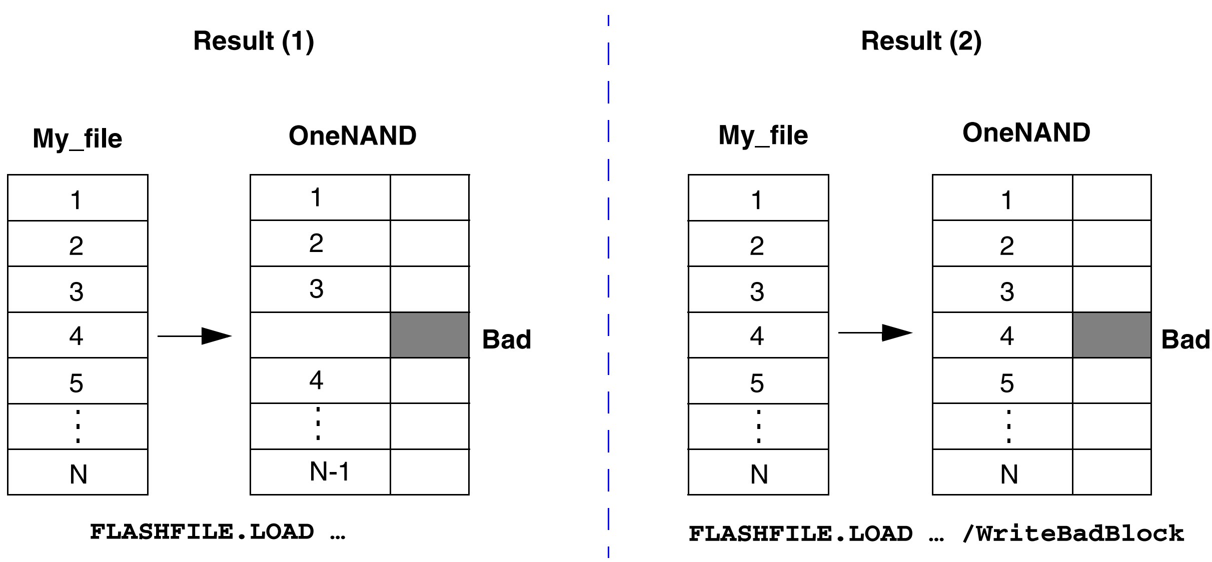
The data from *<file>* is written to the address range specified by *<range>*. If no *<range>* or *<address>* is specified, programming starts at address 0x0. Currently only binary files can be programmed.

Example 1

```
; Program contents of my_file.bin to the OneNAND Flash main area starting
; at address 0x0.
; If a block is bad, the data is programmed to the next valid block.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF
```

Example 2

```
; Program the contents of my_file.bin to OneNAND Flash main area starting
; at address 0x0.
; Even if a block is bad, data will be programmed.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF /WriteBadBlock
```



Verifying the Main Area (OneNAND)

The following command is used to compare the OneNAND Flash main area with the specified target file:

FLASHFILE.LOAD *<file>* [*<address>* | *<range>*] **/ComPare**

The data from *<file>* is compared to the address range specified by *<range>*. If no *<range>* or *<address>* is specified, comparing starts at address 0x0.

Example 1

```
; Verify the contents of my_file.bin against the NAND Flash main area,
; starting at address 0x0.
; If a block is bad, then the data in the file is verified against
; The next valid block up to the end of the range specified.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF /ComPare
```

Example 2

```
; Verify the contents of my_file.bin against NAND Flash main area,
; starting at address 0x0.
; Even if a block is bad, the data will be verified against the bad block
; data.
FLASHFILE.LOAD my_file.bin 0x0--0xFFFFF /WriteBadBlock /ComPare
```

©1989-2014 Lauterbach GmbH

Copying the Main Area (OneNAND)

The following command is available to copy:

- Any data from any CPU memory area to the OneNAND Flash, or
- Any data from one address range of the OneNAND Flash to another address range within the same OneNAND Flash; for example, for backup purposes.

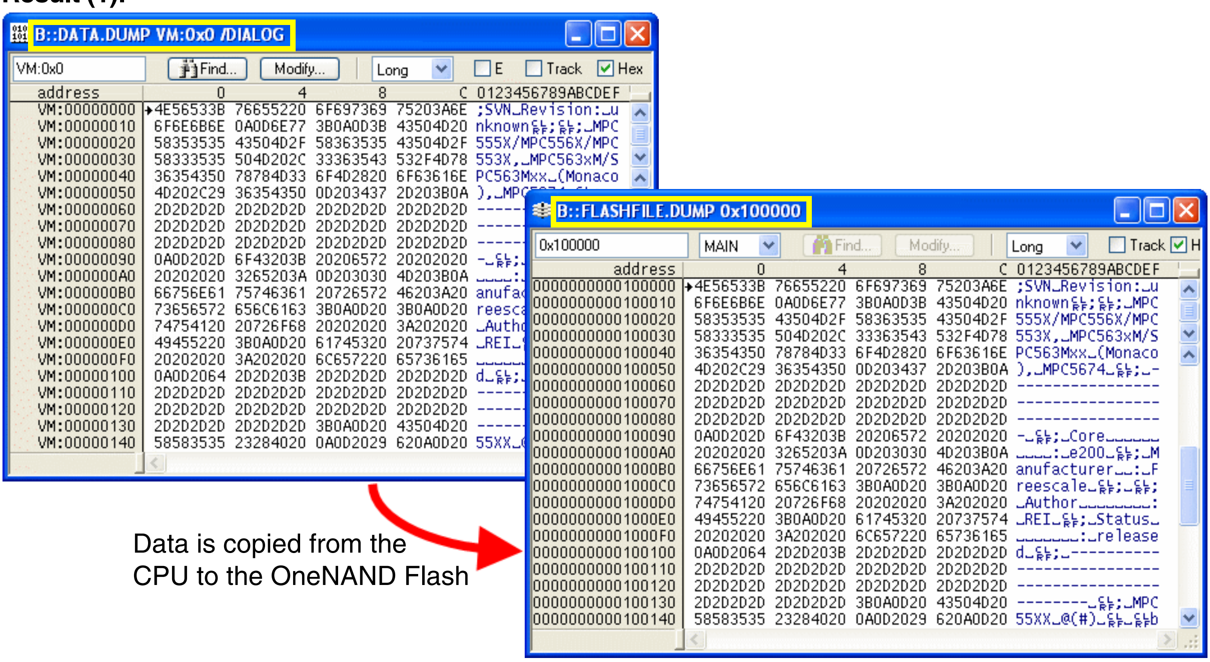
FLASHFILE.COPY *<source range> <target addr>*	Copy data from the source range to the defined address of the OneNAND Flash.

FLASHFILE.COPY *<source range> <target addr>* **/ComPare**	Verify the source range data against the target range data.

Example 1

```
; Copy the 2MB virtual memory data at 0x0 to the OneNAND Flash address
; at 0x100000.
; Bad blocks are skipped, data is written to the next valid block.
; VM: stands for virtual memory.
FLASHFILE.COPY VM:0x0--0x1FFFFF 0x100000
```

Result (1):



Example 2

```
; Verify the data between virtual memory and OneNAND Flash.
FLASHFILE.COPY VM:0x0--0x1FFFFF 0x100000 /ComPare
```

Example 3

```
; Copy the 4MB OneNAND Flash data at 0x0 to the OneNAND Flash
; at 0x800000.
; Bad blocks are skipped, data is written to the next valid block.
FLASHFILE.COPY 0x0--0x3FFFFF 0x800000

; Verify the 4MB OneNAND Flash data between 0x0 and 0x800000.
FLASHFILE.COPY 0x0--0x3FFFFF 0x800000 /ComPare
```

©1989-2014 Lauterbach GmbH

The following command is available to modify the contents of the OneNAND Flash. The maximum range that one **FLASHFILE.Set** command can modify is only one block of the Flash memory. If you want to modify three blocks, you need three FLASHFILE.Set commands, etc.

FLASHFILE.Set [*<address>* | *<range>*] *%<format>* *<data>*	Modify the contents of the OneNAND Flash.

Example 1

```
; Write 4 bytes of data (= 0x12345678) to the address 0x100000.
; LE = little endian
FLASHFILE.Set 0x100000 %LE %Long 0x12345678
```

Example 2

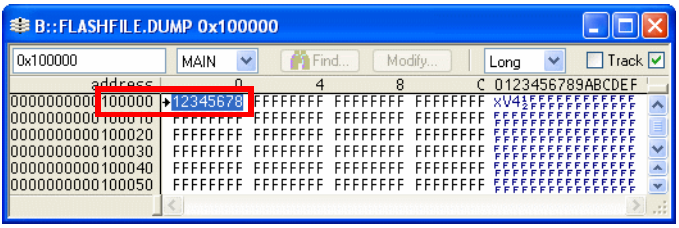
```
; Write data 0x0 to the address range 0x100000++0xFFF.
FLASHFILE.Set 0x100000++0xFFF %Byte 0x0
```

Example 3

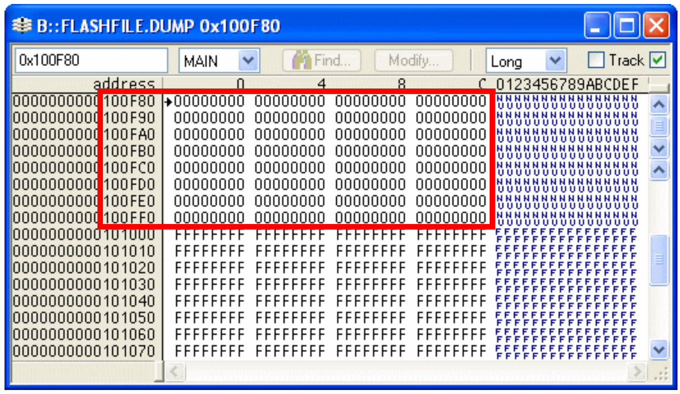
```
; A OneNAND Flash has 128KB per block (0x20000).
; Write data 0x0 from 0x100000 to 0x15FFFF in the OneNAND Flash.
FLASHFILE.Set 0x100000++0x1ffff %Byte 0x0
FLASHFILE.Set 0x120000++0x1ffff %Byte 0x0
FLASHFILE.Set 0x140000++0x1ffff %Byte 0x0
```

©1989-2014 Lauterbach GmbH

Result (1)



Result (2)



Programming the Spare Area (OneNAND)

The following commands are available to program the OneNAND Flash spare area:

Program the OneNAND Flash spare area except bad blocks.

FLASHFILE.LOADSPARE *<file>* [*<address>* | *<range>*]

Program the OneNAND Flash spare area including bad blocks.

FLASHFILE.LOADSPARE *<file>* [*<address>* | *<range>*] **/WriteBadBlocks**

Compare the OneNAND Flash spare area except bad blocks.

FLASHFILE.LOADSPARE *<file>* [*<address>* | *<range>*] **/ComPare**

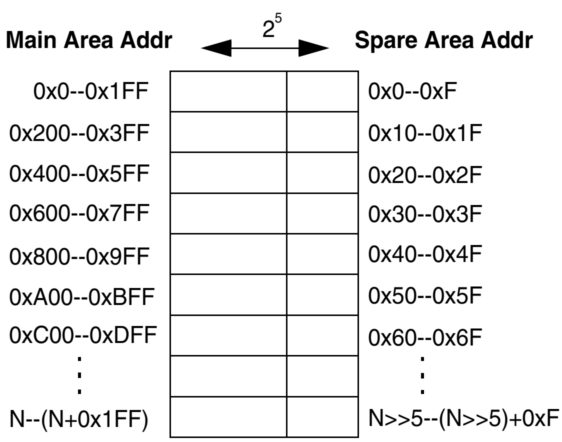
Compare the OneNAND Flash spare area including bad blocks.

FLASHFILE.LOADSPARE *<file>* [*<address>* | *<range>*] **/WriteBadBlocks /ComPare**

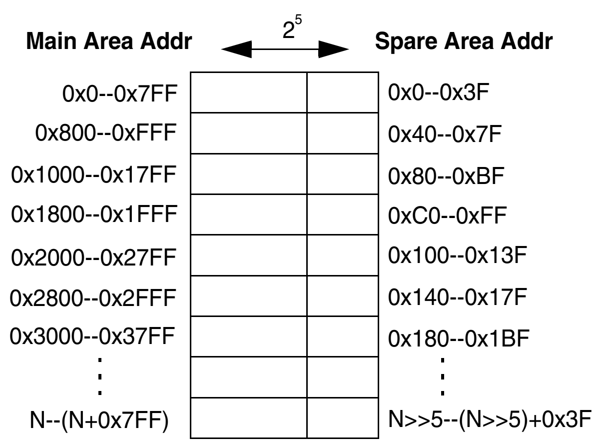
The data from *<file>* is written to the address range specified by *<range>*. If no *<range>* or *<address>* is specified, programming starts at address 0x0. Currently only binary files can be programmed.

NOTE:

- You need a third-party tool to create the spare file (*<file>*).
- Be careful when you specify *<range>*: You should input *<range>* in the spare area address format, not in the main area format (see figure below).



Small Page OneNAND



Large Page OneNAND

Example 1

```
; Write my_spare.bin to the OneNAND Flash spare area.
; Start at the address 0x0 of the spare area.
; The bad blocks of my_spare.bin are excluded.
FLASHFILE.LOADSPARE my_spare.bin 0x0
```

©1989-2014 Lauterbach GmbH

Example 2

When specifying the address range, remember to use the address format of the spare area.

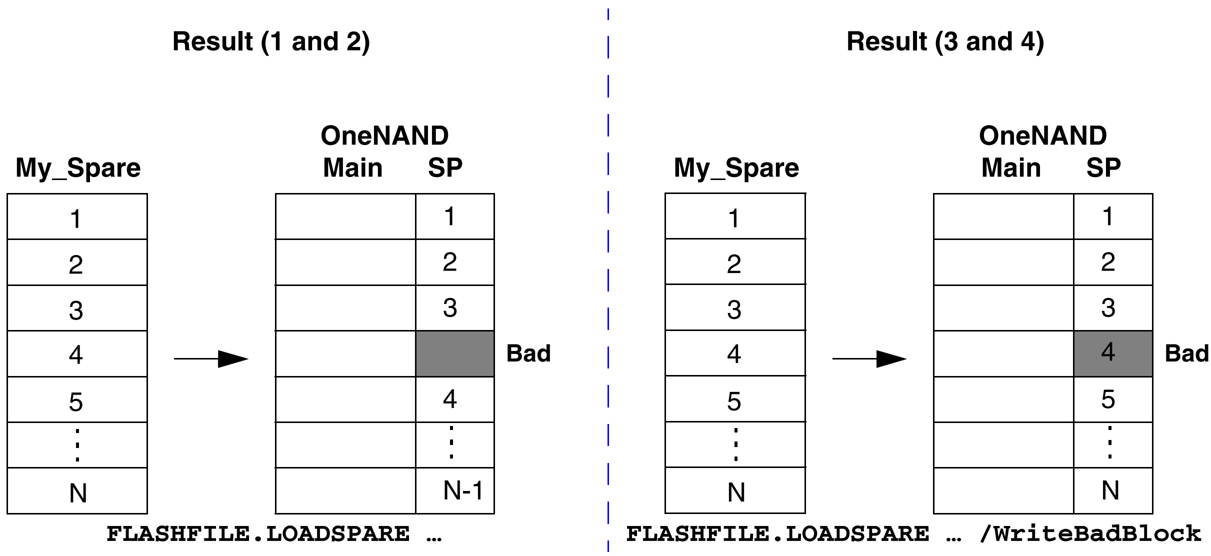
```
; Write 32KB of my_spare.bin to the specified address range
; of the spare area.
; The bad blocks of my_spare.bin are excluded.
FLASHFILE.LOADSPARE my_spare.bin 0x0--0x7FFF
```

Example 3

```
; Write my_spare.bin to the spare area.
; Start at the address 0x0 of the spare area.
; Include the bad blocks of my_spare.bin.
FLASHFILE.LOADSPARE my_spare.bin 0x0 /WriteBadBlock
```

Example 4

```
; Write 32KB of my_spare.bin to the spare area.
; Start at the address 0x0 of the spare area.
; Include the bad blocks of my_spare.bin.
FLASHFILE.LOADSPARE my_spare.bin 0x0--0x7FFF /WriteBadBlock
```



©1989-2014 Lauterbach GmbH

Example 5

```
; Compare the entire file my_spare.bin with the spare area.
; Start at the address 0x0 of the spare area.
FLASHFILE.LOADSPARE my_spare.bin 0x0 /ComPare
```

NOTE:	OneNAND Flash controllers generate the ECC data automatically when data is programmed to the main area, so the ECC codes in the spare area do not need to be programmed.

©1989-2014 Lauterbach GmbH

Reading/Saving the OneNAND Flash Device

The CPU cannot read OneNAND Flash devices directly. But TRACE32 provides special commands for reading OneNAND Flash devices. The contents of the OneNAND Flash are displayed in a window.

Reading the Main/Spare Area (OneNAND)

The following commands are available to read the OneNAND Flash areas.

FLASHFILE.DUMP [<*address*>] [**/**<*format*>]	Display a hex-dump of the OneNAND Flash main area.

FLASHFILE.DUMP [<*address*> **/SPARE** [**/Track**]	Display a hex-dump of the OneNAND Flash spare area.

Example 1

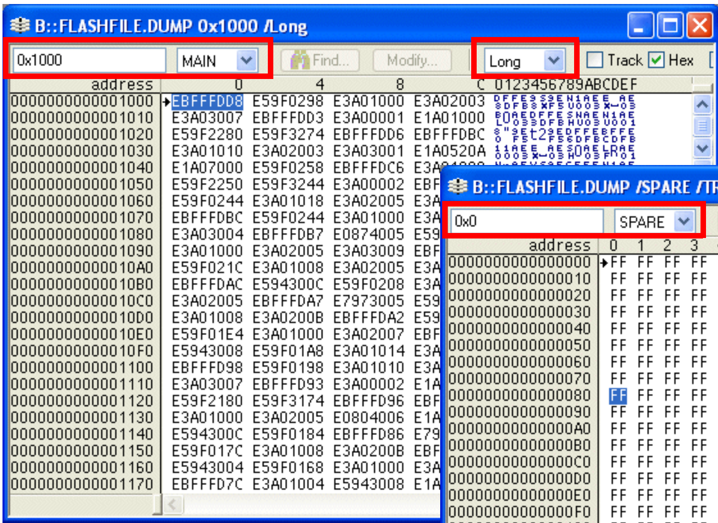
```
; Display a hex-dump of the OneNAND Flash main area starting at 0x1000.
; Display the information in a 32-bit format (Long option).
FLASHFILE.DUMP 0x1000 /Long
```

Example 2

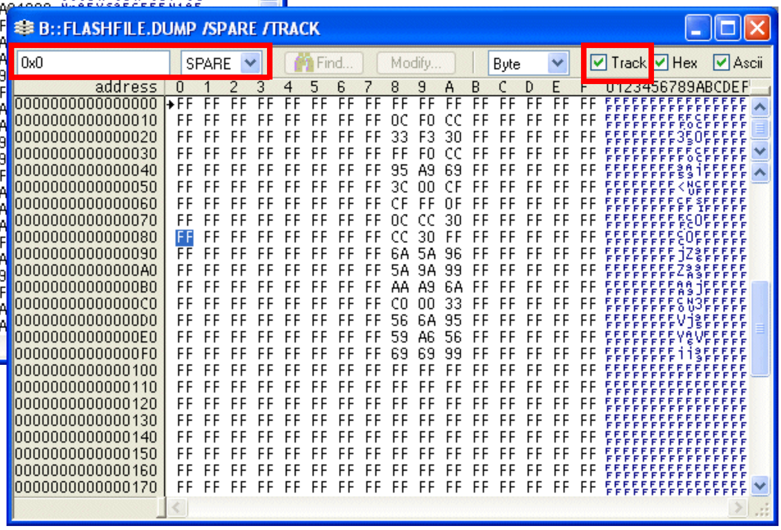
```
; Display a hex-dump of the OneNAND Flash spare area.
; The cursor in the spare area display follows the cursor movements in
; the main area display (Track option).
FLASHFILE.DUMP /SPARE /Track
```

©1989-2014 Lauterbach GmbH

Result (1)



Result (2)



Saving the Main Area (OneNAND)

The following commands are available to save the contents of the OneNAND Flash main area to a file.

FLASHFILE.SAVE *<file> <range>*	Save the contents of the OneNAND Flash main area into *<file>*, bad blocks are saved.
FLASHFILE.SAVE *<file> <range>* **/SkipBadBlocks**	Save the contents of the OneNAND Flash main area into *<file>*, bad blocks are skipped.

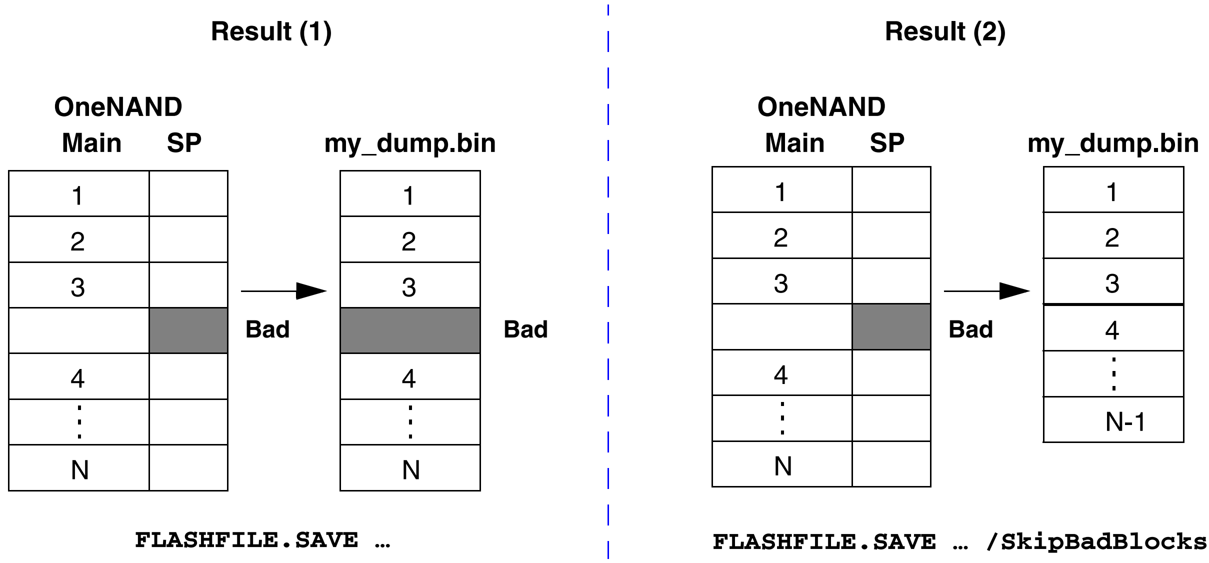
Example 1

```
; Save 1MB of the OneNAND Flash main area starting at 0x0 to the file
; my_dump.bin.
; The contents of bad blocks are also saved.
FLASHFILE.SAVE my_dump.bin 0x0--0xFFFFF
```

©1989-2014 Lauterbach GmbH

Example 2

```
; Save 1MB of the OneNAND Flash main area starting at 0x0 to the file
; my_dump.bin.
; The contents of bad blocks are skipped.
FLASHFILE.SAVE my_dump.bin 0x0--0xFFFFF /SkipBadBlocks
```



Saving the Spare Area (OneNAND)

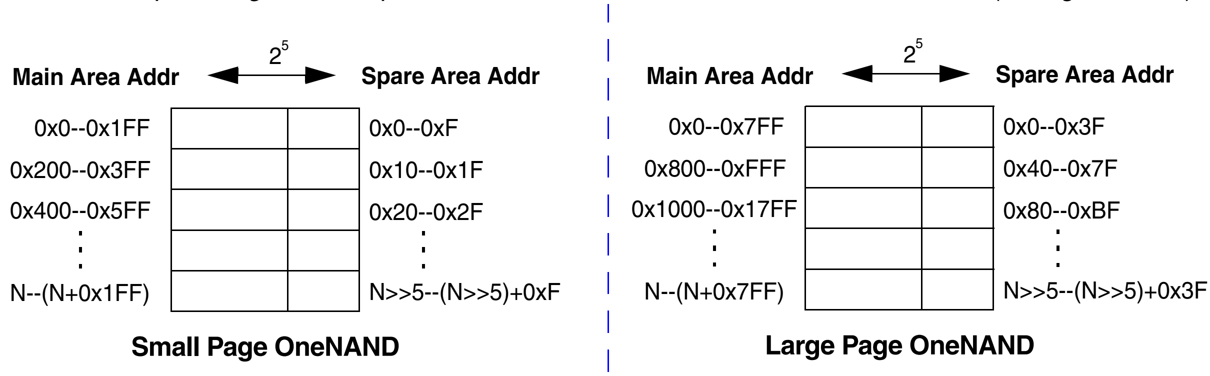
The following commands are available to save the contents of the OneNAND Flash spare area to a file.

FLASHFILE.SAVESPARE *<file>* *<range>*	Save the contents of the OneNAND Flash spare area into *<file>*, bad blocks are saved.
FLASHFILE.SAVESPARE *<file>* *<range>* **/SkipBadBlocks**	Save the contents of the OneNAND Flash spare area into *<file>*, bad blocks are skipped.

Please be careful when you specify *<range>*:

©1989-2014 Lauterbach GmbH

You should input <*range*> in the spare area address format, not in the main area format (see figure below).

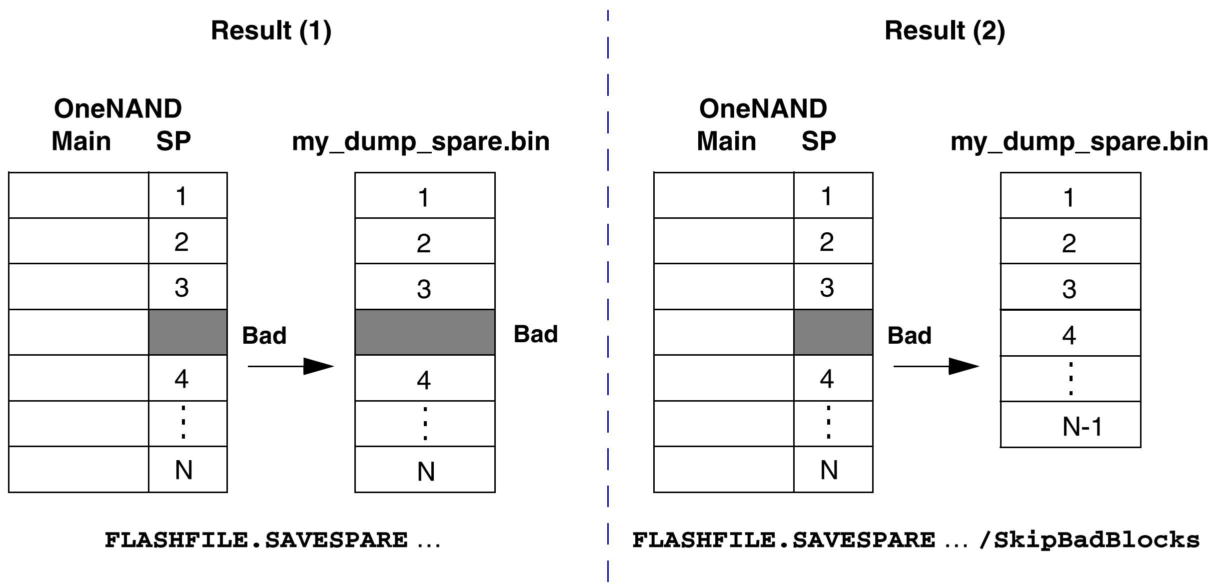


Example 1

```
; Save 32KB of the OneNAND Flash spare area starting at 0x0 to the file
; my_dump_spare.bin.
; The contents of bad blocks are also saved.
FLASHFILE.SAVESPARE my_dump_spare.bin 0x0--0x7FFF
```

Example 2

```
; Save 32KB of the OneNAND Flash spare area starting at 0x0 to the file
; my_dump_spare.bin.
; The contents of bad blocks are skipped.
FLASHFILE.SAVESPARE my_dump_spare.bin 0x0--0x7FFF /SkipBadBlocks
```



©1989-2014 Lauterbach GmbH

Full Example

CPU:	OMAP3430
OneNAND Flash:	KFM2G162M(SAMSUNG)
Bus width:	16-bit muxed
Die:	2 GByte

```
; Select OMAP3430 as target CPU.
SYStem.CPU OMAP3430

; Establish the communication between the debugger and the target CPU.
SYStem.Up

; Define CS2 for 16 bit muxed (address & data) OneNAND Flash.
PER.Set SD:0x6E0000C0 %l 0x1200 ; GPMC_CONFIG1_2

; Enable CS2 and define 128 MB size and the base address is 0x8000000.
PER.Set SD:0x6E0000D8 %l 0x848 ; GPMC_GPMC_CONFIG7_2

; Reset the Flash declaration.
FLASHFILE.RESet

; Specify the OneNAND Flash base address.
FLASHFILE.Config 0x08000000 , ,

; Specify the OneNAND Flash programming algorithm and where it runs
; in the target RAM.
FLASHFILE.TARGET 0x40200000++0x1fff 0x40202000++0x1fff
                             ~~/demo/arm/flash/word/onenand2g16.bin

; Check OneNAND Flash ID value.
FLASHFILE.GETID

; Erase OneNAND Flash including bad blocks.
FLASHFILE.Erase 0x0--0xfffff /EraseBadBlocks

; Program my_file.bin to OneNAND Flash main area.
FLASHFILE.LOAD my_file.bin 0x0--0xfffff

ENDDO
```

OneNAND Flash controllers generate the ECC data automatically when data is programmed to the main area, so the spare area does not need to be programmed.

©1989-2014 Lauterbach GmbH

Appendix A: ECC (Error Correction Code)

The NAND Flash devices are arranged as an array of pages. Each page consists of 256/512/ 2048 byte data and 8/16/64 byte spare (out of band) area. The spare area is used to store ECC (error correction code), bad block information, and file system dependent data. The ECC location in the spare area is flexible, depending on the customer's flash file system.

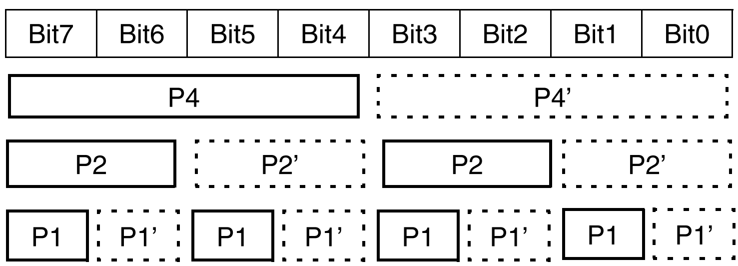
Techniques used to detect and correct error bits include the Hamming, BCH, and Reed Solomon codes.

Hamming codes are most widely used for error detection and correction. According to the Hamming ECC principle, the ECC codes consist of 3 bytes per 256 Kbyte or 3 bytes per 512 Kbyte. ECC codes allow the NAND Flash controller to verify the data and in some cases to correct corrupted data.

How to Generate ECC and to Detect Error

The Hamming ECC can be applied to data sizes of 1 byte, 8 bytes 16 bytes, and so on. The following paragraph shows a simple example for 1 byte (8 bit).

ECC Generation



(+) : XOR

P4=Bit7(+)Bit6(+)Bit5(+)Bit4 P4'=Bit3(+)Bit2(+)Bit1(+)Bit0

P2=Bit7(+)Bit6(+)Bit3(+)Bit2 P2'=Bit5(+)Bit4(+)Bit1(+)Bit0

P1=Bit7(+)Bit5(+)Bit3(+)Bit1 P1'=Bit6(+)Bit4(+)Bit2(+)Bit0

©1989-2014 Lauterbach GmbH

Original Data: 1 0 1 0 1 0 1 0

P4	P4'	P2	P2'	P1	P1'
0	0	0	0	0	0

XOR

Changed Data: 1 0 1 **1** 1 0 1 0

P4	P4'	P2	P2'	P1	P1'
1	0	0	1	0	1

P4	P4'	P2	P2'	P1	P1'
1	0	0	**1**	0	**1**

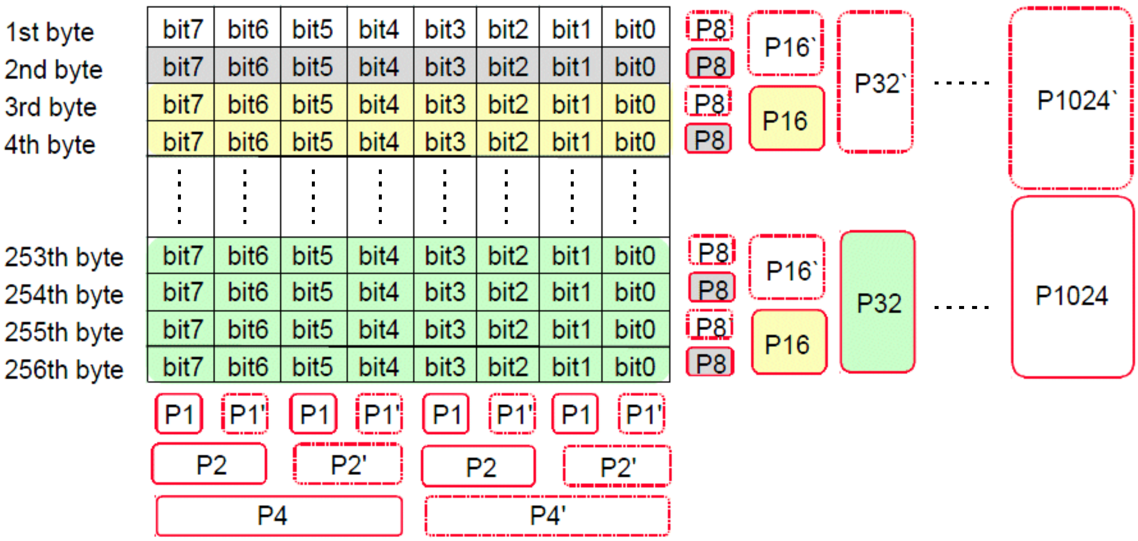
Half bits (3/6) are different -> the error is correctable
All bits are '0' -> no error

So, the error location is at the P4, P2, P1 values (1 0 0 = Bit4)

- P4, P2, P1 = column parity and the error bit position
- P2048, P1024, ..., P8 = line parity and the error byte position

For example, if you get an ECC result like this 01001100 110 (P1024, P512, ..., P2, P1) in the 256byte ECC generation, it means that the error is located in the 6th bit of the 76th Byte.

©1989-2014 Lauterbach GmbH



P8=bit7(+)bit6(+)bit5(+)bit4(+)bit3(+)bit2(+)bit1(+)bit0(+)P8

⋮

P1024=bit7(+)bit6(+)bit5(+)bit4(+)bit3(+)bit2(+)bit1(+)bit0(+)P1024

(+) : XOR

22bit ECC Code = 16bit line parity + 6 bit column parity

	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
ECC0	~P64	~P64’	~P32	~P32’	~P16	~P16’	~P8	~P8’
ECC1	~P1024	~P1024’	~P512	~P512’	~P256	~P256’	~P128	~P128’
ECC2	~P4	~P4’	~P2	~P2’	~P1	~P1’	1	1

22-bit ECC Code Assignment Table

©1989-2014 Lauterbach GmbH

3bytes per 512bytes ECC Codes

24bit ECC Code = 18bit line parity + 6bit column parity

	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
ECC0	~P64	~P64’	~P32	~P32’	~P16	~P16’	~P8	~P8’
ECC1	~P1024	~P1024’	~P512	~P512’	~P256	~P256’	~P128	~P128’
ECC2	~P4	~P4’	~P2	~P2’	~P1	~P1’	~P2048	~P2048’

24-bit ECC Code Assignment Table

©1989-2014 Lauterbach GmbH

Appendix B: Spare Area Schemes

Linux MTD NAND Driver Default Spare Area Schemes

256 Byte Page Size

Offset	Content	Comment
0x0	ECC Byte 0	Error correction code byte 0
0x1	ECC Byte 1	Error correction code byte 1
0x2	ECC Byte 2	Error correction code byte 2
0x3	Autoplace 0	
0x4	Autoplace 1	
0x5	Bad Block Marker	If any bit in this byte is zero, then this block is bad.
0x6	Autoplace 2	
0x7	Autoplace 3	

512 Byte Page Size

Offset	Content	Comment
0x0	ECC Byte 0	Error correction code byte 0 of the lower 256 Byte data in this page
0x1	ECC Byte 1	Error correction code byte 1 of the lower 256 Bytes of data in this page
0x2	ECC Byte 2	Error correction code byte 2 of the lower 256 Bytes of data in this page
0x3	ECC Byte 3	Error correction code byte 0 of the upper 256 Bytes of data in this page
0x4	Reserved	Reserved
0x5	Bad Block Marker	If any bit in this byte is zero, then this block is bad.

©1989-2014 Lauterbach GmbH

0x6	ECC Byte 4	Error correction code byte 1 of the upper 256 Bytes of data in this page
0x7	ECC Byte 5	Error correction code byte 2 of the upper 256 Bytes of data in this page
0x08 - 0x0F	Autoplace 0 - 7	

2048 Byte Page Size

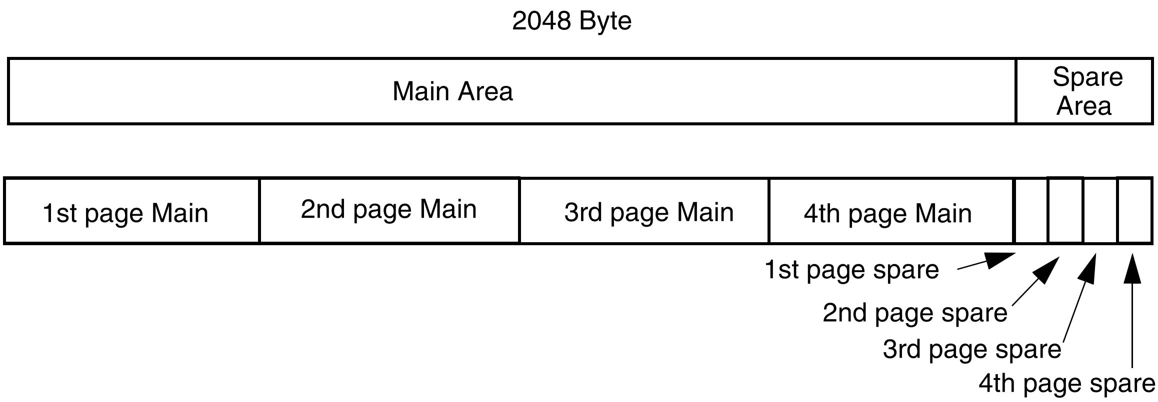
Offset	**Content**	**Comment**
0x0	Bad block marker	If any bit in this byte is zero, then this block is bad.
0x1	Reserved	Reserved
0x02-0x27	Autoplace 0 - 37	
0x28-0x2A	ECC Byte 0-2	Error correction code 3 bytes of the first 256 Byte data in this page
0x2B-0x2D	ECC Byte 3-5	Error correction code 3 bytes of the second 256 Byte data in this page
0x2E-0x30	ECC Byte 6-8	Error correction code 3 bytes of the third 256 Byte data in this page
0x31-0x33	ECC Byte 9-11	Error correction code 3 bytes of the fourth 256 Byte data in this page
0x34-0x36	ECC Byte 12-14	Error correction code 3 bytes of the fifth 256 Byte data in this page
0x37-0x39	ECC Byte 15-17	Error correction code 3 bytes of the sixth 256 Byte data in this page
0x3A-0x3C	ECC Byte 18-20	Error correction code 3 bytes of the seventh 256 Byte data in this page
0x3D-0x3F	ECC Byte 21-23	Error correction code 3 bytes of the eighth 256 Byte data in this page

©1989-2014 Lauterbach GmbH

512B(Small Page): 16Byte Spare Area

Offset	Content	Comment
0x0-0x2	LSN 0-2	Logical Sector Number
0x3-0x4	WC 0-1	Status flag against sudden power failure during write
0x5	BI	Bad block marker
0x6-0x8	ECC Byte 0-2	ECC code for 512KB main area data
0x9-0x0A	S-ECC Byte 0-1	ECC code for LSN data
0x0B-0x0F	Reserved	Reserved

2048B(Large Page): 64 Byte Spare Area



Description of the Spare Area

Offset	Content	Comment
0x0	BI	1st bad block marker
0x1	Reserved	Reserved
0x2-0x4	LSN 0-2	Logical sector number
0x5	Reserved	Reserved
0x6-0x7	WC 0-1	Status flag against sudden power failure during write
0x8-0x0A	ECC Byte 0-2	ECC code for first 512KB main area data

©1989-2014 Lauterbach GmbH

0x0B-0x0C	S-ECC Byte 0-1	ECC Code for first LSN data
0x0D-0x0F	Reserved	Reserved
0x10-0x1F		2nd page spare structure is the same as the 1st page spare
0x20-0x2F		3rd page spare structure is the same as the 1st page spare
0x30-0x3F		4th page spare structure is the same as the 1st page spare