TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Documents	
FLASH Programming	
Serial FLASH Programming User's Guide	1
Introduction How This Manual is Organized	3 3
Contacting Support	3 4
List of Abbreviations	6
Background Knowledge What is a Serial Flash Device? About SPI Interface Controllers in Serial Flash Memories About Blocks and Pages File Name Convention for Serial Flash Drivers	7 7 8 9
Standard Approach Identifying and Running Scripts for Serial Flash Programming If There Is No Script	10 10 12
Scripts for SPI Controllers Establishing Communication between Debugger and Target CPU Configuring the SPI Controller Resetting Default Values Informing TBACE32 about the Serial Elash Begister Addresses (SPI)	13 15 16 17
Informing TRACE32 about the Serial Flash Programming Algorithm Identifying the Correct Driver Binary File for a Serial Flash Device Finding the <serialflash_code> of Your Serial Flash Device Example for SPI Controllers</serialflash_code>	18 20 21 23
FLASHFILE Declaration Examples Declaration Example for STM32F103 (Cortex-M3) Declaration Example for AT91SAM9XE (ARM9) Checking the Identification from the Serial Elash Device	24 24 25 26
Erasing the Serial Flash Device Programming the Serial Flash Device Copying the Serial Flash Memory Modifying the Serial Flash Memory	27 28 29 31

Other Useful Commands	32
Reading the Serial Flash	32
Saving the Serial Flash Device	33
Full Examples	34
Example 1	34
Example 2	36
FLASH Programming via Boundary Scan	37
Example 1 for the SPI Protocol	37
Example 2 for the I2C Protocol	39

Version 06-Nov-2015

05-Feb-15 Added section "Finding the <serialflash_code> of Your Serial Flash Device".

Introduction

This manual describes the basic concept of serial Flash programming.

How This Manual is Organized

- Background Information: Provides information about important terms in serial Flash programming, such as SPI interface controller, block, and page.
- Standard Approach: Describes the fastest way to get started with SPI serial Flash programming. All you need to do is to identify and run the correct script.

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

demo/<architecture>/flash/<*>.cmm

e.g. at91sam9xe-spi.cmm, stm32f10x-spi.cmm, str910-spi.cmm, ...

 New Scripts for SPI Controllers: Describes how you can create a script if there is no demo script for the SPI controller you are using.

Related Documents

A complete description of all serial 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 serial Flash memory devices.
- A list of the supported CPU families for the serial Flash protocol.

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

- Supported Flash devices under: http://www.lauterbach.com/ylist.html
- Supported serial 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.

Lauterbach Homepage						
Support	🕨 🖉 Systeminfo					
About TRACE32	▲ Online Support Contact Lauterbach	🖉 Generate TI	RACE32 Support Information			
	License details	Press following bu	itton to get help on how to generate Su	pport Information:	💡 не	elp / About)
		Company:	Lauterbach	Department:		
		Prefix: Firstname:	Andrea			
		Surname:	Martin Altlaufstrasse 40	P.O. Box:		
		City:	Hoehenkirchen-Siegertsbr.	ZIP Code:	85635	
		Country: Telephone:	Germany ++49-8104-8943-555			
		eMail:	andrea.martin@lauterbach.c	om		
		Product.:	PowerTrace 512MB			
		Target CPU: Hostsystem:	ARM7 TDMI PC Windows 7			
		Compiler:	ARM			
		RealtimeOS:	Nono		5	Safe Mode: 📃
			Generate Support Information:	Save to Clip	board Sa	ave to File

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" on page 12.

The abbreviations vary from chip manufacturer to chip manufacturer.

MOSI or SI	Master output, slave input		
MISO or SO	Master input, slave output		
SCLK or SCK	Serial clock		
SS	Slave select		
SDI	Serial data input		
SDO	Serial data output		
CS or CE	CS = Chip selection CE = Chip enable		
SPI	Serial peripheral interface		
eMMC	Embedded multimedia card		
ММС	Multimedia card		
GPIO	General purpose input/output		
SSI	Synchronous serial interface		

This chapter of the manual is aimed at users who are new to serial 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 serial Flash programming, such as serial flash device, sector, page, and SPI interface controller.

What is a Serial Flash Device?

A serial Flash device is a non-volatile memory that can be electrically erased and reprogrammed. It is used for storing executable code in devices such as DVD players, DSL modems, routers, hard-disk drives, and printers. After power-up, the executable code is downloaded from the serial Flash to RAM, and then executed by the processor. The code in the serial Flash is not changed by the download process and is write-protected.

Reasons for the widespread use of serial Flash devices include:

- Cost effective and space-saving solution thanks to a reduced number of pins
- Low power consumption
- Life cycle of about 20 years

About SPI Interface Controllers in Serial Flash Memories

Serial Flash memories consist of an interface controller (for example, a SPI interface controller) and a Flash memory. Access to the Flash memory is performed by the interface controller on the SPI slave side.



Figure: Processor/Chip and Serial Flash Memory with a SPI Interface

Serial Flash memories are controlled by many kinds of serial interface protocols (SPI, SSP, SSI, SMI, etc.). The protocol of the SPI interface (serial peripheral interface) specifies four signals:

- Slave select (SS)
- Master output, slave input (MOSI)
- Master input, slave output (MISO)
- Serial clock (SCLK)

Most chip manufacturers have proprietary SPI interface controllers (short: SPI controllers) and thus require special *driver binary files* for programming serial Flash memories. These driver binary files are *programs executed by a core in the target* and interact with the master SPI controller for controlling the SPI slave controller in order to program the serial Flash memory.

Once the required driver binary file was loaded to the target, the TRACE32's command group **FLASHFILE** can be used to program and erase the serial Flash memory.

Lauterbach provides driver binary files for a large number of SPI controllers. See further down for instructions how to identify the correct driver for your application.

About Blocks and Pages

The Flash memory of a serial Flash device consists of sectors, and each sector is subdivided into pages; see example diagram below.



 Sector A sector is the minimum size unit for erasing. A sector can have a size of 32, 64, or 256 KBytes. The sector sizes are part of the file names of the algorithms required for serial Flash programming: spi32*.bin, spi64*.bin, spi256*.bin. For more information, see "File Name Convention for Serial Flash Drivers" on page 9.
 Page A page is the minimum size unit for writing and has a size of 256 bytes. The serial Flash drivers for SPI controllers, i.e. the algorithm files, use the following file name convention:

Serial Flash driver name: SPInn_CPU.bin						
where nn is a two-digit number that indicates the size of one sector in KByte (the sector erase size in KByte):						
	nn	Sector	Hex Size			
	32 64	32KB 64KB	0x00x7FFF 0x00xFFFF			
where CPU is the CPU family name.						

The chapter "Standard Approach" provides a compact description of the steps required to program serial Flash memories. This description is intentionally limited to the standard use case.

Overview of the Standard Approach:

- Identify and run the required script for serial Flash programming.
- What to do if there is no script for NAND Flash programming.

The following step-by-step procedures describe the standard approach in detail.

For a detailed description of the serial Flash programming concepts, see "Scripts for SPI Controllers" on page 13.

Identifying and Running Scripts for Serial Flash Programming

Lauterbach provides scripts (*.cmm) for serial Flash programming which can be found below the TRACE32 installation directory. The scripts support serial Flash memories that have a *serial Flash controller with a SPI interface* (short: SPI controller).

Path and file name convention of scripts to be used with SPI controllers:

<TRACE32_installation_directory>/ demo/**<architecture>**/flash/**<cpu_name>-spi**.cmm

To identify and run the required script:

- 1. Make a note of the <cpu_name> printed on the CPU; for example, bcm5357
- 2. Put the <cpu_name> and spi together to form the script name: bcm5357-spi.cmm tbd.

The script file resides in this folder: ~~/demo/mips/flash/bcm5357-spi.cmm

Note: ~~ is replaced by 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 12.

- 3. 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 serial Flash
	programming algorithm (*.bin).
	You do not need to program or select the algorithm.

Example

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

E-Mail Template:

Chip name: _____

Name of serial 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 about how to create your own script (* . cmm), see "Scripts for SPI Controllers" on page 13.

This chapter describes how to create new scripts for serial Flash memories that are equipped with SPI controllers.

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. "Establishing Communication between Debugger and Target CPU", page 15
- 2. "Configuring the SPI Controller", page 16
- 3. "Resetting Default Values", page 17
- 4. "Informing TRACE32 about the Serial Flash Register Addresses (SPI)", page 17
- 5. "Informing TRACE32 about the Serial Flash Programming Algorithm", page 18
- 6. "Checking the Identification from the Serial Flash Device", page 26
- 7. "Erasing the Serial Flash Device", page 27
- 8. "Programming the Serial Flash Device", page 28

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

	; Establish the communication ; between the target CPU and the ; TRACE32 debugger.
	; Configure the SPI controller.
FLASHFILE.RESet	; Reset the serial Flash ; environment in TRACE32 to its ; default values.
FLASHFILE.CONFIG	; Inform TRACE32 about ; - the serial Flash register ; addresses and ; - the CS address of the serial ; Flash
FLASHFILE.TARGET	; Specify the serial Flash ; programming algorithm and where ; it runs in target RAM.
FLASHFILE.Erase	; Erase the serial Flash.
FLASHFILE.LOAD <main_file></main_file>	; Program the file to serial Flash.

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.

Establishing Communication between Debugger and Target CPU

Serial 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:

Specify your target CPU.
Establish the communication between the debugger and the target CPU.
; Select STM32F103 as the target CPU.
; Establish the communication between the ; debugger and the target CPU.

©1989-2015 Lauterbach GmbH

Programming a serial Flash device requires an appropriate initialization of the serial Flash interface. The following settings might be necessary:

- Enable the clock (SCLK).
- Configure the registers of the serial Flash interface, such as clock, master/slave, data width, etc.
- Configure the serial Flash pins if they are muxed with other functions of the CPU.

Example: In the CPU STM32F103, all the SPI pins are muxed with the GPIOs.



Using the commands below, the pins of the serial Flash interface (SPI) can be configured for the CPU STM32F103. The resulting configuration allows the CPU to switch from the GPIO A4-A7 to SS,SCLK, MISO, and MOSI.

PER.Set SD:0x40021014 %Long 0x114; Enable the Clock for thePER.Set SD:0x40021018 %Long 0x100C; SPI & PIO.PER.S SD:0x40010800 %Long 0xbbb34444; Switch the PIO.A group toPER.S SD:0x40010810 %Long 0xFFFFFF; the SPI function.PER.S SD:0x40013000 %WORD 0x34F; Configure the SPIPER.S SD:0x40013010 %WORD 0x7; controller.

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

FLASHFILE.RESet

Reset the serial Flash environment in TRACE32 to its default values.

Informing TRACE32 about the Serial Flash Register Addresses (SPI)

The following command is used to inform TRACE32 about the various register addresses (Flash declaration).

Chip selection is controlled by one GPIO pin. FLASHFILE.CONFIG <SPI Tx reg> <SPI Rx reg> <CS GPIO reg> <CS bit>

Chip selection is controlled by the SPI controller. **FLASHFILE.CONFIG** <SPI Tx reg> <SPI Rx reg> <SPI ChipEnable reg>

Parameters for the FLASHFILE.CONFIG command				
<spi reg="" tx=""></spi>	The transmit data register			
<spi reg="" rx=""></spi>	The receive data register			
<cs gpio="" reg=""></cs>	The chip selection GPIO register which is connected to the Flash chip selection			
<cs bit=""></cs>	The chip selection bit in the GPIO CS register			
<spi chipenable="" reg=""></spi>	The chip enable register of the serial flash			

For information about the register addresses and the CS bit, refer to the manufacturer's microcontroller manual and the schematics.

Example 1

Example 2

©1989-2015 Lauterbach GmbH

Informing TRACE32 about the Serial Flash Programming Algorithm

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

FLASHFILE.TARGET <code_range> <data_range> <file>

Specify the serial Flash programming driver and where it runs in the target RAM.

Parameters

<code_range>

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



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.



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>*.

```
<br/>
<buffer_size> =<br/>
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 serial Flash programming algorithm in one call.

<file>

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

~~/demo/<architecture>/flash/byte/

Note: ~~ 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 Serial Flash Device**" on page 20.

Identifying the Correct Driver Binary File for a Serial Flash Device

There are two ways to find the correct *.bin file:

- You can identify the *.bin file via our website, as described in this section.
- Alternatively, run a PRACTICE script (*.cmm), as described in "Finding the <serialflash code> of Your Serial Flash Device", page 21.

To identify the correct *.bin file:

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

Supported FLASH Devices		LAUTERBA	ICH
[content]	Ţ	Supported Flash Devices	-
HYNX	^	home > Tool	Chain
INFINEON			
MACRONIX			
MICRONAS			
NEC	_	Supported FLASH Devi	ices
NXP OKI			
RENESAS		NAND FLASH devices are marked in GREEN.	
SANDISK	_	SERIAL FLASH devices are marked in RED.	

5. Locate the desired Flash device.

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

- 6. Identify the correct *.bin file based on the name of the Flash device. The following example illustrate how to do this.
 - The example below illustrates how to apply the file name convention in practice.
 - For information about the file name convention for driver binary files, see "File Name Convention for Serial Flash Drivers" on page 9.

tbd.

The following step-by-step procedure helps you find the <serialflash_code> of your serial Flash device. Based on the <serialflash_code>, you can then identify the correct *.bin file.

To find the <serialflash_code>:

1. Run the following PRACTICE script file (*.cmm) from the TRACE32 demo folder:

```
CD.DO ~~\demo\etc\flash\find_spidef.cmm
;The path prefix ~~ expands to the system directory of TRACE32,
;by default C:\t32.
```

If this demo script is missing, you can download it from www.lauterbach.com/scripts.html.

The Find SPI flash code dialog opens.

- 2. Under **spi flash parameters**, make your settings.
 - You can find the required information in the serial Flash documentation of the manufacturer.
 - For an example of where to search for the information, see figure and table below:

	Find SPI flash code	- • •	
	spi flash parameters - Address Cycle: - Erase size by cmd (D8h): - Erase size by cmd (D8h):		In the serial Flash documentation of
В	Support Page Program cmd (02h): ON Support Read Flag Status cmd (70h) ON 	OFF OFF (ex. Numonyx)	the manufacturer, search for terms like:
С	- Support Read Bank Register cmd (16h) ON	OFF (ex. Spansion)	 erase capability sector
	SPI4B64FS	Find	- granularity in order to find the correct value.

	Instruction	Description	One-Byte Instruction Code (BIN)	One-Byte Instruction Code (HEX)	Address Bytes	Dummy Clock Cycle	Data Bytes
А	READ	Read Data Bytes	0000 0011	03h	3/4	0	1 to ∞
В	PP	Page Program	0000 0010	02h	3/4	0	1 to 256
С	RFSR	Read Flag Status Register	0111 0000	70h	0	0	1 to ∞

- 3. Click Find.
 - The **code** box displays the **<serialflash_code>** of your serial Flash device.
 - If the code box displays unknown, then proceed as described in "If There is No Script".
- 4. Make a note of the displayed <serialflash_code>.
- 5. Click **End** to close the dialog.
- 6. Identify the correct *.bin file based on the <serialflash_code>. The following example illustrates how to do this.
 - "Example for SPI Controllers", page 23
 - For information about the file name convention for driver binary files, see "File Name Convention for Serial Flash Drivers" on page 9.

Target:

- ARM-based **STM32F10X** CPU with the SPI controller **stm**
- Serial Flash device M25P64

Taken together, the **Code** column and the **Controller** column make up the file name of the serial Flash driver binary file: **spi64_stm.bin**. The number **64** indicates the size of one sector in KByte (the sector erase size in KByte).

Tool Chain	ST Microelectronics I	N V		
Supported Compilers	31 Microelectronics I	N.V.		
Supported Host	CPU	CONTROLLER	COM	IMENT
Operating Systems	SPEAR300	generic	NAN	D
Supported Flash Devices	•	:	:	
Supported NAND/Serial Flash	•		•	
Controller	STM32F103ZE	cortexm3	NAN	D
Supported Target	STM32F10X	stm	SPI	
Operating Systems	CTD040		ODI	
Supported Compilers	ST Microelectronic	company	CODE	COMPAT
Operating Systems	ITPE	COMPANY	CODE	COMMENT
Supported Flash Devices	M25FL128	STM	SPI256	Serial Flash, 256KB sector
Supported NAND/Serial Flash	M25P64	STM	SPI64 🌱	Serial Flash
Controller	M28F101	STM	M28F256	
Supported Target	M28F102	STM	M28F256	
Operating Systems	M28F201	STM	M28F256	
Supported Tool Integrations				
 Supported Simulators/Virtual Prototypes/Target Servers 				
⊞ Support ■				

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

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

This results in the following command line:

; Specify the serial 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/spi64_stm.bin

Declaration Example for STM32F103 (Cortex-M3)

STM32F103 (ARM Cortex-M3) from ST Microelectronics, M25P64 (64Mbit)

- SPI Tx register/SPI Rx register: 0x4001300C
- Sector size: 64Kbytes (0x0--0xFFFF)

•••

Driver file: ~~/demo/arm/flash/byte_thumb/spi64_stm.bin

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

Declaration Example for AT91SAM9XE (ARM9)

AT91SAM9XE (ARM9) from ATMEL, M25P64 (64Mbit)

- SPI Tx register/SPI Rx register: 0xFFFC800C / 0xFFFC8008
- Sector size: 64Kbytes (0x0--0xFFFF)

...

Driver file: ~~/demo/arm/flash/byte/spi64_at91sam9xe.bin

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

Checking the Identification from the Serial Flash Device

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

FLASHFILE.GETID

Get the ID values of the serial Flash device.

; Open the TRACE32 AREA window. AREA.view ; Check the access to the serial Flash device ; by getting the manufacturer ID and the device ID. FLASHFILE.GETID



The following command is available to erase serial Flash devices:

FLASHFILE.Erase <range>

Erase the serial Flash.

NOTE:	The FLASHFILE.Erase command has a time limitation.
	• TRACE32 has to get a response (success or failure) from the serial Flash in
	3 minutes.
	 If you get an error message because of the FLASHFILE.Erase time limita- tion, then divide the original range into several smaller ranges.

Example 1: Original erase range

```
; Erase 2MB starting from 0x0. FLASHFILE.Erase 0x0--0x1FFFFF
```

Example 1 cont'd: Two smaller erase ranges

```
; Erase 2MB starting from 0x0.
FLASHFILE.Erase 0x0--0xFFFFF
FLASHFILE.Erase 0x100000--0x1FFFFF
```

Example 2: Erase 8 MB in increments of 1 MB using a RePeaT loop

```
&addr=0x0
RePeaT 8.
(
    FLASHFILE.Erase &addr++0xFFFFF
    &addr=&addr+0x100000
)
```

The following commands are available to program the serial Flash:

 FLASHFILE.LOAD <file> [<address> | <range>]
 Program serial Flash.

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

Example 1

```
; Program the contents of my_file.bin to the serial Flash memory starting ; at address 0 \times 0 \, .
```

FLASHFILE.LOAD my_file.bin 0x0

Example 2

; Verify the contents of my_file.bin against the serial Flash memory ; starting at address 0x0.

FLASHFILE.LOAD my_file.bin 0x0 /ComPare

The following command is available to copy:

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

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

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

Example 1

- ; Copy the 1MB virtual memory data at 0x0 to the serial Flash address
- ; at 0x100000.

; VM: The virtual memory of the TRACE32 software.

FLASHFILE.COPY VM:0x0--0xFFFFF 0x100000

Result (1)

🕱 B::DATA.DUMP VM:0x0 /DIALOG						
VM:0x0	Find Modify	Long 🔽 🗌 E	Track 🗹 Hex			
address VM:00000000	0 4 +4E56533B 76655220 6F697	8 C 0123 369 75203A6E ;SVN	456789ABCDEF			
VM:00000010 VM:00000020	6F6E6B6E 0A0D6E77 3B0A0 58353535 43504D2F 58363	D3B 43504D20 nkno 535 43504D2F 555X	wnęę;ęę;uMPC /MPC556X/MPC			
VM:00000030 VM:00000040	58333535 504D202C 33363 36354350 78784D33 6F4D2	543 532F4D78 553X 820 6F63616E PC56	,∟MPC563xM/S ≚ 3Mxx⊔(Monaco 📈			
VM:00000050 VM:00000060	4D202C29 36354350 0D203 2D2D2D2D 2D2D2D2D 2D2D2	437 2D203BOA),M D2D 2D2D2D2D	B::FLASHFILE.D	UMP 0x100000		
VM:00000070 VM:00000080	20202020 20202020 20202 20202020 20202020	D2D 2D2D2D2D2D	- 0x100000	MAIN 💌 🍈	Find	Long 🔽 🗌 Track 🗹 H
VM:00000090 VM:000000A0	20202020 8F432038 20208 20202020 3265203A 0D203	030 4D203BOA	address 0000000000100000	0 +4E56533B 7665522	4 8 0 6F697369 75203A8	C 0123456789ABCDEF
VM:000000000 VM:000000000	73656572 65606163 380A0	D20 3B0A0D20 rees 020 3A202020 Aut		6F6E6B6E 0A0D6E7 58353535 43504D2	7 3B0A0D3B 43504D2 F 58363535 43504D2	20 nknownፍኑ;ፍኑ;ԼMPC 🧵 2F 555X/MPC556X/MPC
VM:000000E0	49455220 3B0A0D20 61745	320 20737574 LREI	000000000000000000000000000000000000000	58333535 504D202 36354350 78784D3	C 33363543 532F4D7 3 6F4D2820 6F63616	/8 553X,∟MPC563xM/S 🛛 🞽 8E PC563Mxx⊥(Monaco 🗖
VM:00000100 VM:00000110	0A0D2064 2D2D203B 2D2D2 2D2D2D2D 2D2D2D2D 2D2D2	D2D 2D2D2D2D d	00000000000100050	4D202C29 3635435 2D2D2D2D 2D2D2D2	0 0D203437 2D203B0 D 2D2D2D2D 2D2D2D2	JA),∟MPC5674⊔≨⊭;⊔- 2D
VM:00000120 VM:00000130	2D2D2D2D 2D2D2D2D 2D2D2 2D2D2D2D 2D2D2D2D	D2D 2D2D2D2D2D	0000000000100070 000000000100080	2D2D2D2D 2D2D2D2 2D2D2D2D 2D2D2D2	D 2D2D2D2D 2D2D2D2 D 2D2D2D2D 2D2D2D2	2D 2D
VM:00000140	58583535 23284020 0A0D2	029 620A0D20 55XX	_{ 0000000000100090 00000000001000A0	0A0D202D 6F43203 20202020 3265203	B 20206572 2020202 A 0D203030 4D203B0	20 -uƙ⊧;uCoreuuuuuu)A uuuu:ue200uƙ⊧;uM
122		•	00000000000100080	66756E61 7574636 73656572 656C616	1 20726572 46203A2 3 3B0A0D20 3B0A0D2	20 anufacturer:_F 20 reescale_⊊⊧;_⊊⊧; ≣ .
			00000000000100000 000000000001000E0	49455220 3B0A0D2	8 20202020 3A20202 0 61745320 2073757	20 LAuthor LLLL: 24 LREILSE; Status
Γ	Data is copied from	the	00000000000000000000000000000000000000	0A0D2064 2D2D203	0 60657220 6573616 B 2D2D2D2D 2D2D2D2	35:_release 2D d_윾\;
(CPU to the serial F	lash	000000000000000000000000000000000000000	20202020 2020202 20202020 2020202 20202020	D 20202020 20202020 D 2020202020 20202020 D 2020202020 20202020	2D
			00000000000100130	58583535 2328402	0 0A0D2029 620A0D2	20 55XXL@(#)LՁեշՁեն 💌

Example 2

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

against the target range data.

Example 3

; Copy the 1MB serial Flash data at 0x0 to the serial Flash ; at 0x800000. FLASHFILE.COPY 0x0--0xFFFFF 0x800000

; Verify the 1MB serial Flash data between 0x0 and 0x800000. **FLASHFILE.COPY 0x0--0xFFFFF 0x800000 /ComPare**

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

```
FLASHFILE.Set [<address> | <range>] %<format> <data> Modify the contents of the serial 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

Result (1)

B::FLASHFILE.D	UMP 0x100	000				×
0x100000	MAIN 💌	Find.	Modify	Long	🖌 🗌 Track (~
addross	0	4	8	C 01234	456789ABCDEF	
0000000000 100000	12345678	FFFFFFFF F	FFFFFFF FFF	FFFFF ×V4%	FFFFFFFFFFF	~
000000000000000000000000000000000000000	TEFFFFFF	FFFFFFFF F	FFFFFFF FFF	FFFFF FFFF	FFFFFFFFFFF	
00000000000100020	FFFFFFF	FFFFFFF F	FFFFFFF FFF	FFFFF FFFF	FFFFFFFFFFF	
00000000000100030	FFFFFFF	FFFFFFF F	FFFFFFF FFF	FFFFF FFFF	FFFFFFFFFFF	Y
00000000000100040	FFFFFFF	FFFFFFF F	FFFFFFF FFF	FFFFF FFFF	FFFFFFFFFFF	*
00000000000100050	FFFFFFFF	FFFFFFF F	FFFFFFF FFF	FFFFF FFFF	FFFFFFFFFFF	¥
	<				\geq	

Result (2)

B::FLASHFILE.DU	UMP 0x100F80	
0x100F80	MAIN 💌 👘 Find Modify Long 💌	Track 🗹
address	0 4 8 C_0123456	789ABCDEF
00000000000000000000000000000000000000	+0000000 0000000 0000000 0000000 0000000	
00000000000000000000000000000000000000	COLOCUE COLOCUE COLOCUE COLOCUE COLOCUE COLOCUE DODOLOCO COLOCUE COLOCUE COLOCUE PREFERENTE PREFERENTE PREFERENTE PREFERENTE FRENERENTE PREFERENTE PREFERENTE PREFERENTE FRENERENTE PREFERENTE FRENERENTE PREFERENTE PREFERENTE FRENERENTE PREFERENTE FRENERENTE PREFERENTE PREFERENTE	
00000000000000000000000000000000000000	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF	· · · · · · · · · · · · · · · · · · ·
		1. < 0.000 million

Example 3

; A serial Flash has 64KB per sector (0x10000). ; Write data 0x0 from 0x100000 to 0x12FFFF in the serial Flash. FLASHFILE.Set 0x10000++0xFFFF %Long 0x0 FLASHFILE.Set 0x120000++0xFFFF %Long 0x0

©1989-2015 Lauterbach GmbH

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

Reading the Serial Flash

The following command allows to read the serial Flash memory.

FLASHFILE.DUMP [<address>] [/<format>]

Display a hex-dump of the serial Flash.

Example

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

Result

🟶 B::flashfile.dum	ip 0x1000						
0x1000	MAIN 🔊	Fire	nd Mo	dify	Long 💌	Track	🕑 Hex
address	0	4	8		012343676	JABCDEF	
00000000000001000	♦E3510201	31510003	31A01201	31A02202	12Q550011	1812"81	~
00000000000001010	3AFFFFFA	E3510102	31510003	31A01081	AFF: SAQE &	NQ193801	
00000000000001020	31A02082	3AFFFFFA	E3A00000	E1530001	3_015FF:N	NAESNSE	
00000000000001030	20433001	21800002	E15300A1	204330A1	10C_1808!1	NSEGOCL	\sim
00000000000001040	218000A2	E1530121	20433121	21800122	2 No !! ASE!	1C_" 18!	~
00000000000001050	E15301A1	204331A1	218001A2	E3530000	1155111CL2	18 NUSS	
00000000000001060	11802222	11A01221	1AFFFFEF	E35C0000	""811181F	FF100/5	
00000000000001070	42600000	E1A0F00E	E13C0000	42600000	NN^B8585N	8<589,00	
00000000000001080	E1A0F00E	33A00000	01A00FCC	03800001	8585NN832	SA S S N S E I O H H U O X	
00000000000001090	E1A0F00E	E3510801	21A01821	23A02010	8585188Q5!	1011.0#	
0000000000000010A0	33A02000	E3510C01	21A01421	22822008	N_83%&Q5!	10 8 8" 40 5-2"	
0000000000000010B0	E3510010	21A01221	22822004	E3510004	3 NQ512814	12" FNQ5 -	
000000000000010C0	82822003	908220A1	E35C0000	E1A00233	E 888 89N X-221-200	8/≣3201	
0000000000000010D0	42600000	E1A0F00E	E52DE004	EB000230	NN^B8585	5-50206	
0000000000000010E0	E3A00000	E49DF004	E1A00000	E1A00000	NNAE EF 9E N UUOB TOD 4 U	0810081	
000000000000010F0	E1A00000	E1A00000	E1A00000	E1A00000	NNAENNAEN	0850085	
00000000000001100	E3510000	0A000032	42611000	E1B0C000	88Q528848	8aBNS85	
00000000000001110	42600000	E2512001	11500001	03A00000	NN BL-QE1	NP1008	
00000000000001120	81110002	00000002	9A000026	E3A02000	SN18SNNN8	NN9 N _ AE UUA U⊷03	
00000000000001130	E3510201	31510000	31A01201	32822004	12Q500Q11	181 5 _22	
00000000000001140	3AFFFFFA	E3510102	31510000	31A01081	AFF: MAQS N	NQ111881	
00000000000001150	32822001	3AFFFFFA	E2522003	BAUDOODE	H-22AFF:	RESNNE	
0000000000001160	E1500001	20400001	E15000A1	204000A1	HNP1100_1	NP51N@_	-
00000000000001170	E1500121	20400121	E15001A1	204001A1	11P111@_1	19111@L	×
	<						≥:

The following command is available to save the contents of the serial Flash memory to a file.

FLASHFILE.SAVE <file> <range>

Save the contents of the serial Flash memory into *<file>*.

Example

- ; Save 1MB of the serial Flash data starting at 0x0 to the file
- ; my_dump.bin.

FLASHFILE.SAVE my_dump.bin 0x0--0xFFFFF

Full Examples

Example 1

CPU:	STM32F103 (Cortex-M3 core)
Serial Flash:	M25P64 (STMicroelectronics, 64Mbit)
Serial Flash CS:	Connected to GPIO B.2

RESet SYStem.RESet SYStem.CPU STM32F103ZE SYStem.Up gosub enable_SPI ; Reset the Flash declaration within TRACE32. FLASHFILE.RESet ; For the serial Flash CS connected GPIO B.2 (group B, port number2) ; GPIO B.2 data output register 0x40010C0C <SPI Tx reg> <SPI Rx reg> <CS GPIO reg> <CS bit> ; FLASHFILE.CONFIG 0x4001300C 0x4001300C 0x40010C0C 0x2FLASHFILE.TARGET 0x2000000++0x1fff 0x20002000++0x1FFF ~~/demo/arm/flash/byte_thumb/spi64_stm.bin ; Read Flash manufacture and device ID. FLASHFILE.GETID ; Erase serial Flash. FLASHFILE.Erase 0x0--0xFFFFF ; Program my_file.bin to serial Flash.

FLASHFILE.LOAD my_file.bin 0x0

; Save to my_dump.bin from serial Flash FLASHFILE.SAVE my_dump.bin 0x0--0xFFFFF enable_SPI: ; Clock Enable related with the SPI and GPIO group PER.S SD:0x40021014 %Long 0x114 ; FSCM Clock Enable PER.S SD:0x40021018 %Long 0x100C ; SPI1 & GPIO A/B Enable PER.S SD:0x40010C00 %LONG 44444344 ; GPIOB.2 output PER.S SD:0x40010C0C %LONG 00000004 ; deselect the SPI FLASH CS(GPIOB.2) PER.S SD:0x40013000 %WORD 0x30F ; SPI_Init PER.S SD:0x40013010 %WORD 0x7 ; SPI_CRC pol &spi_enable=Data.word(SD:0x40013000)|0x40 PER.S SD:0x40013000 %WORD & spi_enable ; SPI1 Enable return

ENDDO

Example 2

CPU:	The STR91x is based on a ARM966E core. It has a SSP protocol which is similar to a STM SPI controller.
Serial Flash:	ST Microelectronics M25P64 (64Mbit) Serial Flash is connected to GPIO 5.7 on the STR91x. Then it is controlled by GPIO5 register (0x5800B3FC).
; Select str910 as t ; Establish the comm SYStem.RESet SYStem.CPU STR910 SYStem.Option ResBre SYStem.JtagClock RTC SYStem.Up	arget CPU. Munication between the debugger and the target CPU. Mak OFF K
; Enable SSP (Synchr PER.Set ASD:0x5C0020 PER.Set ASD:0x5C0020 PER.Set ASD:0x5C0020 PER.Set ASD:0x5C0020 PER.Set ASD:0x5C0020 PER.Set ASD:0x5C0020 PER.Set ASD:0x5C0070 PER.Set ASD:0x5C0070 PER.Set ASD:0x5C0070	<pre>onous Serial Peripheral) & configuration of pins. 18 %Long 0x80300 20 %Long 0x80300 28 %Long 0x80300 00 %Long 0xb0 58 %Long 0x40 00 %Long 0x0 78 %Long 0x40 00 %Long 0x5c7 ;SSP0_CR0 10 %Long 0x2 ;SSP0_PR 04 %Long 0x2 ;SSP0 Enable</pre>
FLASHFILE.RESet	I Tx Reg> <spi reg="" rx=""> <cs gpio="" reg=""> <cs bit=""></cs></cs></spi>
; <pre> <co FLASHFILE.TARGET 0x4</co </pre>	<pre>ode_range> <data_range> <algorithm_file> 000000++0x1FFF 0x4002000++0x1FFF</algorithm_file></data_range></pre>
; Programming for th FLASHFILE.GETID FLASHFILE.ERASE 0x0- FLASHFILE.LOAD * 0x FLASHFILE.SAVE dump. ENDDO	e serial Flash -0xFFFFF :0 bin 0x00xFFFFF

The **BSDL** commands of TRACE32 are used to program external FLASH memories via boundary scan. Important BSDL-specific steps are:

- Check that the bypass mode works.
- Check that the IDCODE matches.
- Define the FLASH pin connection.
- Enable serial FLASH programming via boundary scan and define the flash type.

Serial FLASH programming then continues with the **FLASHFILE** commands described in this manual. The following PRACTICE script (*.cmm) illustrates the BSDL-specific steps by way of these examples:

- Example 1 for the SPI protocol
- Example 2 for the I2C protocol

Example 1 for the SPI Protocol

CPU:	AT91SAM3U4
Serial FLASH:	M25P64 (STMicroelectronics, 64Mbit)
Pin connection:	SPI_CE: Port A16 SPI_SCK: Port A15 SPI_MOSI: Port A14 SPI_MISO: Port A13

```
SYStem.JtagClock 15.Mhz
                                ; set JTAG clock
                                ; reset boundary scan configuration
BSDL.RESet
BSDL.FILE ./sam3u4e lgfp144.bsd ; load the required BSDL file
BSDL.HARDRESET
                                ; toggle TRST_N pin
BSDL.SOFTRESET
                                 ; do a sequential JTAG reset
IF BSDL.CHECK.BYPASS()
                                ; check, if BYPASS mode works
(
   IF BSDL.CHECK.IDCODE() ; check, if the IDCODE matches
      BSDL.FLASH.IFDefine RESet ; reset the boundary scan flash
                                ; configuration
       BSDL.FLASH.IFDefine SPI 1. ; define boundary scan flash interface:
                                 ; - protocol: SPI
                                 ; - SPI flash memory connected to IC1
                                 ; of the boundary scan chain
      BSDL.FLASH.IFMap CE PA16 ; map generic SPI pin CE to port PA16
      BSDL.FLASH.IFMap SCK PA15 ; map generic SPI pin SCK to port PA15
      BSDL.FLASH.IFMap SI PA14 ; map generic SPI pin SI to port PA14
      BSDL.FLASH.IFMap SO PA13 ; map generic SPI pin SO to port PA13
                                ; Initialize boundary scan chain to
      BSDL.FLASH.INIT SAFE
                                 ; safe values according to SAFE state
                                 ; from BSDL file
                                ; Enable serial flash programming via
      FLASHFILE.BSDLaccess ON
                                 ; boundary scan
      FLASHFILE.BSDLFLASHTYPE SPI64 ; define serial flash type
                                 ; get the SPI flash memory ID
      FLASHFILE.GETID
       ; continue with serial flash programming, e.g.:
       ; FLASHFILE.DUMP 0x0
       ; FLASHFILE.ERASE 0x0--0xFFFFF
      ; FLASHFILE.LOAD * 0x0
   )
)
ENDDO
```

Example 2 for the I2C Protocol

CPU: AT91SAM3U4 eMMC FLASH: ST24C08 (STMicroelectronics, 8kbit) Pin connection: I2C_SCL: Port A25 I2C SDA: Port A24

```
SYStem.JtagClock 15.Mhz
                               ; set JTAG clock
BSDL.RESet
                                ; reset boundary scan configuration
BSDL.FILE ./sam3u4e_lqfp144.bsd ; load the required BSDL file
BSDL.HARDRESET
                               ; toggle TRST N pin
BSDL.SOFTRESET
                                ; do a sequential JTAG reset
IF BSDL.CHECK.BYPASS()
                               ; check, if BYPASS mode works
(
  IF BSDL.CHECK.IDCODE() ; check, if the IDCODE matches
   (
     BSDL.FLASH.IFDefine RESet ; reset the boundary scan flash
                                ; configuration
     BSDL.FLASH.IFDefine I2C 1. ; define boundary scan flash interface:
                                ; - protocol: I2C
                                ; - I2C flash memory connected to IC1 of
                                  the boundary scan chain
                                ;
     BSDL.FLASH.IFMap SCL PA25 ; map generic I2C pin SCL to port PA25
     BSDL.FLASH.IFMap SDA PA24 ; map generic I2C pin SDA to port PA24
     BSDL.FLASH.INIT SAFE
                               ; Initialize boundary scan chain to safe
                                ; values according to SAFE state
                                : from BSDL file
     FLASHFILE.BSDLaccess ON ; Enable serial flash programming via
                                ; boundary scan
     FLASHFILE.BSDLFLASHTYPE I2C08 ; define serial flash type
     FLASHFILE.GETID
                               ; get the I2c flash memory ID
     ; continue with serial flash programming, e.g.:
     ; FLASHFILE.DUMP 0x0
     ; FLASHFILE.ERASE 0x0--0x3FF
     ; FLASHFILE.LOAD * 0x0
    )
)
ENDDO
```