LAUTERBACH
DEVELOPMENT TOOLS

Tips to Solve NOR FLASH Programming Problems

Release 09.2023

MANUAL

Tips to Solve NOR FLASH Programming Problems

Tips to Solve NOR FLASH Programming Problems

Version 09-Oct-2023

Introduction

The goal of this manual is to provide tips to solve problems that might arise while programming off-chip NOR FLASHs by TRACE32.

An introduction to the FLASH programming can be found in the **"Onchip/NOR FLASH Programming User's Guide"** (norflash.pdf).

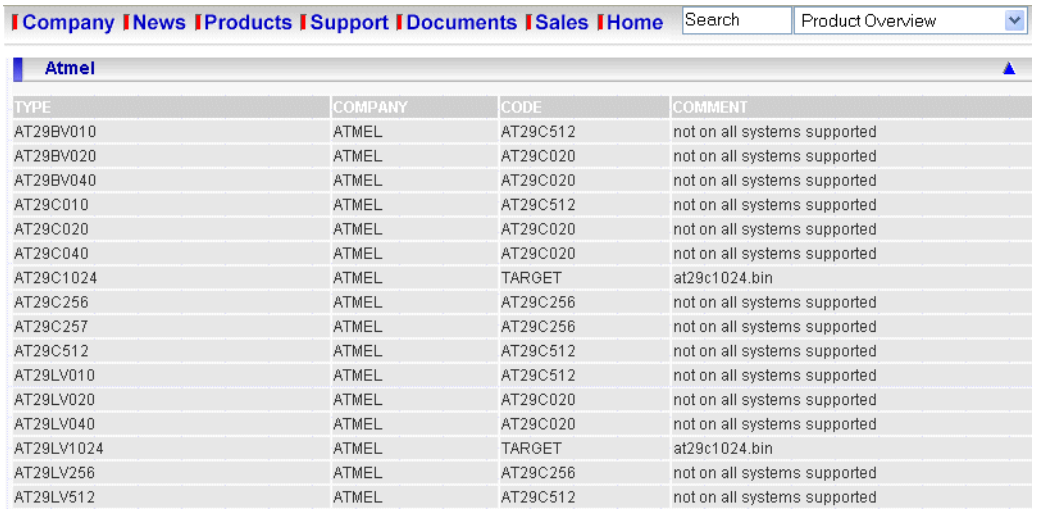
For a complete list of all FLASH programming commands refer to the **FLASH** command group.

Nearly all off-chip NOR FLASHs can be programmed via:

- TRACE32 tool-based FLASH programming
- target-controlled FLASH programming

Just a few FLASH devices work only via target-controlled FLASH programming. This affects the following FLASH devices:

- FLASHs that are connected to a PowerPC CPU in true little endian mode.
- FLASHs for which the CODE column in **https://www.lauterbach.com/ylist.html** shows “TARGET”.
- FLASHs for which the COMMENT column in **https://www.lauterbach.com/ylist.html** shows “not on all systems supported” may not be programmed via TRACE32 tool-based FLASH programming for all processor architectures.



Company | News | Products | Support | Documents | Sales | Home Search Product Overview

Atmel

TYPE	COMPANY	CODE	COMMENT
AT29BV010	ATMEL	AT29C512	not on all systems supported
AT29BV020	ATMEL	AT29C020	not on all systems supported
AT29BV040	ATMEL	AT29C020	not on all systems supported
AT29C010	ATMEL	AT29C512	not on all systems supported
AT29C020	ATMEL	AT29C020	not on all systems supported
AT29C040	ATMEL	AT29C020	not on all systems supported
AT29C1024	ATMEL	TARGET	at29c1024.bin
AT29C256	ATMEL	AT29C256	not on all systems supported
AT29C257	ATMEL	AT29C256	not on all systems supported
AT29C512	ATMEL	AT29C512	not on all systems supported
AT29LV010	ATMEL	AT29C512	not on all systems supported
AT29LV020	ATMEL	AT29C020	not on all systems supported
AT29LV040	ATMEL	AT29C020	not on all systems supported
AT29LV1024	ATMEL	TARGET	at29c1024.bin
AT29LV256	ATMEL	AT29C256	not on all systems supported
AT29LV512	ATMEL	AT29C512	not on all systems supported

For detailed information about both FLASH programming methods please refer to the **“Onchip/NOR FLASH Programming User’s Guide”** (norflash.pdf)

General Recommendation

If you start with a new FLASH device, it is recommended to set up a script that uses TRACE32 tool-based FLASH programming first. Errors are less likely with this method because less target resources are required.

If the script that uses TRACE32 tool-based FLASH programming runs faultless, you can almost be sure that:

- The FLASH declaration is correct.
- The bus configuration registers for the FLASH devices are set up correctly.
- The interface between the CPU and the FLASH devices on your target hardware works faultless.
- TRACE32 can erase and program the FLASH devices correctly.

After TRACE32 tool-based FLASH programming works correctly, you can convert this script to a target-controlled FLASH programming script. This is advisable because target-controlled FLASH programming is much faster. Refer to **"Conversion of the Tool-based to Target-controlled FLASH Programming"** in Tips to Solve NOR FLASH Programming Problems, page 49 (flash_diagnosis.pdf) for details.

TRACE32 Error Messages

One of the following error messages is displayed, when a FLASH erasing/programming error occurs:

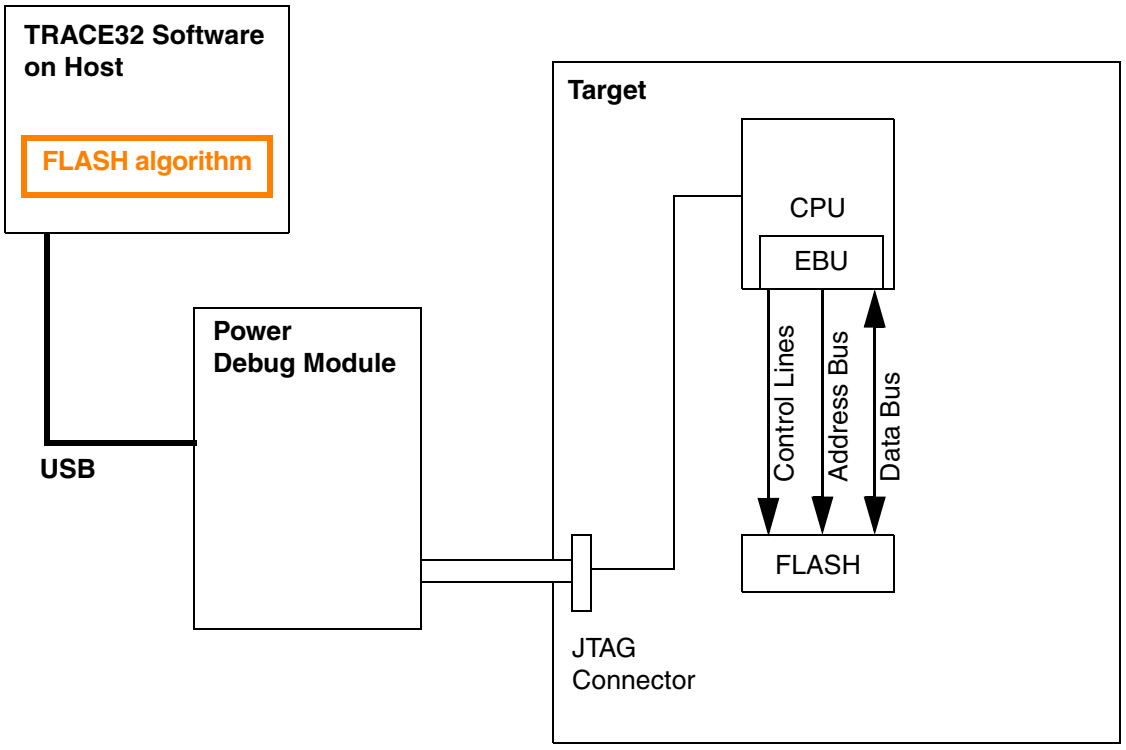
TRACE32 Error Messages	
programming error	Error in programming the FLASH. Proceed as described in **"General Course of Action in the Case of Problems (Tool-based)"** in Tips to Solve NOR FLASH Programming Problems, page 10 (flash_diagnosis.pdf).
erase error	Error in erasing the FLASH. Proceed as described in **"General Course of Action in the Case of Problems (Tool-based)"** in Tips to Solve NOR FLASH Programming Problems, page 10 (flash_diagnosis.pdf).
bus error	There is no memory at the address range declared for the FLASH. Please check the bus configuration for the FLASH device. Refer to **"Checking the Bus Configuration"** in Tips to Solve NOR FLASH Programming Problems, page 50 (flash_diagnosis.pdf) for details.

data alignment error	Wrong or incorrect alignment used while writing the FLASH data. It is strongly recommended to use a format option equivalent to the *<bus_width>* in the FLASH declaration for all write accesses.
	
	`FLASH.Create … Long`
	`FLASH.Program ALL`
	`Data.LOAD.Elf demo.elf /Long`
	`FLASH.Program OFF`

For target-controlled FLASH programming the following error message may be displayed:

TRACE32 Error Messages	
FLASH algorithm did not execute completely	Proceed as described in **"General Course of Action in the Case of Problems (Target-controlled)"** in Tips to Solve NOR FLASH Programming Problems, page 36 (flash_diagnosis.pdf). Make sure that the data cache is switched off, otherwise TRACE32 can not read/update the FLASH status registers.

If TRACE32 tool-based FLASH programming is used, the FLASH algorithm is part of the TRACE32 software and runs on the host.



EBU=External Bus Unit

Sources of Errors

CPU Related Errors

- Wrong or incomplete configuration of the bus configuration registers for the FLASH devices
- FLASH area is indicated as cacheable and the data cache is on (TRACE32 can not read/update the FLASH status registers)

Errors Related to Address/Data Bus or the Control Lines

- Defect on the data bus (short circuit, broken line, soldering problem …)
- Defect on the address bus (short circuit, broken line, soldering problem …)
- Swapped address or data lines
- Defect on a control line (Chip Select, Output Enable, Write Enable)

Errors Related to the FLASH

- FLASH (sector) is locked

Errors Related to the TRACE32 FLASH Declaration

- Wrong FLASH family code
- Wrong bus width definition

Errors Related to the File that is Programmed

- The file that is programmed unintentionally overwrites peripheral configuration registers

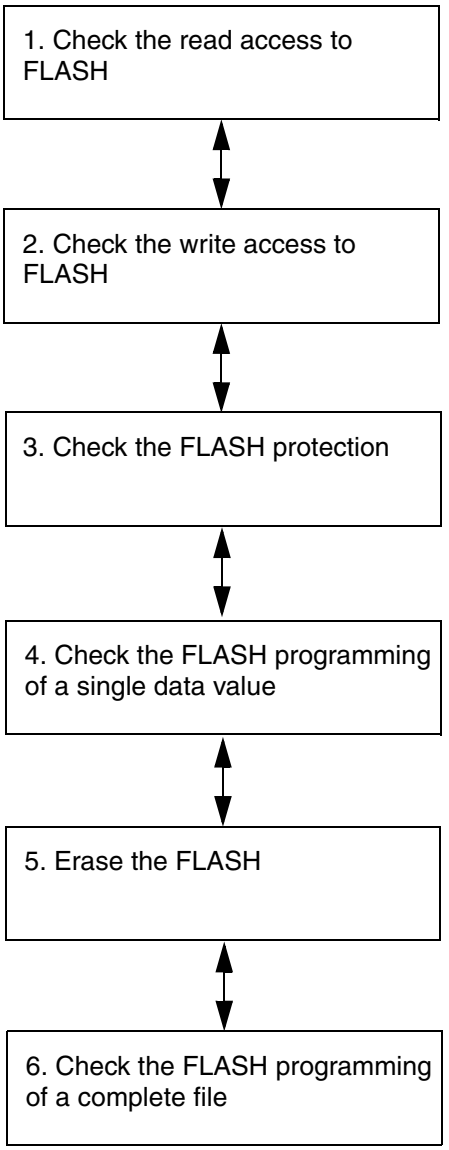
Alignment Error

- Wrong or incorrect alignment used while writing the FLASH data. It is strongly recommended to use a format option for all write accesses that is equivalent to the *<bus_width>* in the FLASH declaration.

```
FLASH.Create … Long

FLASH.Program ALL

Data.LOAD.Elf demo.elf /Long

FLASH.Program OFF
```

General Course of Action in the Case of Problems (Tool-based)

In case of FLASH programming problems the following course of actions is recommended:

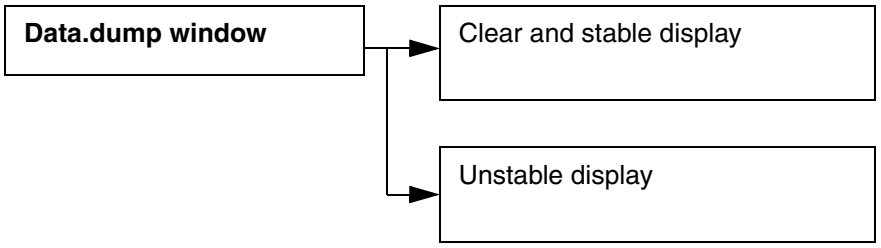


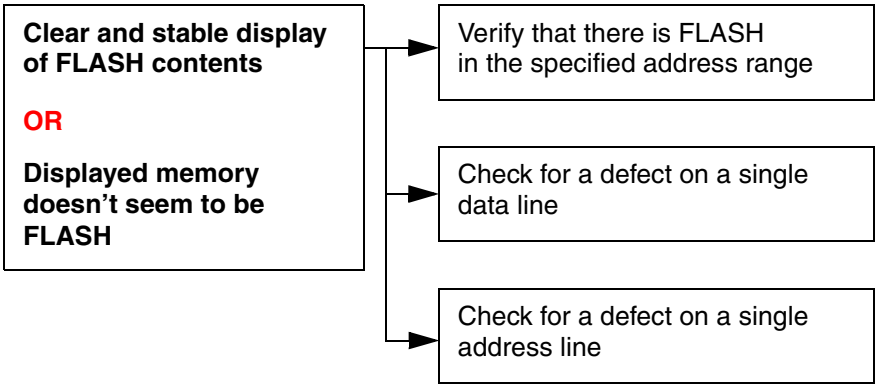
1. Check Read Access to FLASH

In case of FLASH programming problems it is recommended to test, if TRACE32 can read the FLASH contents. This can be checked by using the **Data.dump** command.

Data.dump *<start_address>* *<bus_width>* **/SpotLight**	Display the contents of the FLASH in a Data.dump window. Use the start address of the FLASH devices as *<start_address>*. *<bus_width>* defines the data bus width that is used to connect the FLASH devices to the CPU. If the option **/Spotlight** is used, changed FLASH contents is highlighted.

```
FLASH.Program OFF                          ; Disable FLASH programming

Data.dump 0xA0000000 /Word /SpotLight      ; Example 1

Data.dump 0x40000000 /Long /SpotLight      ; Example 2
```

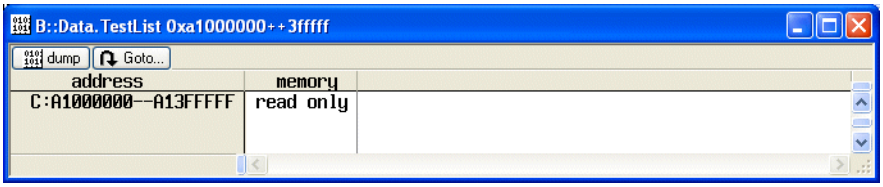



Clear and stable display of FLASH contents
OR
Displayed memory doesn't seem to be FLASH
Verify that there is FLASH in the specified address range
Check for a defect on a single data line
Check for a defect on a single address line

1. Verify that the specified address range is FLASH

Here an example of how to verify if there is FLASH at the specified address range:

```
FLASH.Program OFF                       ; Disable FLASH programming

Data.TestList 0xa1000000++3fffff        ; Check memory type located at
                                        ; the specified address range.
                                        ; It is recommended to specify
                                        ; the complete FLASH address
                                        ; space.
```



This test can have the following results:

ok	RAM
read only	FLASH / ROM
read fail **write fail**	No memory

If the result is **No memory**:

- Please check the bus configuration registers. For details refer to **"Checking the Bus Configuration"** in Tips to Solve NOR FLASH Programming Problems, page 50 (flash_diagnosis.pdf).
- Please check your target for hardware problems accessing the off-chip FLASH.

If the result is **RAM**:

- Please check the bus configuration registers. For details refer to **"Checking the Bus Configuration"** in Tips to Solve NOR FLASH Programming Problems, page 50 (flash_diagnosis.pdf).
- Please check your target design for the correct FLASH address range.

Short description and link to newly introduced command:

Data.TestList *<address_range>*	Test memory type for specified address range The memory test stays active as long as the **Data.TestList** window is displayed.

2. Is there a defect on a single data line?

If there is a defect on a single data line, a single bit is either permanently 0 or permanently 1.

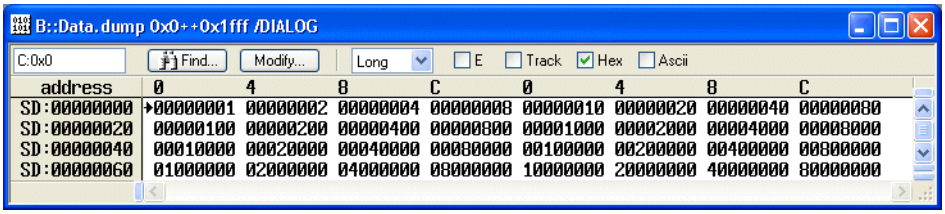
Use the **Data.dump** command to display the FLASH contents.

If the FLASH is empty (0xff) it is easy to check, if a single data line is permanently 0. This behavior points to a short-circuit, a broken data line or soldering problems on your target.

If the FLASH isn't empty, a defective data line might be visible if you look carefully to the contents of the **Data.dump** window.

If a RAM is connected to the same data bus, it is possible to perform the following test for a short-circuit on a data line by using the RAM:

```
; Display the memory contents of the RAM
Data.dump 0x0++0x1fff /Long /SpotLight

; Write a bit shifting pattern to the RAM
Data.Pattern 0x0++0x1fff /LongShift

; Test if the bit shifting pattern is written correctly to RAM
Data.Pattern 0x0++0x1fff /LongShift /Compare
```

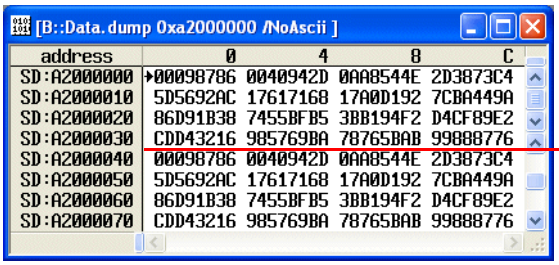


Short description and link to newly introduced command:

Data.Pattern *<range>* [*/<option>*]	Fill memory with selected pattern

3. Is there a defect on a single address line?

A defective address line results in mirrored memory. This effect is only visible in the **Data.dump** window, if the FLASH is not empty.



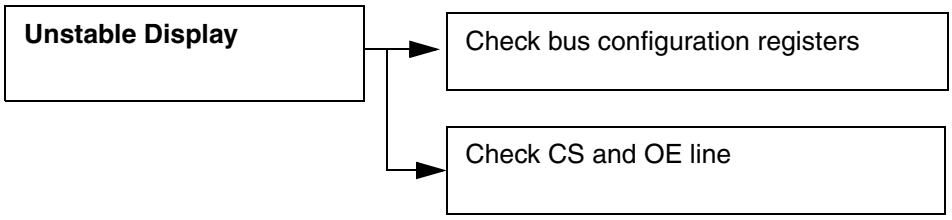
Mirrored address ranges can be found by the following procedure:

```
&start_address=0x40000000

&flash_size=0x3fffff

; display a hex dump
Data.dump &start_address /Long

; store the contents of the first four 32-bit values into PRACTICE
; variables to build a test pattern
&v1=Data.Long(D:&start_address)

&v2=Data.Long(D:&start_address+0x4)

&v3=Data.Long(D:&start_address+0x8)

&v4=Data.Long(D:&start_address+0xc)

; search for the test pattern in the FLASH
Data.Find (&start_address+0x10)++&flash_size %Long &v1 &v2 &v3 &v4


IF FOUND()

      ; if the test pattern is found in the FLASH, lock carefully to
      ; the Data.dump window to check if this is a result of mirrored
      ; address ranges
      Data.dump TRACK.ADDRESS() /Long

ENDDO
```

Please be aware, that one reason for mirrored address ranges can also be a wrong bus configuration for the FLASH device.

Short description and link to newly introduced command:

Data.Find *<range>* **%***<format>* *<test_pattern>*	**FOUND()** returns true if the *<test_pattern>* was found **TRACK.ADDRESS()** returns the address at which the *<test_pattern>* was found



1. Check the bus configuration registers

For details refer to **"Checking the Bus Configuration"** in Tips to Solve NOR FLASH Programming Problems, page 50 (flash_diagnosis.pdf).

If only some bits are toggling, this leads to the assumption, that the timing for the FLASH is not set up correctly. In most cases it is sufficient to increase the number of waits states for the memory access by 1.

2. Check your target hardware

Keep the **Data.dump** window open. This way the FLASH memory contents is read 10 times/s. Please measure the following lines:

Measure if the Chip Select line becomes active, if the FLASH is accessed.

Measure if the Output Enable line becomes active, if the FLASH contents is read.

2. Check Write Access to FLASH

A simple way to check if TRACE32 has write access to the FLASH is to switch the FLASH into ID-mode (Autoselect Mode for AMD/Spansion devices, Read Identifier Mode for Intel devices).

In ID-mode the following information is provided:

- Manufacturer identification
- Device identification
- Sector protection information if available

Intel FLASH Devices

The procedure to switch to ID-mode is described in **"Switch to ID-Mode (Intel)"** in Tips to Solve NOR FLASH Programming Problems, page 52 (flash_diagnosis.pdf).

Alternative test procedures:

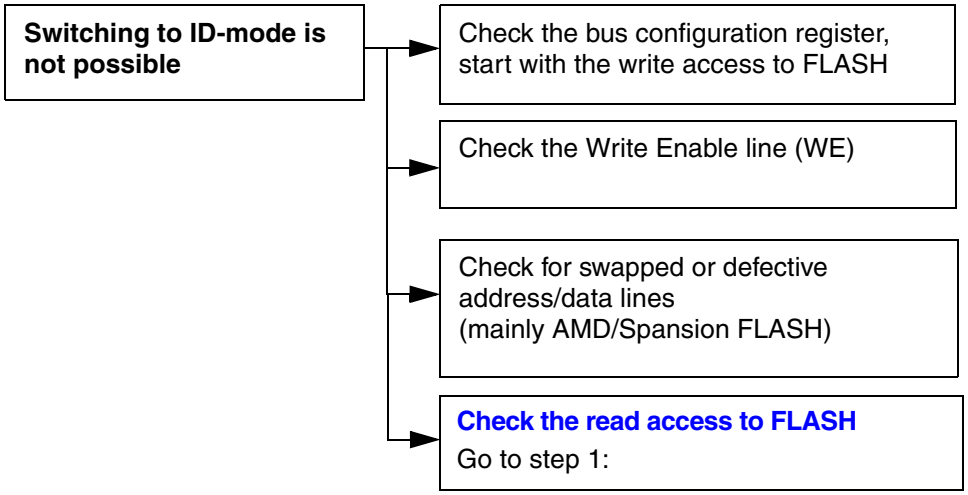
- Try to read the FLASH status and reset the status registers. For details refer to **"Status Analysis (Intel)"** in Tips to Solve NOR FLASH Programming Problems, page 54 (flash_diagnosis.pdf). Reading the FLASH status requires less correctly operating address and data lines, but this check is sufficient to test the write access.

AMD/Spansion FLASH Devices

The procedure to switch to ID-mode is described in **"Switch to ID-Mode (AMD/Spansion)"** in Tips to Solve NOR FLASH Programming Problems, page 56 (flash_diagnosis.pdf).

Alternative test procedures:

- Try to switch to CFI mode for Intel/AMD FLASH devices, if CFI mode is supported by your FLASH device. For details refer to **"Switch to CFI Mode (AMD/Spansion)"** in Tips to Solve NOR FLASH Programming Problems, page 58 (flash_diagnosis.pdf). Switching to CFI mode requires less correctly operating address and data lines, but this check is sufficient to test the write access.



If switching to ID mode is not possible, the write access to FLASH doesn't seem to work.

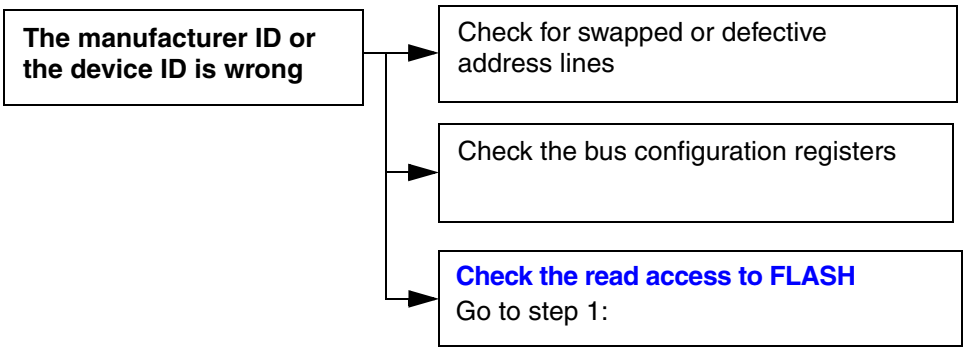
1. **Check the bus configuration register**

   For details refer to **"Checking the Bus Configuration"** in Tips to Solve NOR FLASH Programming Problems, page 50 (flash_diagnosis.pdf).

2. **Check your target hardware**

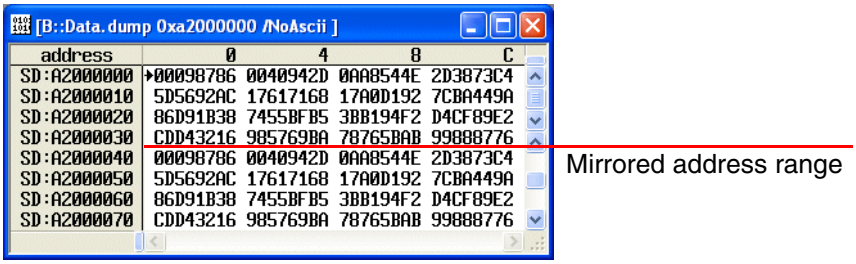
   Please check the WE, OE and CS line on your target hardware.

   Swapped or defect address/data lines can prohibit that the FLASH is switched to ID mode.



1. Is there a defect on an address line?

A defective address line results in mirrored memory. This effect is only visible in the **Data.dump** window, if the FLASH is not empty.



Mirrored address ranges can be found by the following procedure:

```
&start_address=0x40000000

&flash_size=0x3fffff

; display a hex dump
Data.dump &start_address /Long

; store the contents of the first four 32-bit values into PRACTICE
; variables to build a test pattern
&v1=Data.Long(D:&start_address)

&v2=Data.Long(D:&start_address+0x4)

&v3=Data.Long(D:&start_address+0x8)

&v4=Data.Long(D:&start_address+0xc)

; search for the test pattern in the FLASH
Data.Find (&start_address+0x10)++&flash_size %Long &v1 &v2 &v3 &v4


IF FOUND()

      ; if the test pattern is found in the FLASH, lock carefully to
      ; the Data.dump window to check if this is a result of mirrored
      ; address ranges
      Data.dump TRACK.ADDRESS() /Long

ENDDO
```

Please be aware, that one reason for mirrored address ranges can also be a wrong bus configuration for the FLASH device.

For swapped address lines please check your schematics or measure each address line on your target.

Short description and link to newly introduced command:

Data.Find *<range>* **%***<format>* *<test_pattern>*	**FOUND()** returns true if the *<test_pattern>* was found. **TRACK.ADDRESS()** returns the address at which the *<test_pattern>* was found.

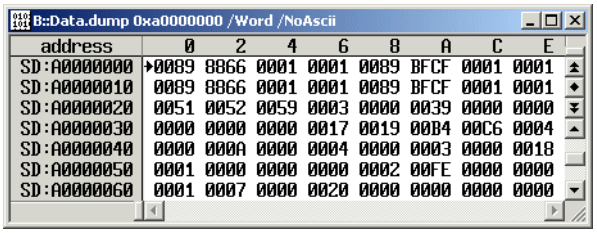
2. Check the bus configuration registers

For details refer to **"Checking the Bus Configuration"** in Tips to Solve NOR FLASH Programming Problems, page 50 (flash_diagnosis.pdf).

3. Check the FLASH Protection

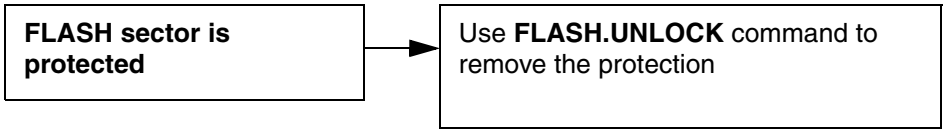
The FLASH protection can be checked, if the FLASH is in ID-Mode. For a detailed description on how to switch the FLASH to ID-mode, refer to **"2. Check Write Access to FLASH"** in Tips to Solve NOR FLASH Programming Problems, page 18 (flash_diagnosis.pdf).

E.g ID information for Intel 28F128W18T:



1st word: Manufacturer ID	0x89 (Intel)
2nd word: Device ID	0x8866 (28F128W18T as bottom boot block device)
3rd word: Sector protection	0x1 (sector locked)

Sector Protection Information	
0x0	Sector is not locked
0x1	Sector is locked
any other value	Please refer to the data sheet for your FLASH device



FLASH.UNLOCK *<unit>* | *<range>* | **ALL** Unlock FLASH

```
FLASH.UNLOCK ALL

FLASH.UNLOCK 0xa0000000++0xfffff
```

Some FLASH devices are locked after power-up. Refer to the data sheet for your FLASH device for details. If your FLASH device is locked after power-up, the following command sequence is recommended:

```
…                                   ; Configure your CPU

SYStem.Up                           ; Establish the communication
                                    ; between TRACE32 and the CPU

…                                   ; Configure the bus configuration
                                    ; registers

FLASH.Create …                      ; Declare your FLASH

FLASH.UNLOCK ALL                    ; Remove the FLASH protection
```

FLASH devices behave differently when removing the protection.

- Each FLASH sector has to be unlocked separately. In this case the command **FLASH.UNLOCK ALL** is recommended if the complete FLASH device should be unprotected.

  If the block address (BA) is required to unlock the FLASH, each sector has to be unlocked separately.

Mode	Command	Bus Cycles	First Bus Cycle			Second Bus Cycle		
			Oper	Addr[1]	Data[2]	Oper	Addr[1]	Data[2]
Block Locking/ Unlocking	Lock Block	2	Write	BA	0x60	Write	BA	0x01
	Unlock Block	2	Write	BA	0x60	Write	BA	0xD0
	Lock-down Block	2	Write	BA	0x60	Write	BA	0x2F

- Each unlock takes effect on the complete FLASH device. In this case the command **FLASH.UNLOCK** *<range_of_first_sector>* is more effective, since it performs much faster.

  If the block address (BA) isn't required to unlock the FLASH, all sectors are unlocked in one step.

Command	Scaleable or Basic Command Set(14)	Bus Cycles Req'd	Notes	First Bus Cycle			Second Bus Cycle		
				Oper(1)	Addr(2)	Data(3,4)	Oper(1)	Addr(2)	Data(3,4)
Set Block Lock-Bit	SCS	2	11	Write	X	60H	Write	BA	01H
Clear Block Lock-Bits	SCS	2	12	Write	X	60H	Write	X	D0H

For details refer to the data sheet of your FLASH device.

Please use the command **FLASH.LOCK** if you want to enable the sector protection after the FLASH is programmed.

FLASH.LOCK *<unit>* | *<range>* | **ALL** Lock FLASH

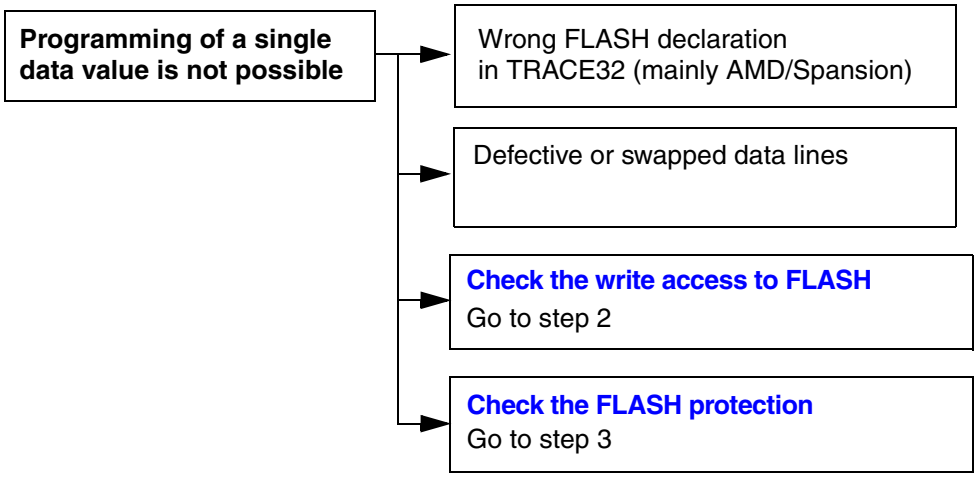
Locking the FLASH is always performed sector-wise.

4. Check the Programming of a Single Data Value

Here a short example to program a single data value to FLASH.

```
; Search for empty FLASH cell
Data.Find 0x0++0xfffff %Long 0xffffffff

; Display empty cell
Data.dump TRACK.ADDRESS() /SpotLight

; Enable FLASH programming
FLASH.Program ALL

; Write new data value to empty FLASH cell
Data.Set TRACK.ADDRESS() %Long 0x01234567

; Disable FLASH programming
FLASH.Program OFF
```

Data.Find *<range>* **%***<format>* *<test_pattern>*	**FOUND()** returns true if the *<test_pattern>* was found. **TRACK.ADDRESS()** returns the address at which the *<test_pattern>* was found.


Programming of a single data value is not possible
Wrong FLASH declaration in TRACE32 (mainly AMD/Spansion)
Defective or swapped data lines
Check the write access to FLASH
Go to step 2
Check the FLASH protection
Go to step 3

1. Log all read/write accesses that are performed by TRACE32 to write a single data value

If you succeeded to switch the FLASH to ID mode by using the **Data.Set** commands described in **"Switch to ID-Mode (AMD/Spansion)"** in Tips to Solve NOR FLASH Programming Problems, page 56 (flash_diagnosis.pdf), perform the following test sequence:

```
WinClear

SYStem.LOG.List

FLASH.Program ALL

; Search for empty FLASH cell
Data.Find 0x++0xfffff %Long 0xffffffff

; Write single data value for empty FLASH cell
Data.Set TRACK.ADDRESS() %Long 0x01234567

FLASH.Program OFF
```

Data.Find *<range>* **%***<format>* *<test_pattern>*	**FOUND()** returns true if the *<test_pattern>* was found. **TRACK.ADDRESS()** returns the address at which the *<test_pattern>* was found.

An introduction on how to analyze the logging of the TRACE32 read/write accesses is given in **"Interpretation of the SYStem.LOG.List"** in Tips to Solve NOR FLASH Programming Problems, page 62 (flash_diagnosis.pdf).

Compare the command register addresses used by TRACE32 with the command register addresses you used to switch to ID-mode. Differences in the command register addresses lead to the assumption that a wrong *<family_code>* or a wrong *<bus_width>* is used in the TRACE32 FLASH declaration.

Differences in the endianness used for the data values lead to the assumption that the FLASH is connected to your PowerPC CPU in true little endian mode. In this case only target-controlled FLASH programming can be used.

2. Is there a defect on a single data line?

To test for a defective data line, write a test pattern to the FLASH.

```
WinClear

FLASH.Program ALL

; Search for an empty FLASH cell
Data.Find 0x++0xfffff %Long 0xffffffff

; Write test pattern 1
Data.Set TRACK.ADDRESS() %Long 0x55555555

; Check the result
Data.Print TRACK.ADDRESS() %Long

; Search for the next empty FLASH cell
Data.Find

; Write test pattern 2
Data.Set TRACK.ADDRESS() %Long 0xaaaaaaaa

; Check the result
Data.Print TRACK.ADDRESS() %Long

FLASH.Program OFF
```

In case of an error, check your target hardware for a defective data line.

Data.Find *<range>* **%***<format>* *<test_pattern>*	Repeat search by using **Data.Find** without any parameters. **FOUND()** returns true if the *<test_pattern>* was found. **TRACK.ADDRESS()** returns the address at which the *<test_pattern>* was found.

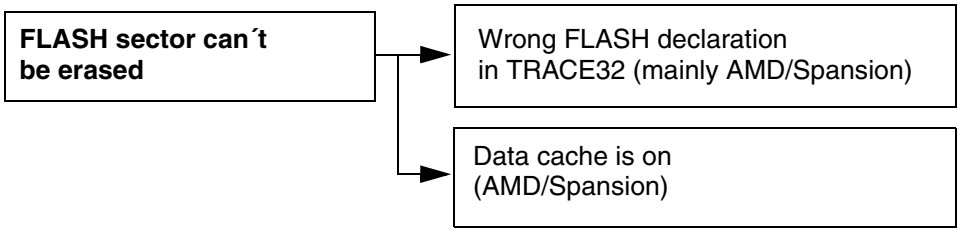
5. Erase the FLASH

Before you try to erase the complete FLASH it is recommended to test if a single FLASH sector can be erased.

Check the protection for a selected FLASH sector as described in **"3. Check the FLASH Protection"** in Tips to Solve NOR FLASH Programming Problems, page 23 (flash_diagnosis.pdf). Use the following command to erase this single sector.

FLASH.Erase *<range>* Erase the contents of the specified address range

```
FLASH.Erase 0x0--0xfffff
```



1. **Read the Status Registers for Intel FLASH devices.**

   Refer to **"Status Analysis (Intel)"** in Tips to Solve NOR FLASH Programming Problems, page 54 (flash_diagnosis.pdf) for details.

2. **Use the SYStem.LOG.List window to check the correct FLASH declaration.**

3. **Use the SYStem.LOG.List window to check if the data cache is on.**

Both test are described in **"Interpretation of the SYStem.LOG.List"**, page 62.

Please be aware that TRACE32 doesn't detect any problem in erasing a FLASH sector, if the FLASH sector is already empty.

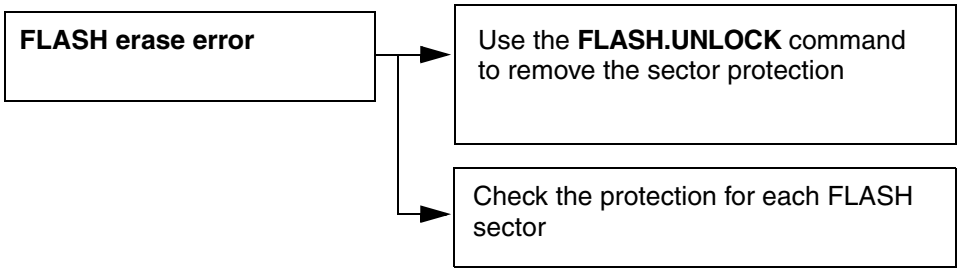
If TRACE32 is able to erase a single sector, you can be sure that TRACE32 can erase the complete FLASH.

Errors that occur now when you try to erase several sectors or the complete FLASH can be isolated as a sector protection problem.

Please be aware that a bulk erase/chip erase command is used to erase the FLASH device if

- bulk erase/chip erase is supported by the FLASH device
- and if one of the following commands is used to erase the FLASH
  FLASH.Erase ALL
  FLASH.Erase *<unit_number>*
  FLASH.Erase *<range>*, when the *<range>* covers the complete address range declared for a *<unit_number>*

For details refer also to **"Unintentional Erasing of the Complete FLASH Device"** in Tips to Solve NOR FLASH Programming Problems, page 47 (flash_diagnosis.pdf).



Refer to **"3. Check the FLASH Protection"** in Tips to Solve NOR FLASH Programming Problems, page 23 (flash_diagnosis.pdf) in both cases.

Unintended Overwrite of Peripheral Configuration Registers

At this point it makes only sense to check if the file you want to program into FLASH unintentionally overwrites peripheral configuration registers.

Here an example, that shows you how to check if your file unintentionally overwrites important configuration registers.

```
Data.LOAD.Elf demo.elf /VM            ; Download the file to the virtual
                                      ; memory

sYmbol.List.Map                       ; Display the load order and the
                                      ; address ranges to which the file
                                      ; was downloaded
```

Please make sure, that the file is solely downloaded to the address ranges of the FLASH.

If your file overwrites peripheral configuration registers, please check your map file or continue as follows:

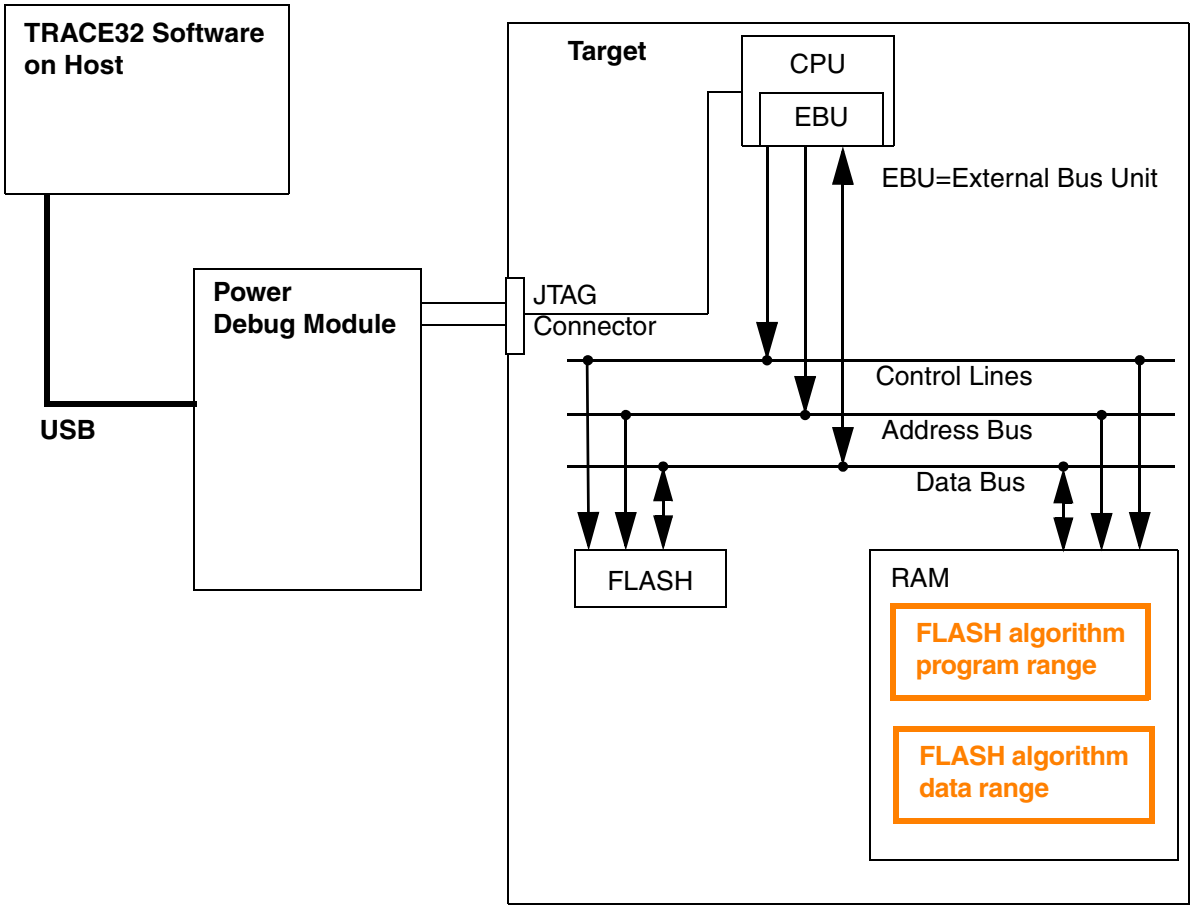
```
FLASH.Program ALL                              ; Enable FLASH programming

Data.LOAD.Elf demo.elf 0xc00000++0x1ffff       ; Load file to FLASH address
                                               ; ranges

FLASH.Program OFF                              ; Disable FLASH programming
```

Short description and link to newly introduced command:

sYmbol.List.MAP	Display to which address ranges in the memory a file was downloaded.

Target-controlled Programming

If target-controlled FLASH programming is used, the FLASH algorithm is downloaded to the target RAM and is executed there.



The fact that the FLASH algorithm is downloaded to the target RAM and is executed there adds new requirements:

1. Read and write access to the target RAM by TRACE32
2. Execution of the downloaded FLASH algorithm on the target RAM
3. Setting of software breakpoints to target RAM

Sources of Errors

CPU Related Errors

- Wrong or incomplete configuration of the bus configuration registers for the RAM

Errors Related to the RAM

- Wrong or incomplete SDRAM initialization
- Missing ECC initialization for the RAM

Watchdog Problems

- The FLASH programming algorithm doesn't serve any active watchdog. An active watchdog results in an abort of the FLASH programming algorithm.

Errors Related to the File that is Programmed

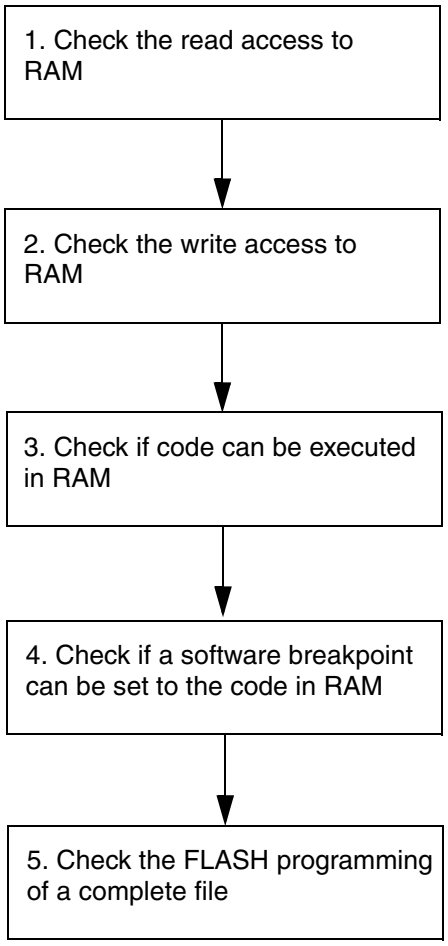
- The file that is programmed unintentionally overwrites the FLASH algorithm

Address Range for FLASH Algorithm is to Small

- The data of the FLASH algorithm unintentionally overwrite the FLASH algorithm

General Course of Action in the Case of Problems (Target-controlled)

After you defined both, the code range for the FLASH algorithm and the data range for the FLASH algorithm, the following course of action is recommended:



1. Check Read Access to RAM

To check if TRACE32 can read the RAM use the **Data.dump** command. It is recommended to perform all test either on the program or on the data range defined for the FLASH algorithm.

Data.dump *<start_address>* *<bus_width>* **/SpotLight**

```
Data.dump 0x1000 /Long /SpotLight
```

Clear and Stable Display

Clear and stable display of RAM contents

OR

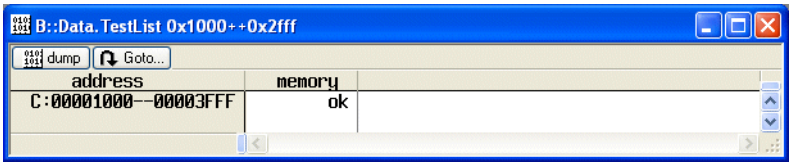
Displayed memory doesn't seem to be RAM

→ Verify that there is RAM in the specified address range

Verify that the address range required for the FLASH algorithm and its data is RAM

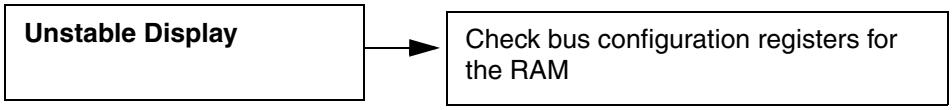
Data.TestList *<address_range>* Test memory type for specified address range

```
Data.TestList 0x1000++0x2fff
```



If the result is **read fail** or **write fail**, please check the bus configuration registers for the RAM and the RAM initialization.

Since TRACE32 tool-based FLASH programming runs faultless, you can almost be sure that the address bus, the data bus and the control lines are working correctly.


Unstable Display
Check bus configuration registers for the RAM

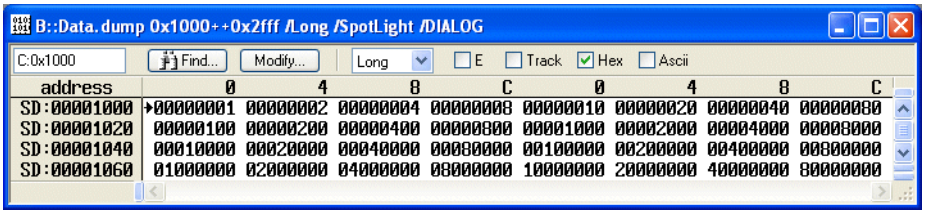
2. Check Write Access to RAM

To test the write access to RAM, use the following command:

Data.Pattern *<range>* [*/<option>*] Fill memory with selected pattern

Example:

```
; Display the memory contents of the RAM
Data.dump 0x1000 /Long /SpotLight

; Write a bit shifting pattern to the RAM
Data.Pattern 0x1000++0x2fff /LongShift

; Test if the bit shifting pattern is written correctly to RAM
Data.Pattern 0x1000++0x2fff /LongShift /Compare
```

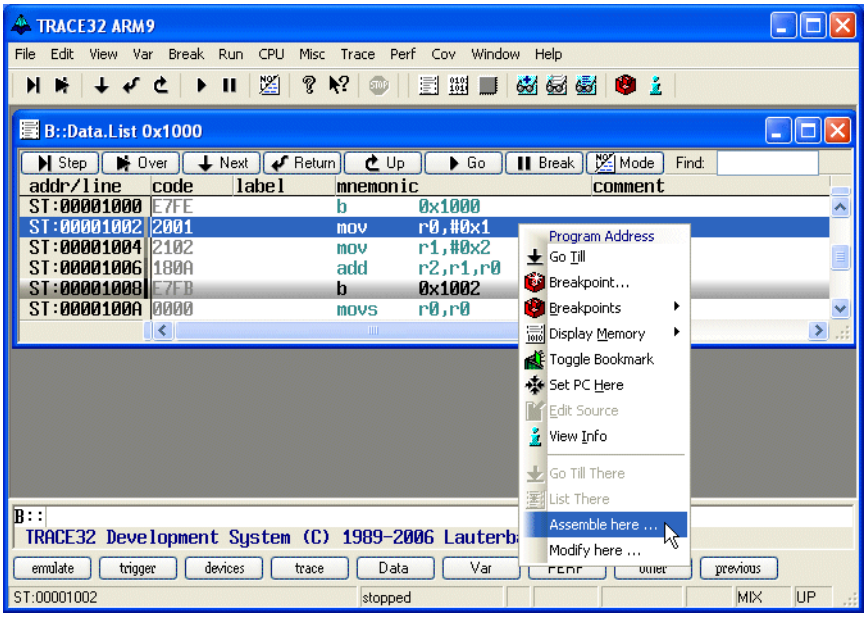


If TRACE32 can not write to RAM, check the bus configuration registers.

3. Check if Code can be Executed in RAM

To check, if code can be executed use the following command to write a small test program into RAM.

Data.Assemble *<address> <mnemonic>* Assemble the mnemonic to RAM



Example:

```
Data.Assemble 0x1000 b 0x1000          ; b 0x1000 prevents that test
                                       ; program is unintentionally
                                       ; re-entered after a TRAP or
                                       ; interrupt

Data.Assemble 0x1002 mov r0,#1         ; Start of test program

Data.Assemble 0x1004 mov r1,#2

Data.Assemble 0x1006 add r2,r1,r0

Data.Assemble 0x1008 b 0x1002
```

Set the Program Counter/Instruction Pointer to the start address of your test program and perform a few single steps.

```
Register.Set pc 0x1002

Step

Step
```

Set the Program Counter/Instruction Pointer to the start address of your test program. Start the program execution, let it run for a few seconds and stop it again. With this test you can check if an active watchdog will cause problems for the target-controlled FLASH algorithm.

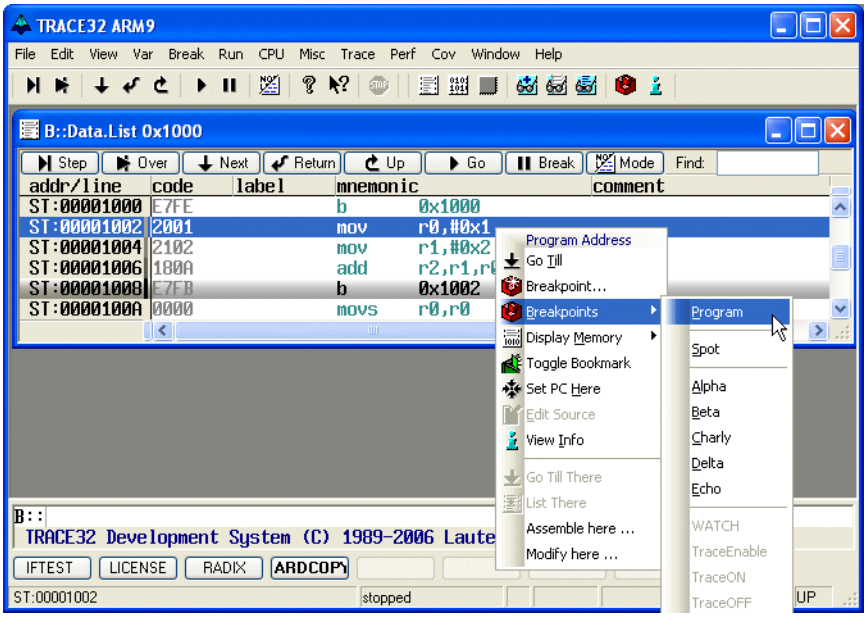
```
Register.Set pc 0x1002

Go

; WAIT 10.s                          can be used only in scripts

Break
```

Please check carefully, if the program execution is stopped within your test program. If this is not case this leads to the assumption, that the watchdog on your target needs to be deactivated.

4. Check if a Software Breakpoint can be Set

Check if a software breakpoint can be set to your small test program.

Break.Set *<address>* **/Program /SOFT**	Set a software breakpoint to an instruction



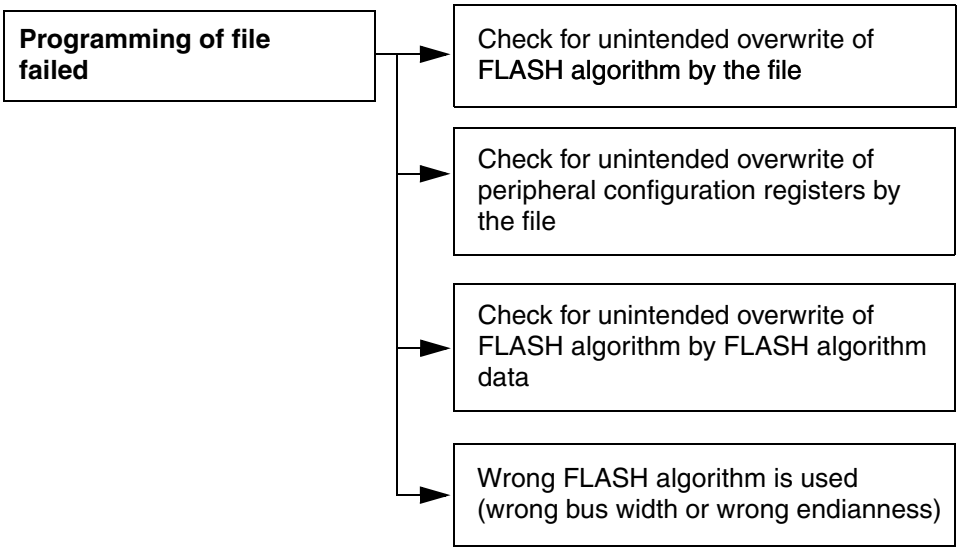
Example:

```
Break.Set 0x1004 /Program /SOFT

Go
```

5. Check the FLASH Programming of a Complete File

```
FLASH.Create … TARGET …                ; Declare your FLASH

FLASH.TARGET …

FLASH.Erase                            ; Erase the FLASH contents

FLASH.Program ALL                      ; Enable FLASH programming

Data.LOAD.Elf demo.elf                 ; Load file to FLASH address ranges

FLASH.Program OFF                      ; Disable FLASH programming

Data.LOAD.Elf demo.elf /Compare
```

Programming of File Failed



1. Does the file that is programmed to the FLASH unintentionally overwrite important configuration registers or parts of the FLASH programming algorithm?

Here an example, that shows you how to check if your file unintentionally overwrites important configuration registers or parts of the FLASH programming algorithm.

```
Data.LOAD.Elf demo.elf /VM        ; Download the file to the virtual
                                  ; memory

sYmbol.List.Map                   ; Display the load order and the
                                  ; address ranges to which the file
                                  ; was downloaded
```

Please make sure, that the file is solely downloaded to the address ranges of the FLASH.

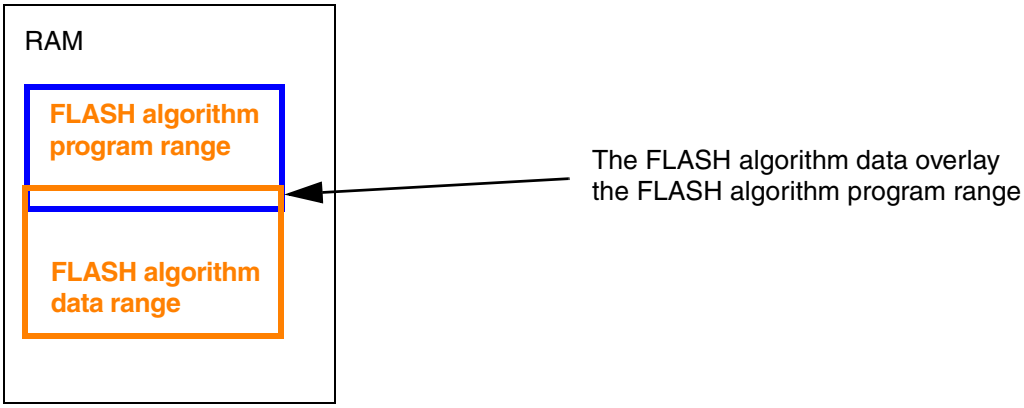
If you also want to download some program parts to RAM, please proceed as follows:

```
FLASH.Program ALL                                 ; Enable FLASH
                                                  ; programming

Data.LOAD.Elf demo.elf 0xc00000++0x1ffff          ; Load file to FLASH
                                                  ; address ranges

FLASH.Program OFF                                 ; Disable FLASH
                                                  ; programming

Data.LOAD.Elf demo.elf 0xe00000++0x7ff            ; Load file to the RAM
                                                  ; address ranges
```

Short description and link to newly introduced command**:**

sYmbol.List.MAP	Display to which address ranges in memory a file was downloaded.

2. Do the data of the FLASH programming algorithm unintentionally overwrite the FLASH algorithm?



Please make sure, that you reserve enough target RAM for the FLASH programming algorithm. The required size for the FLASH algorithm is calculated as follows:

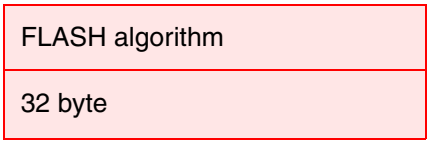


Figure: Memory mapping for the *<code_range>*

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

The required size for the data of the FLASH algorithm is calculated as follows:

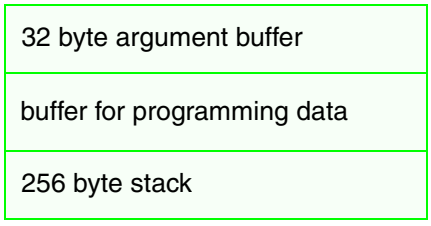


Figure: Memory mapping for the *<data_range>*

```
<buffer_size> =
size_of(<data_range>) - 32 byte argument buffer - 256 byte stack
```

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

Details on the memory mapping for the FLASH algorithm can be found in the **"Onchip/NOR FLASH Programming User's Guide"** (norflash.pdf).

3. Check if the Correct FLASH Algorithm is used.

To check if the correct FLASH algorithm is used, use the following commands:

```
FLASH.Create … TARGET …          ; FLASH declaration

FLASH.TARGET  …

…

FLASH.Program ALL

Data.List
```

The command **FLASH.Program ALL** loads the FLASH algorithm to the target RAM and set the program counter/instruction pointer correctly. Please look carefully to the code displayed in the **Data.List** window. If the code displayed there doesn't seem to be the code at the beginning of a function, check if you use the FLASH algorithm from the correct directory.

Ready-to-run binary files for target-controlled FLASH programming are available for the most common architectures in the TRACE32 demo folder under:
`~~/demo/<architecture>/flash`

The FLASH algorithms are organized by *<bus_width>* and by *<endianness>.*

*<bus_width>*_**be** stands for FLASH support for big endian mode.
*<bus_width>*_**le** stands for FLASH support for little endian mode.

If your processor architecture has a preferred endianness, this *<endianness>* is left out and only the *<bus_width>* is listed.

Example: the preferred *<endianness>* for the ARM architecture is little endian mode.

```
~~/demo/arm/flash/long
   ; algorithms for little endian
~~/demo/arm/flash/long_be
   ; algorithms for big endian
```

If your FLASH is connected in true little endian mode to your PowerPC CPU, please use the FLASH algorithm from the directory
`~~/demo/powerpc/flash/<bus_width>_tle`

Unintentional Erasing of the Complete FLASH Device

Please be aware that a bulk erase/chip erase command is used to erase the FLASH device if

- bulk erase/chip erase is supported by the FLASH device
- and if one of the following commands is used to erase the FLASH
  FLASH.Erase ALL
  FLASH.Erase *<unit_number>*
  FLASH.Erase *<range>*, when the *<range>* covers the complete address range declared for a *<unit_number>*

Here an example:

FLASH typ	S29CD016, chip erase possible
FLASH start address	0xa2000000
FLASH sectors	8 x 8 KByte (0x2000) 30 (0x1E) x 64 KByte (0x10000) 8 x 8 KByte (0x2000)

TRACE32 FLASH declaration:

```
; FLASH declarations for the top boot sectors only
FLASH.Create 1. 0xa2000000--0xa200ffff 0x2000 AM29BDD Long

; FLASH.Create 1. 0xa2010000--0xa21effff 0x10000 AM29BDD Long

; FLASH.Create 1. 0xa21f0000--0xa21fffff 0x2000 AM29BDD Long

FLASH.List
```

The following command will erase the complete FLASH **(0xa2000000--0xa21fffff)**:

```
FLASH.Erase ALL

FLASH.Erase 1.

FLASH.Erase 0xa2000000--0xa200ffff
```

Multiple FLASH Devices on one Target

TRACE32 can handle only a single FLASH algorithm for target-controlled FLASH programming. This applies for both on-chip FLASH and off-chip FLASH devices.

The FLASH algorithm defined with the last **FLASH.TARGET** command is always the valid one.

If you have multiple FLASH devices on your target they require different FLASH algorithms, proceed as follows:

```
FLASH.Create 1. … TARGET …          ; Declaration for first FLASH device

…

FLASH.Create 2. … TARGET …          ; Declaration for second FLASH device

…

FLASH.TARGET … am29lv100.bin        ; Declare target algorithm for FLASH
                                    ; device with <unit_number> 1.

FLASH.Erase 1.

FLASH.Program 1.

Data.Load <file1>

FLASH.Program OFF

FLASH.TARGET … i28f200b.bin         ; Declare target algorithm for FLASH
                                    ; device with <unit_number> 2.

FLASH.Erase 2.

FLASH.Program 2.

Data.Load <file2>

FLASH.Program OFF
```

FAQ

Please refer to https://support.lauterbach.com/kb.

Appendix A

Conversion of the Tool-based to Target-controlled FLASH Programming

```
&FLASH_PROGRAMMING_METHOD="TRACE32 tool-based"
; &FLASH_PROGRAMMING_METHOD="target-controlled"

FLASH.RESet
IF "&FLASH_PROGRAMMING_METHOD"=="TRACE32 tool-based"
(
; FLASH.Create <unit_number> <physical_range> <sector_size> <family_code> <bus_witdth>
      FLASH.Create 1. 0x40000000++0x3fffff 0x20000 I28F200B Long
)
IF "&FLASH_PROGRAMMING_METHOD"=="target-controlled"
(
      ; FLASH.Create <unit_number> <physical_range> <sector_size> TARGET <bus_witdth>
      FLASH.Create 1. 0x40000000++0x3fffff 0x20000 TARGET Long

      ; FLASH.Target <code_address> <data_address> <buffer_size> <flash_algorithm>
      FLASH.TARGET 0x0 0x1000 0x1000 ~~/demo/arm/flash/long_be/i28f200b.bin
)
```

Conversion from TRACE32 tool-based FLASH declaration to target-controlled FLASH declaration:

1. Replace the `<family_code>` by the keyword **TARGET** when using the **FLASH.Create** command.
2. Use the **FLASH.TARGET** command to specify the *<code_address>, <data_address>, <buffer_size>* and the *<flash_algorithm>*. Please remember to use the FLASH algorithm from the directory with the adequate *<bus_width>* and the correct *<endianness>*. The name of *<flash_algorithm>* matches with the *<family_code>*.

After the declarations for the FLASH devices are done, all other commands of the command group FLASH apply to both programming methods.

For details, refer to the **"Onchip/NOR FLASH Programming User's Guide"** (norflash.pdf).

Checking the Bus Configuration

Correct settings in the bus configuration registers are key for the FLASH programming. The following settings in the bus configuration are of importance:

- The base address for the FLASH devices
- The size of the FLASH devices
- The bus size that is used to access the FLASH devices

  If several FLASH devices are used in parallel to realize the required *<bus_width>* between the processor and the FLASH memory, make sure that the *<bus_width>* is configured correctly
- The timing (number of wait states for the access to the FLASH devices)
- The write access has to be enabled for the FLASH devices

Use the **PER.view** command to check the settings in the bus configuration registers:

Main Difference between Intel and AMD/Spansion FLASH Devices

Intel FLASH Devices

Commands are executed on Intel FLASH devices by writing a specified data value to any FLASH address.

Command	Scaleable or Basic Command Set(14)	Bus Cycles Req'd	Notes	First Bus Cycle			Second Bus Cycle		
				Oper(1)	Addr(2)	Data(3,4)	Oper(1)	Addr(2)	Data(3,4)
Read Array	SCS/BCS	1		Write	X	FFH			
Read Identifier Codes	SCS/BCS	≥2	5	Write	X	90H	Read	IA	ID
Read Query	SCS	≥ 2		Write	X	98H	Read	QA	QD
Read Status Register	SCS/BCS	2		Write	X	70H	Read	X	SRD
Clear Status Register	SCS/BCS	1		Write	X	50H			

```
Data.Set 0xa0000000 %Long 0x90909090    ; Switch the FLASH device to ID
                                        ; mode

Data.Set 0xa0000000 %Long 0x70707070    ; Read FLASH status registers

Data.Set 0xa0000000 %Long 0x50505050    ; Reset FLASH status register
```

AMD/Spansion FLASH Devices

Commands are executed on AMD/Spansion FLASH devices by writing a command sequence to the command registers of the FLASH.

Command Sequence Read/Reset	Bus Write Cycles Req'd	First Bus Write Cycle		Second Bus Write Cycle		Third Bus Write Cycle		Fourth Bus Read/Write Cycle		Fifth Bus Write Cycle		Sixth Bus Write Cycle	
		Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read/Reset	1	XXXH	F0H										
Read/Reset	4	555H	AAH	AAAH	55H	555H	F0H	RA	RD				
Autoselect Manufacturer ID	4	555H	AAH	AAAH	55H	555H	90H	00H	01H				
Autoselect Device ID (Top Boot Device)	4	555H	AAH	AAAH	55H	555H	90H	01H	B0H				

Example: Switch an AM29BDD with *<bus_width>* long to ID mode.

```
Data.Set 0xa0001554 %Long 0xaa

Data.Set 0xa000aa8 %Long 0x55

Data.Set 0xa0001554 %Long 0x90
```

Switch to ID-Mode (Intel)

INTEL FLASH devices are switched to ID mode by writing 0x90 to any address of the FLASH address range.

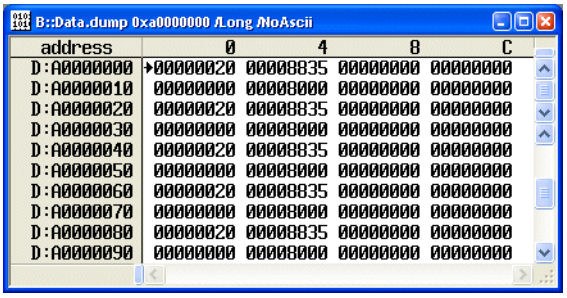
Data.Set *<start_address_of_flash> <bus_width>* **0x90**

INTEL FLASH devices are switched back to Read Mode by writing 0xff to any address of the FLASH address range.

Data.Set *<start_address_of_flash> <bus_width>* **0xff**

Example for FLASH device M58BW016BB from ST Microelectronics. The FLASH device is connected to the CPU via a 32 bit data bus.

```
Data.Dump 0xa0000000 /Long /NoAscii      ; Display the ID mode
                                         ; information

Data.Set 0xa0000000 %Long 0x90           ; Switch the FLASH device to ID
                                         ; mode
```



ID information:

Manufacturer ID	0x20 (ST Microelectronics)
Device ID	0x8835 (FLASH device M58BW016BB as bottom boot block device)

The ID information is either displayed:

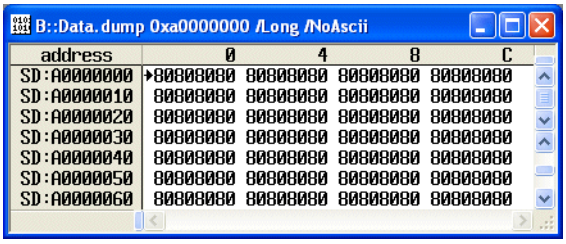
- always at the beginning of each FLASH sector
- repetitive in each FLASH sector (see example above), the repetition rate depends on the FLASH device

```
Data.Set 0xa0000000 %Long 0xff        ; Switch the FLASH back to Read
                                      ; Mode
```

Intel FLASH devices can be switched to **Read Status Register** mode by writing 0x70 to any address of the FLASH.

Data.Set *<start_address_of_flash> <bus_width>* **0x70**

```
Data.Set 0xa0000000 %Long 0x70

Data.dump 0xa0000000 /Long
```



Any other value then 0x80 indicates an error. For a description of the error codes refer to the data sheet of your FLASH device.

Status Register Definitions

WSMS	ESS	ECLBS	PSLBS	VPENS	R	DPS	R
bit 7	bit 6	bit 5	bit 4	bit 3	bit2	bit 1	bit 0

High Z When Busy?	Status Register Bits	Notes
No	SR.7 = WRITE STATE MACHINE STATUS 1 = Ready 0 = Busy	Check STS or SR.7 to determine block erase, program, or lock-bit configuration completion. SR.6–SR.0 are not driven while SR.7 = "0."
Yes	SR.6 = ERASE SUSPEND STATUS 1 = Block Erase Suspended 0 = Block Erase in Progress/Completed	If both SR.5 and SR.4 are "1"s after a block erase or lock-bit configuration attempt, an improper command sequence was entered.
Yes	SR.5 = ERASE AND CLEAR LOCK-BITS STATUS 1 = Error in Block Erasure or Clear Lock-Bits 0 = Successful Block Erase or Clear Lock-Bits	SR.3 does not provide a continuous programming voltage level indication. The WSM interrogates and indicates the programming voltage level only after Block Erase, Program, Set Block Lock-Bit, or Clear Block Lock-Bits command sequences.
Yes	SR.4 = PROGRAM AND SET LOCK-BIT STATUS 1 = Error in Setting Lock-Bit 0 = Successful Set Block Lock Bit	SR.1 does not provide a continuous indication of block lock-bit values. The WSM interrogates the block lock-bits only after Block Erase, Program, or Lock-Bit configuration command sequences. It informs the system, depending on the attempted operation, if the block lock-bit is set. Read the block lock configuration codes using the Read Identifier Codes command to determine block lock-bit status.
Yes	SR.3 = PROGRAMMING VOLTAGE STATUS 1 = Low Programming Voltage Detected, Operation Aborted 0 = Programming Voltage OK	SR.0 is reserved for future use and should be masked when polling the status register.
Yes	SR.2 = PROGRAM SUSPEND STATUS 1 = Program suspended 0 = Program in progress/completed	
Yes	SR.1 = DEVICE PROTECT STATUS 1 = Block Lock-Bit Detected, Operation Abort 0 = Unlock	
Yes	SR.0 = RESERVED FOR FUTURE ENHANCEMENTS	

Writing 0x50 to any address clears the status registers.

Writing 0xff to any address switches the FLASH device back to Read Mode.

```
Data.dump 0xa0000000 /Long /SpotLight

Data.Set 0xa0000000 %Long 0x50

Data.Set 0xa0000000 %Long 0xff
```

Switch to ID-Mode (AMD/Spansion)

Switching to ID-mode is done by writing a specific command sequence to the command registers of the FLASH device.

To switch an AMD/Spansion FLASH device via TRACE32 to ID-mode, the following 3 commands have to be used:

Data.Set *<start_address_of_flash>+<addra> <bus_width> <value1>*

Data.Set *<start_address_of_flash>+<addrb> <bus_width> <value2>*

Data.Set *<start_address_of_flash>+<addra> <bus_width> <value3>*

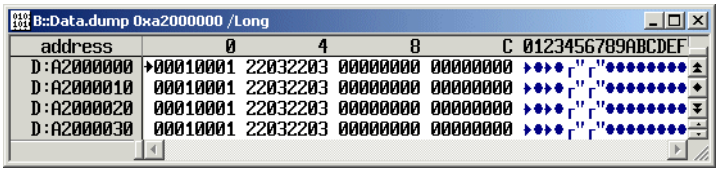
<addra>, *<addrb>* and *<valuex>* depend on the FLASH device type and can be referenced from **"Appendix B"**, page 68.

Example for 2 AM29BL162C (16-bit mode) in parallel to realize a 32-bit bus.

```
Data.Set <start_address_of_flash>+0x1554 %Long 0x00aa00aa

Data.Set <start_address_of_flash>+0xaa8 %Long 0x00550055

Data.Set <start_address_of_flash>+0x1554 %Long 0x00900090
```

ID information:

Manufacturer ID	0x01 (AMD)
Device ID	0x2203 (FLASH device AM29BL162C as bottom boot block device)
Sector protection information	0 (Sector unlocked)



The ID information is either displayed:

- always at the beginning of each FLASH sector
- repetitive in each FLASH sector (see example above), the repetition rate depends on the FLASH device

Data.Set <*start_address_of_flash*> <*bus_width*> **0xf0** Switch the FLASH back to Read Mode

```
Data.Set <start_address_of_flash> %Long 0xf0f0f0f0
```

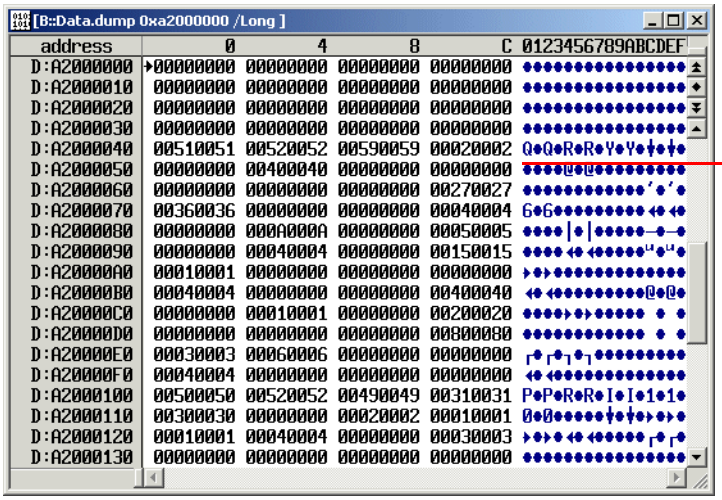
If switching to ID-mode for an AMD/Spansion device can be done and if the manufacturer-id and the device-id is displayed correctly, you can almost be sure that the data bus is working properly.

Switch to CFI Mode (AMD/Spansion)

To switch an AMD/Spansion FLASH device via TRACE32 to CFI-mode, the following command has to be used:

Data.Set *<start_address_of_flash>+<addra> <bus_width>* 0x98

```
Data.dump <start_address_of_flash> /Long /SpotLight

Data.Set <start_address_of_flash>+0x1554 %Long 0x98989898
```



The letters Q R Y indicate, that the FLASH is in CFI mode

In the example above 2 16 bit FLASH devices are used in parallel to realize a 32 bit bus. For this reason

```
0x00510051                              Q

0x00520052                              R

0x00590059                              Y
```

is displayed at address 0xa0000040.

Please be aware, that the memory display of **QRY** will vary depending on:

- the data width of the FLASH device
- the number of parallel FLASH devices

The address 0xa0000040 is also just an example.

The operations on AMD/Spansion FLASH devices (e.g. erasing, programming) are initiated by writing a command sequence to the command registers. Since these write accesses are performed by TRACE32, they are not visible to the user by default.

The TRACE32 **SYStem.LOG.List** window provides now the possibility to log all read and write accesses performed by TRACE32 in order to get full visibility. To restrict the logging to the read/write accesses to the FLASH device, it is recommended to close all TRACE32 windows, that display memory.

By using the **SYStem.LOG.List** window, you can check the following:

1. **Does TRACE32 use the correct addresses for the FLASH command registers?**

   Wrong command register addresses lead to the assumption, that the wrong *<family_code>* or the wrong *<bus_width>* is used in the TRACE32 FLASH declaration.

2. **Does TRACE32 write the required data values in the correct endianness to the command registers?**

   Differences in the endianness used for the data values lead to the assumption that the FLASH is connected to your PowerPC CPU in true little endian mode. In this case only target-controlled FLASH programming can be used.

3. **Can TRACE32 read the FLASH status correctly?**

   If TRACE32 is unable to read the FLASH status, this leads to the assumption, that the data cache is on.

SYStem.LOG.List *<size> <file>*	Log all read/write accesses performed by TRACE32. For support purposes it is possible to save the logging to a file.

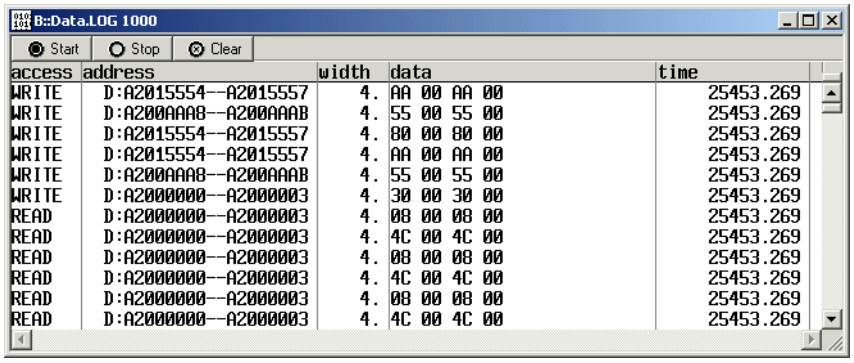
The former **Data.LOG** command is now called **SYStem.LOG.List**.

Example for TRACE32 screen based logging:

```
WinCLEAR                        ; Close all TRACE32 windows

SYStem.LOG.List                 ; Open a data log window

…                               ; Perform your test sequence
```

By default the **SYStem.LOG.List** window can log up to 256 read/write accesses. To log more read/write accesses, define the required size.

```
WinCLEAR                        ; Close all TRACE32 windows

SYStem.LOG.List 1000.           ; Open a data log window

…                               ; Perform your test sequence
```

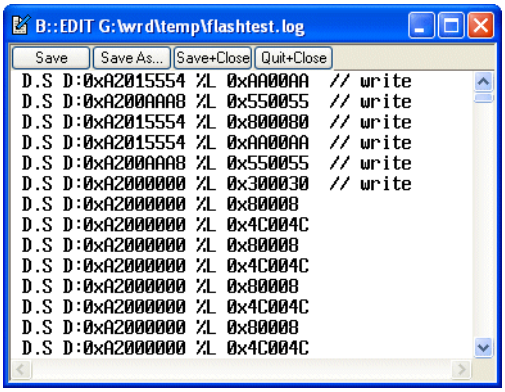


access	address	width	data	time
WRITE	D:A2015554--A2015557	4.	AA 00 AA 00	25453.269
WRITE	D:A200AAA8--A200AAAB	4.	55 00 55 00	25453.269
WRITE	D:A2015554--A2015557	4.	80 00 80 00	25453.269
WRITE	D:A2015554--A2015557	4.	AA 00 AA 00	25453.269
WRITE	D:A200AAA8--A200AAAB	4.	55 00 55 00	25453.269
WRITE	D:A2000000--A2000003	4.	30 00 30 00	25453.269
READ	D:A2000000--A2000003	4.	08 00 08 00	25453.269
READ	D:A2000000--A2000003	4.	4C 00 4C 00	25453.269
READ	D:A2000000--A2000003	4.	08 00 08 00	25453.269
READ	D:A2000000--A2000003	4.	4C 00 4C 00	25453.269
READ	D:A2000000--A2000003	4.	08 00 08 00	25453.269
READ	D:A2000000--A2000003	4.	4C 00 4C 00	25453.269

An unlimited number of read/write accesses can be logged, if the logging is redirected to a file.

```
WinCLEAR                          ; Close all TRACE32 windows

SYStem.LOG.OPEN flashtest         ; Open a file to store the data log
                                  ; information (standard extension .log)

…                                 ; Perform your test sequence

SYStem.LOG.CLOSE                  ; Close the log file
```

Example:



If the read/write accesses performed by TRACE32 are logged into a file, they are converted to **Data.Set** commands. This way it is possible to re-run the log with a TRACE32 Instruction Set Simulator.

The following shows a step by step interpretation of the TRACE32 read/write accesses logged to the **SYStem.LOG.List** window.

FLASH configuration: 2 AM29BL162C (16-bit mode) in parallel to realize a 32-bit bus width

Tested FLASH command: Sector erase

```
FLASH.Create … Long                    ; FLASH declaration

FLASH.Erase 0xa2000000++0x3fff         ; Erase the defined FLASH sector
```

1. Check the command register addresses

Please refer to the command definition for the sector erase in the data sheet for your FLASH device.

Am29BL162C Command Definitions

Command Sequence (Note 1)	Cycles	Bus Cycles (Notes 2–5)											
		First		Second		Third		Fourth		Fifth		Sixth	
		Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Sector Erase	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30

- TRACE32 uses byte addresses to write to the FLASH command registers. If word addresses are used in the data sheet of the FLASH device, they have to be converted to byte addresses.
- If parallel FLASHs are used to realize the required *<bus_width>*, this has also to be taken into account:

2 parallel FLASHs	Multiply the byte address of the command register by 2
4 parallel FLASHs	Multiply the byte address of the command register by 4

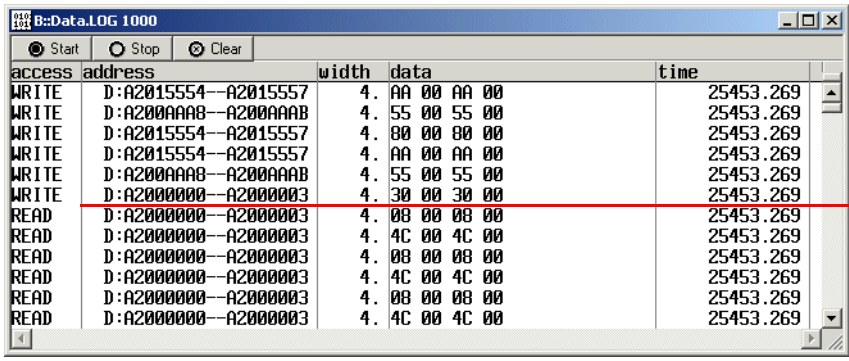
- Due to compatibility reasons TRACE32 uses 15 bit command register addresses instead of 11.

  Calculation example:

Command register address for first bus cycle from data command definition	0x555 (word address)
Conversion from word address to byte address	2 * 0x555 = 0xAAA
Multiplication for parallel FLASHs	2 * 0xAAA = 0x1554
Conversion from 11 bit address into 15 bit command register address	0x15554

The **SYStem.LOG.List** window shows that TRACE32 uses the correct command register addresses.

The **width** column indicates the *<bus_width>* used by TRACE32. 4. shows that TRACE32 perform 32 bit write accesses.



2. Check the data values

Please refer to the command definition for the sector erase in the data sheet for your FLASH device.

Am29BL162C Command Definitions

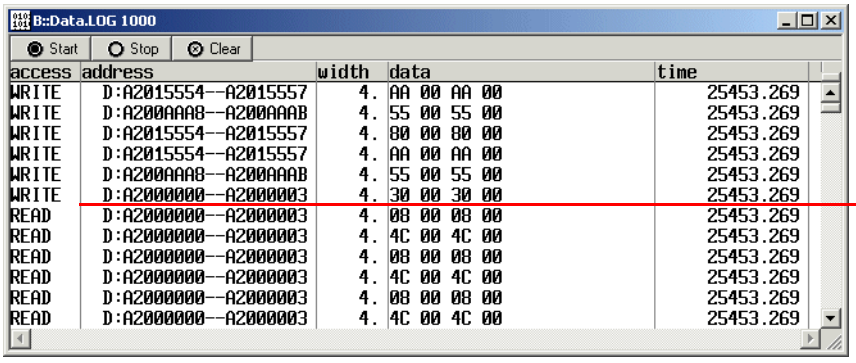
Command Sequence (Note 1)	Cycles	Bus Cycles (Notes 2–5)											
		First		Second		Third		Fourth		Fifth		Sixth	
		Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Sector Erase	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30

The **width** column in the **SYStem.LOG.List** window indicates the *<bus_width>* used by TRACE32.

The byte order is as follows:

Byte Order	
Little Endian Mode	B0 B1 B2 B3
Big Endian Mode	B3 B2 B1 B0

The **SYStem.LOG.List** window shows that TRACE32 writes the correct data values in little endian mode. Since 2 parallel FLASHs are used, B0 and B2 are used to write the data value.



3. Check the status read

After TRACE32 wrote the require 6 commands to erase the FLASH sector, it starts reading the FLASH status.

Please refer to the write operation status information in the data sheet for your FLASH device.

Write Operation Status

Operation		DQ7 (Note 2)	DQ6	DQ5 (Note 1)	DQ3	DQ2 (Note 2)	RY/BY#
Standard Mode	Embedded Program Algorithm	DQ7#	Toggle	0	N/A	No toggle	0
	Embedded Erase Algorithm	0	Toggle	0	1	Toggle	0

The following status bits are of importance:

Status Bits	
DQ2/6	Bit 2 and Bit 6 need to toggle at each status read
DQ5	DQ5==1 && DQ7==0 indicates a time-out caused by an internal FLASH error
DQ7	A soon as DQ7 becomes 1, erasing of the FLASH sector is done

The following **SYStem.LOG.List** window show that DQ2 and DQ6 are toggling at each status read.

B::Data.LOG 1000

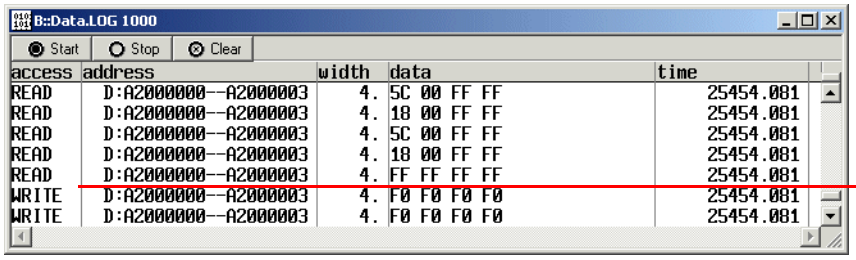
Start | Stop | Clear

access	address	width	data	time
WRITE	D:A2015554--A2015557	4.	AA 00 AA 00	25453.269
WRITE	D:A200AAA8--A200AAAB	4.	55 00 55 00	25453.269
WRITE	D:A2015554--A2015557	4.	80 00 80 00	25453.269
WRITE	D:A2015554--A2015557	4.	AA 00 AA 00	25453.269
WRITE	D:A200AAA8--A200AAAB	4.	55 00 55 00	25453.269
WRITE	D:A2000000--A2000003	4.	30 00 30 00	25453.269
READ	D:A2000000--A2000003	4.	08 00 08 00	25453.269
READ	D:A2000000--A2000003	4.	4C 00 4C 00	25453.269
READ	D:A2000000--A2000003	4.	08 00 08 00	25453.269
READ	D:A2000000--A2000003	4.	4C 00 4C 00	25453.269
READ	D:A2000000--A2000003	4.	08 00 08 00	25453.269
READ	D:A2000000--A2000003	4.	4C 00 4C 00	25453.269

Write command sequence

Read status

The **SYStem.LOG.List** window below shows that erasing the sector is done (DQ7==1). Afterwards the FLASH is switched back to read mode.



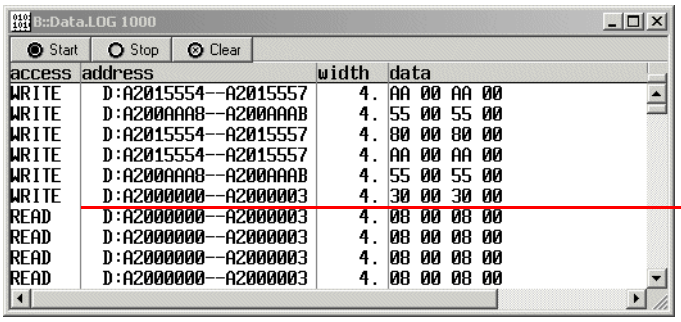
Read status

Sector erase for both FLASH devices is done

Write command to switch FLASH to read mode

The picture above shows, that the sector erase for one FLASH performed faster (Status already 0xff 0xff).

Erasing the FLASH sector failed because FLASH area is indicated as cacheable and data cache is on.



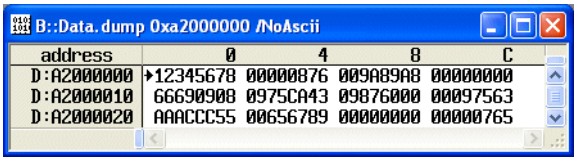
Write command sequence

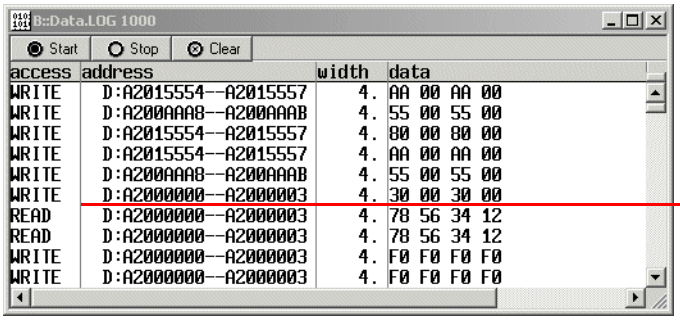
Read status

Status can only be read once

Since the FLASH area is cached to the data cache, the status never changes. As a result, TRACE32 can´t read the ready status (DQ7==1). This situation leads to an **erase timeout error**.

Erasing the FLASH sector failed, because TRACE32 has no write access to the FLASH





Write command sequence

FLASH is still in read mode,
FLASH contents is read instead of status

&base=<*start_address_of_flash*>

FLASH Width BYTE

CODE	Test write access

AM29F010
AM29LV010
AM29M010
AT49F010
SST29F010
	`Data.Set &base+0x5555 %Byte 0xaa`
	`Data.Set &base+0x2aaa %Byte 0x55`
	`Data.Set &base+0x5555 %Byte 0x90`

AM29F100B
AM29LV100B
AM29LV256B
AM29M100B
AM29M256B
AM29N256B
AT49F100B
	`Data.Set &base+0xaaaa %Byte 0xaa`
	`Data.Set &base+0x5555 %Byte 0x55`
	`Data.Set &base+0xaaaa %Byte 0x90`

CODE	Test write access	Test PPC target for true little endian mode
AM29F010 AM29LV010 AM29M010 AT49F010 SST29F010	D.S &base+0xaaaa %W 0xaaaa D.S &base+0x5554 %W 0x5555 D.S &base+0xaaaa %W 0x9090	
AM29F100 AM29LV100 AM29LV256 AM29M100 AM29M256 AM29N256 AT49F100	D.S &base+0xaaaa %W 0x00aa D.S &base+0x5554 %W 0x0055 D.S &base+0xaaaa %W 0x0090	D.S &base+0xaaaa %W 0xaa00 D.S &base+0x5554 %W 0x5500 D.S &base+0xaaaa %W 0x9000
AM29F100B AM29LV100B AM29LV256B AM29M100B AM29M256B AM29N256B AT49F100B	D.S &base+0x15554 %W 0xaaaa D.S &base+0xaaaa %W 0x5555 D.S &base+0x15554 %W 0x9090	
AM29BDDW	D.S &base+0x15554 %W 0x00aa D.S &base+0xaaaa %W 0x0055 D.S &base+0x15554 %W 0x0090	D.S &base+0x15554 %W 0xaa00 D.S &base+0xaaaa %W 0x5500 D.S &base+0x15554 %W 0x9000

CODE	

AM29F010
AM29LV010
AM29M010
AT49F010
SST29F010
	`Data.Set &base+0x15554 %Long 0xaaaaaaaa`
	`Data.Set &base+0xaaa8 %Long 0x55555555`
	`Data.Set &base+0x15554 %Long 0x90909090`

AM29F100 AM29LV100 AM29LV256 AM29M100 AM29M256 AM29N256 AT49F100	**Test write access**

	`Data.Set &base+0x15554 %Long 0x00aa00aa`
	`Data.Set &base+0xaaa8 %Long 0x00550055`
	`Data.Set &base+0x15554 %Long 0x00900090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x15554 %Long 0xaa00aa00`
	`Data.Set &base+0xaaa8 %Long 0x55005500`
	`Data.Set &base+0x15554 %Long 0x90009000`

AM29F100B
AM29LV100B
AM29LV256B
AM29M100B
AM29M256B
AM29N256B
AT49F100B
	`Data.Set &base+0x2aaa8 %Long 0xaaaaaaaa`
	`Data.Set &base+0x15554 %Long 0x55555555`
	`Data.Set &base+0x2aaa8 %Long 0x90909090`

AM29BDD	**Test write access**

	`Data.Set &base+0x15554 %Long 0x000000aa`
	`Data.Set &base+0xaaa8 %Long 0x00000055`
	`Data.Set &base+0x15554 %Long 0x00000090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x15554 %Long 0xaa000000`
	`Data.Set &base+0xaaa8 %Long 0x55000000`
	`Data.Set &base+0x15554 %Long 0x90000000`

CODE	
AM29BDDW	**Test write access**

	`Data.Set &base+0x2aaa8 %Long 0x00aa00aa`
	`Data.Set &base+0x15554 %Long 0x00550055`
	`Data.Set &base+0x2aaa8 %Long 0x00900090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x2aaa8 %Long 0xaa00aa00`
	`Data.Set &base+0x15554 %Long 0x55005500`
	`Data.Set &base+0x2aaa8 %Long 0x90009000`

AM29M2562	**Test write access**

	`Data.Set &base+0x15554 %Long 0x0000aaaa`
	`Data.Set &base+0xaaa8 %Long 0x00005555`
	`Data.Set &base+0x15554 %Long 0x00009090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x15554 %Long 0xaaaa0000`
	`Data.Set &base+0xaaa8 %Long 0x55550000`
	`Data.Set &base+0x15554 %Long 0x90900000`

CODE	

AM29F010
AM29LV010
AM29M010
AT49F010
SST29F010
	`Data.Set &base+0x2aaa8 %Quad 0xaaaaaaaaaaaaaaaa`
	`Data.Set &base+0x15550 %Quad 0x5555555555555555`
	`Data.Set &base+0x2aaa8 %Quad 0x9090909090909090`

AM29F100 AM29LV100 AM29LV256 AM29M100 AM29M256 AM29N256 AT49F100	**Test write access**

	`Data.Set &base+0x2aaa8 %Quad 0x00aa00aa00aa00aa`
	`Data.Set &base+0x15550 %Quad 0x0055005500550055`
	`Data.Set &base+0x2aaa8 %Quad 0x0090009000900090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x2aaa8 %Quad 0xaa00aa00aa00aa00`
	`Data.Set &base+0x15550 %Quad 0x5500550055005500`
	`Data.Set &base+0x2aaa8 %Quad 0x9000900090009000`

AM29F100B
AM29LV100B
AM29LV256B
AM29M100B
AM29M256B
AM29N256B
AT49F100B
	`Data.Set &base+0x55550 %Quad 0xaaaaaaaaaaaaaaaa`
	`Data.Set &base+0x2aaa8 %Quad 0x5555555555555555`
	`Data.Set &base+0x55550 %Quad 0x9090909090909090`

AM29BDD	**Test write access**

	`Data.Set &base+0x2aaa8 %Quad 0x000000aa000000aa`
	`Data.Set &base+0x15550 %Quad 0x0000005500000055`
	`Data.Set &base+0x2aaa8 %Quad 0x0000009000000090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x2aaa8 %Quad 0xaa000000aa000000`
	`Data.Set &base+0x15550 %Quad 0x5500000055000000`
	`Data.Set &base+0x2aaa8 %Quad 0x9000000090000000`

CODE	
AM29BDDW	**Test write access**

	`Data.Set &base+0x55550 %Quad 0x00aa00aa00aa00aa`
	`Data.Set &base+0x2aaa8 %Quad 0x0055005500550055`
	`Data.Set &base+0x55550 %Quad 0x0090009000900090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x55550 %Quad 0xaa00aa00aa00aa00`
	`Data.Set &base+0x2aaa8 %Quad 0x5500550055005500`
	`Data.Set &base+0x55550 %Quad 0x9000900090009000`

AM29M2562	**Test write access**

	`Data.Set &base+0x2aaa8 %Quad 0x0000aaaa0000aaaa`
	`Data.Set &base+0x15550 %Quad 0x0000555500005555`
	`Data.Set &base+0x2aaa8 %Quad 0x0000909000009090`

	Test PPC target for true little endian mode

	`Data.Set &base+0x2aaa8 %Quad 0xaaaa0000aaaa0000`
	`Data.Set &base+0x15550 %Quad 0x5555000055550000`
	`Data.Set &base+0x2aaa8 %Quad 0x9090000090900000`