

# **Establish Your Debug Session**

Release 02.2024



TRACE32 Online Help	
TRACE32 Directory	
TRACE32 Index	
TRACE32 Debugger Getting Started	
Establish Your Debug Session	1
History	4
Establish your Debug Session	5
Key TRACE32 Setup Commands	5
The PER.view/PER.Set Command	5
The Data.LOAD Command	7
Debug Scenarios	10
Establish the Debug Communication	12
Debug Scenario 1	18
Onchip/NOR Flash Programming	19
The Flash Programming File	19
On-chip Flash Programming	20
Off-chip NOR Flash Programming	25
Configure the TRACE32 OS Awareness	36
Debug Scenario 2	37
Typical Boot Sequence	37
Flash Programming (NAND/Serial/eMMC)	42
The Flash Programming File and the Debug Symbol File	42
NAND Flash Programming (non-generic NAND Flash Controller)	43
eMMC Flash Programming	55
Establish the Communication	56
Load the Debug Symbols	56
Debug Scenario 3	57
Run the Boot Loader	58
Load Application (and/or OS) Code and Debug Symbols	59
Load Debug Symbols only	59
Configure the TRACE32 OS Awareness	59
Complete Setup Example	59
Debug Scenario 4	60
Write a Script to Configure the Target	61
Load Application (and/or OS) Code and Debug Symbols	61
Configure the TRACE32 OS Awareness	61

Start-Up Scripts	62
Write a Start-Up Script	62
Run a Start-up Script	63
Automated Start-up Scripts	64

Version 04-Mar-2024

# History

16-Oct-2014 Initial version of the manual.

23-Jan-2020 Full revision of the manual.

Before you can start debugging, the debug environment has to be set up. The necessary setup depends crucially on the debug scenario.

# Key TRACE32 Setup Commands

#### The PER.view/PER.Set Command

A debug setup requires various configurations in core/chip related configuration registers.

The main commands to perform these configurations are:

<b>PER.Set.simple</b> <address> <range> [%<format>] <string></string></format></range></address>	Modify configuration register/on- chip peripheral
Data.Set <address> <range> [%<format>] <string></string></format></range></address>	Modify memory-mapped configuration register/on-chip peripheral

The main command to inspect the configurations is:

PER.view <file> [<tree\_search\_item>] /SpotLight Display the configuration registers/onchip peripherals, highlight changes ; Display "Watchdog Timer" configuration registers, highlight changes
; a comma is used instead of the <file>
PER.view , "Watchdog Timer" /SpotLight

B::PER.view	, "Watch	dog Ti	mer" /Sp	otLight				×
■ <u>Watchdoq</u> WTCON WTDAT WTCNT	Timer 8021 8000 6429	PV CRV CV	80 8000 6429	WT Enabled	CS 16	IG Disabled	RE Enabled	^^
•			III					<u>ار</u> ا

; Disable Watchdog timer by configuring Watchdog Timer Control Register ; (WTCON)

PER.Set.simple 0x53000000 %Long 0x0

B::PER.view	, "Watch	dog Ti	mer" /Sp	otLig	Iht				×
Watchdog	Timer	PV	00	WT	Disabled	CS 16	IG Disabled	RF Disabled	_ ^
WTDAT	8000 4458	CRV CV	8000 4458		orodored	0.0		NE DISUBICU	
•			III						•

A debug setup requires to load the code to be debugged and to load the debug symbols. TRACE32 PowerView supports a wide range of compilers and compiler output formats.

To get a list of all compilers supported for your core proceed as follows:

1. Open the **Processor Architecture Manual** for your core.



2. Refer to the compiler list provided by this manual. The **Option** column lists the supported output formats.

Þ	deb	ugger_arm.pdf - Adobe Acrobat Pro						
F	ile I	Edit View Window Help						×
	1	Create 🗸 🛛 🎦 📄 🖨 🖂 🛛 🏟 🌻	> 🄛 💊 💊					all and a second
1	٢	130 / 146   I 🖑   - + 89,55	% -   占 🕼			Tool	s Comment	Share
[		Bookmarks	Compilers					-
	P		Language	Compiler	Company	Option	Comment	
	0	ARM and XSCALE Debugger	C	HIGH-C	ABC International	ELF/DWARE		
	IJ.	- Warning	C	CARM	ARM Germany GmbH	ELF/DWARF		-
		Cuick Start of the ITAG Debugger	C	ARMCC	ARM Ltd.	AIF		
	4J2		С	ARMCC	ARM Ltd.	ELF/DWARF		
		■ Troubleshooting	С	REALVIEW MDK	ARM Ltd.	ELF/DWARF2		
		<sup>®</sup> ↓ FAQ	С	GCCARM	Free Software Foundation, Inc.	COFF/STABS		
		Symmetric Multiprocessing	С	GCCARM	Free Software Foundation, Inc.	ELF/DWARF2		
		ARM Specific Implementations	С	GREENHILLS C	Greenhills Software Inc.	ELF/DWARF2		
			C	ICCARM	IAR Systems AB	ELF/DWARF2		
		ARM specific System Commands     ARM Specific Benchmarking	С	ICCV7-ARM	Imagecraft Creations Inc.	ELF/DWARF	ARM7	
		Commands	С	TI-C	Texas Instruments	COFF		
		APM Specific TrOnchin Commande	C	GNU-C	Wind River Systems	COFF		
		ARM specific fronchip commands	C++	HIGH-C++	ARC International	ELF/DWARF		
			C++	ARM SDT 2.50	ARM Ltd.	ELF/DWARF2		
		TrBus Commands	C++	REALVIEW MDK	ARM Ltd.	ELF/DWARF2		_
		🖲 🖵 Target Adaption	C++	GCCARM	Free Software Foundation, Inc.	COFF/STABS		
		Support	C++	GNU	Free Software Foundation, Inc.	EXE/STABS		
		Compilers	C++	GCCARM	Free Software Foundation, Inc.	ELF/DWARF2		
		Paaltime Operation Systems	C++	GREENHILLS C++	Greenhills Software Inc.	ELF/DWARF2		
		Realtime Operation Systems	C++	MSVC	Microsoft Corporation	EXE/CV5	WindowsCE	
		"If arty Tool Integrations	C/C++	XCODE	Apple Inc.	Mach-O		
		Products	C/C++	VX-ARM	TASKING	ELF/DWARF2		
L								-

The most important commands to load the code to be debugged and the debug symbols are:

Data.LOAD. <sub_cmd> <file> I<option></option></file></sub_cmd>	Load code and debug symbols
Data.LOAD.Binary <file> I<option></option></file>	Load only code
Data.LOAD. <sub_cmd> <file> /NoCODE /<option></option></file></sub_cmd>	Load only debug symbols
Data.LOAD.Elf demo-flash.elf	; Load code and debug symbols from ; ELF file
Data.LOAD.AIF demo.axf	; Load code and debug symbols from ; AIF file
Data.LOAD.Elf *	<pre>; Load code and debug symbols from ; ELF file : open file browser to select file</pre>
Data.LOAD.Elf demo.elf /NoCODE	; Load debug symbols from ELF file
Data.LOAD.Binary my_app.bin	; Load code from binary file

A in-depth introduction to the **Data.LOAD** command is given in the chapter "Load the Application **Program**" (training\_hll.pdf).

Today most applications use an operating system. TRACE32 PowerView includes a configurable target-OS debugger to provide symbolic debugging of operating systems.

Lauterbach provides ready-to-run configuration files for most common available OSes.

#### To get the appropriate information on your OS, proceed as follows:

1. Open the online help and deactivate the help filter.

Help		
🤋 Ca	ontents	
The In	dex <sup>k3</sup>	
ji ji Fir	nd	
E: Tr	ree la	
🯄 Oj	perating System User Manual	
🤰 Pr	rocessor Architecture Manual	
🤰 De	D HELP	×
1 Ti	? Contents 🛐 Index 🏦 Find 🛱 Command Tree 📮 Bookmarks 📇 Print	
🔹 Po	⊕ open all ⊖ close all ⊕ more ⊖ less □ use filter: bdmarm;icrstm;icretm;	
🛓 Tr		
Ex		-
La	About the TRACE32 Online Help     TRACE32 Clossary     TRACE32 Debucar Cetting Started	
Su	TRACE32 Debugger dectring Started	
Ał	J ⊞ TRACE32 Training ⊞ TRACE32 Installation ⊞ TRACE32 Technical Support	11
	TRACE32 Index	-
	( )	ы

2. Open the TRACE32 OS Awareness Manual for your operating system.

B HELP	- • ×
🕐 Contents 📉 Index 🏦 Find 🛱 Command Tree 📕 Bookmarks 📇 Print	
🕒 open all 🕒 close all 🕀 more 🕞 less 🗌 use filter: bdmarm;icrstm;icretm;	
	A
AULOFOCUS USER'S GUIDE     AuloFocus	
System Trace	_
Bootloader Awareness Manuals	=
🕀 UEFI Debuggers	
😑 OS Awareness Manuals	
TRACE32 Extension Development	
BOS Awareness Manual AMA	
DOS Awareness Manual Akin	
OS Awareness Manual ARTX-166	
OS Awareness Manual ChibiOS/RT	
🕀 05 Awareness Manual Chorus Classic	
■ OS Awareness Manual Chorus Micro	
OS Awareness Manual Cmicro	
B US Awareness Manual CMX	
B OS Awareness Manual DSP/BIOS	-
<	

The necessary setup for your debug session depends crucially on the debug scenario. The graphic below shows you that there are mainly four debug scenarios.



After the communication between the debugger and the core(s) is established, there a four debug scenarios. Each debug scenario requires a different setup.

#### Debug Scenario 1

The boot loader or the application (and/or the operating system) under debug is running out of Onchip/NOR flash.

#### • Debug Scenario 2

The boot loader under debug is running out of a flash e.g. a NAND or serial flash.

#### • Debug Scenario 3

The application (and/or the operating system) under debug are running out of RAM and a readyto-run boot loader configures the target system and especially the RAM for this debug scenario.

#### • Debug Scenario 4

The application (and/or the operating system) under debug are running out of RAM. The target configuration, especially the RAM configuration has to be done by TRACE32 commands, because there is no ready-to-run boot loader.



#### 1. Select the target core/chip



Inform the debugger about the core/chip on your target, if an automatic detection of the core/chip is not possible. Wild card symbols \* or ? are allowed.

# SYStem.DETECT CPU SYStem.CPU <cpu>

Auto detection of CPU Select the CPU/chip

SYStem.CPU CortexR5

SYStem.CPU CortexR5\*

#### 2. Adjust the JTAG clock

The debugger uses a default JTAG clock of 10 MHz. Adjusting the JTAG clock might be necessary:

- if a fixed relation between the core clock and the JTAG clock is specified.
- if RTCK has to be used as JTAG clock for an ARM core.



#### SYStem.JtagClock <frequency>

Select the JTAG clock

SYStem.JtagClock 1.MHz

SYStem.JtagClock 100.kHz

SYStem.JtagClock RTCK

#### 3. Set the required options for your core/chip

Some cores/chips require additional settings before the communication can be established.

B::SYStem				- • ×	
- Mode	MemAccess	- Option	Option	DisMode	
Own	O DAP	IMASKASM	DACR	AUTO	
© NoDebug	© TSMON3	IMASKHLL	MMUSPACES	ACCESS	
Prepare	© RealMON	TURBO	MPU	C ARM	
🔘 Go	C TrkMON	BigEndian	CFLUSH	C THUMB	
O Attach	C GdbMON	ResBreak		L	A datition and a sufficiency
StandBy	Oenied	INTDIS	AMBA	· · · · · · · · · · · · · · · · · · ·	Additional settings
🔿 Up (StandBy)	- CpuAccess	DBGACK	NODATA	CONFIG	
🔘 Up	Enable	🗷 EnReset	EXEC	DETECT	
	Oenied	TRST	SPLIT		
reset	Nonstop	PWRDWN			
RESetOut		– WaitReset ––––			
		OFF			
- CPU	JtagClock				
CortexR5	10.0MHz -				
Bernard and the second s					

For details refer to the Processor Architecture Manual.



#### 4. Establish the communication

The most common way to establish the communication between the debugger and the core(s) is Up.

	B::SYStem				
	Mode	MemAccess	Option	Option	DisMode
	O Down	© DAP	IMASKASM	DACR	AUTO
	NoDebug	© TSMON3	IMASKHLL	MMUSPACES	ACCESS
	Prepare	RealMON	TURBO	MPU	© ARM
	© Go	C TrkMON	BigEndian	CFLUSH	© THUMB
	Attach	C GdbMON	ResBreak		L
	StandBy	Oenied	INTDIS	AMBA	
	O Up (StandBy)	- CpuAccess	DBGACK	NODATA	CONFIG
	🔍 Up	Enable	🗹 EnReset	EXEC	DETECT
-		Oenied	TRST	SPLIT	
	reset	Nonstop	PWRDWN		
	RESetOut		– WaitReset –		
			100.000ms		
	- CPU	JtagClock			
	S5PV310	12.0MHz 🔻			

If *Up* is selected, the following steps are performed:

- Reset of the core/chip
- Initialization of the communication between the debugger and the core(s)
- Stop of the core(s) at the reset vector



A second useful way to establish the communication between the debugger and the core/chip is *Attach*. *Attach* allows to connect the debugger to an already running core/chip.

	B::SYStem				
	Mode	MemAccess	Option	Option	DisMode
	Own	O DAP	IMASKASM	DACR	AUTO
	NoDebug	© TSMON3	IMASKHLL	MMUSPACES	ACCESS
	Prepare	RealMON	TURBO	MPU	© ARM
	© Go	C TrkMON	🔲 BigEndian	CFLUSH	© ТНИМВ
	R Attach	GdbMON	ResBreak		
_	StandBy	Oenied	INTDIS	AMBA	
	O Up (StandBy)	- CpuAccess	DBGACK	NODATA	CONFIG
	© Up	© Enable	EnReset	EXEC	DETECT
		Oenied	TRST	SPLIT	
	reset	Nonstop	PWRDWN		
	RESetOut		- WaitReset		
			100.000ms		
	CPU	JtagClock			
	CortexA8	12.0MHz -			

If *Attach* is selected, the following step is performed:

1. Initialization of the communication between the debugger and the core(s).

SYStem.Mode Attach	Establish the communication between the debugger and the target core(s) (without reset)			
SYStem.Mode Attach				
Break	; stop the program execution			

The boot loader or the application (and/or the operating system) under debug is running out of the on-chip flash or out of a NOR flash device.



\*Considering the circumstance that a process has to be started manually e.g. via a TERMinal window

The debugger supports the programming of on-chip flash and off-chip NOR flash devices.

**NOTE:** Flash programming requires that data cache is disabled for the address range covered by the FLASH.

#### The Flash Programming File

On-chip flash and off-chip NOR flash programming allows to load any output file generated by your compiler.

Videos about the on-chip flash programming can be found here: support.lauterbach.com/kb/articles/flash-programming

Ready-to-run scripts for most on-chip flashs can be found in ~~/demo/<architecture>/flash/<cpu>.cmm

- e.g.~~/demo/arm/flash/mk20.cmm
- e.g. ~~/demo/powerpc/flash/mpc5xxx.cmm

To program the software to the on-chip flash of your processor/chip proceed as follows:

1. Start the script appropriate for your processor/chip.



A B::B::CD.DO *								
Computer > System (C:) > T32_MPC > demo > powerpc > flash > + 4 Search flash								
Organize ▼ New folder 🗄 ▼ 🗍 🔞								
★ Favorites	Name	Date modified	Туре	Size	-			
🧮 Desktop	퉬 byte	10.06.2013 11:23	File folder					
😺 Downloads	🔒 long	10.06.2013 11:23	File folder					
🕮 Recent Places	퉬 long_tle	10.06.2013 11:23	File folder					
	퉬 quad	10.06.2013 11:23	File folder					
🧮 Desktop	퉬 quad_tle	10.06.2013 11:23	File folder		=			
🥃 Libraries	퉬 word	10.06.2013 11:23	File folder					
😹 amartin	퉬 word_tle	10.06.2013 11:23	File folder					
👰 Computer	🛋 apm86xxx-nand2g08.cmm	30.01.2013 10:15	CMM File	4 KI	B			
🏭 System (C:)	😰 jpc560s_quadspi.cmm	07.12.2011 10:35	CMM File	8 KI	в			
🔮 DVD RW Drive (D:) 🗧	😰 jpc560x.cmm	28.03.2013 14:59	CMM File	9 KI	В			
🖵 SYS (\\OESI2) (F:)	🖬 jpc563xm.cmm	28.03.2013 14:59	CMM File	5 KI	В			
🖵 VOL_BUSINESS (\\OESI3) (G:)	😰 jpc563xm_rev0.cmm	29.11.2012 15:38	CMM File	4 KI	В			
🖵 VOL_DEVEL (\\OESI2) (H:)	🖻 jpc564xa.cmm	28.03.2013 14:59	CMM File	6 KI	В			
Solution (\\OESI2) (I:)	🖻 jpc564xbc.cmm	28.03.2013 14:59	CMM File	9 KI	В			
🖵 _HOME (\\OESI2\VOL_DEVEL) (J	🖻 jpc564xl.cmm	28.03.2013 14:59	CMM File	5 KI	В			
🖵 L_DRIVE (\\OESI1) (L:)	🖻 jpc567xk.cmm	28.03.2013 14:59	CMM File	7 KI	В			
🖵 T32NEW (\\OESI1) (T:)	🖻 jpc574xk.cmm	28.03.2013 14:59	CMM File	5 KI	В			
🐺 ZENworks Adaptive Agent	🖬 jpc574xm.cmm	28.03.2013 14:59	CMM File	5 KI	В			
🙀 Network	🖬 jpc577xm.cmm	28.03.2013 14:59	CMM File	7 KI	В			
😝 Control Panel	🖻 mpc5xxx.cmm	28.03.2013 14:59	CMM File	3 KI	B 🗸			
🥘 Recycle Bin 👻	•	III			Þ			
File name: mnc5vvv c	File name: mnc5vx cmm							
Inpessare								
			Open 🔻	Cancel				

2. TRACE32 PowerView informs you when all preparations are done. Please confirm that you are ready to choose the boot loader or the application to be programmed.



3. Please select the boot loader or application to be programmed.

A B::Data.LOAD.auto * E:0x00(0x100	0000-1)						
🚱 🔵 🗢 📕 « mpc563x > mpc563x_spc563x 🔹 🗣 🍫 Search mpc563x_spc563x 🔎							
Organize 🔻 New folder			:= - 1 🔞				
☆ Favorites	•	Name	Date modified				
🧮 Desktop		🖻 demo.cmm	09.02.2012 15:28				
👔 😺 Downloads	-	diabc.c	09.02.2012 15:28				
🖳 Recent Places	=	diabc.x	09.02.2012 15:28				
		🖻 errata_e_5200.cmm	09.02.2012 15:28				
🥽 Libraries		📧 mmu_init.cmm	09.02.2012 15:28				
Documents		🖻 sram_init.cmm	09.02.2012 15:28				
J Music							
Pictures							
Videos							
📲 Computer	-	٠ III					
File name: dia	abc.x	•	Current (*) (*) 🔻				
			Open Cancel				

TRACE32 PowerView informs you, that the programming is done.

B::
file 'C:\T32_MPC\demo\powerpc\hardware\mpc56xx\mpc563x_spc563x\diabc.x' (ELF/DWARF) loaded.
emulate trigger devices trace Data Var List PERF SYStem Step
SF:40000004 \\diabc\Global\_start

If the boot loader/application is compiled with debug symbols they are automatically loaded into TRACE32 PowerView with the flash programming.

For details on the on-chip flash programming open the flash programming script.

; ~~ represents the TRACE32 installation directory **PEDIT ~~/demo/powerpc/flash/mpc5xxx.cmm** 



If you create your own start-up script for your target hardware, please call the flash programming script from there.

If you leave the flash programming script unchanged, you can always replace it with its most current version.

The following parameters can be used, when the flash programming script is called:

CPU= <cpu></cpu>	If a FLASH programming script supports a CPU family, you can provide your target CPU as parameter.
PREPAREONLY	Advise the FLASH programming script to prepare the FLASH programming by declaring the FLASH sectors and by linking the appropriate programming binary. The FLASH programming commands are bypassed.
DUALPORT=0 1	Disable/enable DualPort FLASH programming. For all processors/cores that allow to <b>write to physical memory</b> <b>while the CPU is running</b> a higher FLASH programming performance can be achieved by the use of <b>DualPort FLASH</b> <b>Programming</b>

Not every script supports all parameters. The parameters relevant for your script are described in the meta data section of the script.

	; GTitle: Generic script for Freescale MK20, MK21 and MK22 internal flash									
	; @Description:									
	, ; Example for flash declaration of Freescale MK20, MK21 and MK22 internal ; flash.									
Γ	; Script arguments:									
	; ; D0 mk20 [PREPAREONLY] [CPU= <cpu>] [DUALPORT=0 1] [MASSERASE]</cpu>									
	; ; PREPAREONLY only declares flash but does not execute flash programming									
	; ; CPU= <cpu> selects CPU derivative <cpu>. <cpu> can be CPU name out of the ; table listed below. For these derivatives the flash declaration ; is done by the script.</cpu></cpu></cpu>									
	; ; DUALPORT default value is 0 (disabled). If DualPort mode is enabled ; flash algorithm stays running until flash programming is ; finished. Data is tranferred via dual port memory access.									
	; ; MASSERASE forces mass erase of device before establishing debug connection									
	; ; For example:									
	; D0 ~-/demo/arm/flash/mk20 CPU=MK20DN512VLK10 DUALPORT=1 PREPAREONLY									
	; List of MK20/MK21/MK22 derivatives and their configuration:									
	; CPU-Type Flash ProgFlash FlexNVM EEPROM RamSize ; type [Byte] [Byte] [Byte] [Byte]									
	; MK2ODN32VEX5 0 32KB 8KB ; MK2ODN32VFM5 0 32KB 8KB ; MK2ODN32VFT5 0 32KB 8KB									

The following framework can be used to call the flash programming script from your start-up script.

```
----
DO <flash_script> [CPU=<cpu>] PREPAREONLY [DUALPORT=0|1]
; program file to on-chip FLASH
FLASH.ReProgram ALL /Erase
Data.LOAD.Elf <file>
FLASH.ReProgram off
; reset processor/chip
; might be necessary to reset all target settings made by the flash
; programming script
SYStem.Up
; continue with start-up script
;...
```

More details on the on-chip flash programming can be found in "Onchip/NOR FLASH Programming User's Guide" (norflash.pdf).

TRACE32 PowerView provides two methods to program off-chip NOR flash:

#### 1. Tool-based programming

Tool-based programming means that the flash programming algorithm is part of the TRACE32 software. Tool-based programming is easy to configure but slow.

#### 2. Target-controlled programming

Target-controlled flash programming means that the underlying flash programming algorithm is detached from the TRACE32 software. Target-controlled flash programming works as follows:

- 1. The flash algorithm is downloaded to the target RAM.
- 2. The programming data are downloaded to the target RAM.
- 3. The flash algorithm running in the target RAM programs the data to the flash devices.

Target-controlled flash programming minimizes the communication between the host and the debugger hardware. This makes target-controlled flash programming fast.

**NOTE:** It is recommended to start with tool-based flash programming. If this works properly you can switch to target-controlled flash programming.

#### Programming off-chip NOR flash requires the following steps (see next page):



#### 1. Disable Internal and External Watchdog

Example

```
; Display "Watchdog Timer" configuration registers, highlight changes PER.view , "Watchdog Timer" /SpotLight
```

Seperative www. "Watchdog Timer" /SpotLight								
■ <u>Watchdog</u> WTCON WTDAT WTCNT	Timer 8021 8000 <mark>6429</mark>	PV CRV CV	80 8000 6429	WT Enabled	CS <b>16</b>	IG Disabled	RE Enabled	^
•			111					

; Disable Watchdog timer by configuring Watchdog Timer Control Register ; (WTCON)

PER.Set.simple 0x53000000 %Long 0x0

Í	🛹 B::PER.view , "Wat	chdog Timer" /S	potLight				x
	Watchdog Tim WTCON 000	er • PV <b>00</b>	WT Disabled	CS 16	IG Disabled	RE Disabled	- ^
	WTDAT 800 WTCNT 445	CRV 8000 CV 4458					
	•	"	1				► a

#### 2. Disable Data Cache

The data cache has to be disabled for the address ranges of all flash devices to enable TRACE32 PowerView to read the flash status information.

Example

```
; Display the memory management configuration registers ; highlight changes
```

PER.view , "Core Registers, Memory Management Unit" /SpotLight

B::PER.view , "Core Registers,Men	nory Manage	ement Unit" /SpotLig	ht				x
Memory Management Un SCTLR 00C50070	IT NMFI V SW M	ARM Disabled 0x00000000 Disabled Disabled	AFE EE I C	Disabled Little Disabled Enabled	TRE RR Z A	Disabled Random Disabled Disabled	*
< III						1	h. 4

; Disable Data Cache by configuring the control register SCTLR **PER.Set.simple C15:0x1 %Long 0x550078** 

🖝 B::PER.view , "Core Registers,Memory Management Unit" /SpotLight 📃 💷 🗈					23		
<u> Memory Management Unit</u> <u> SCTLR</u> <u> 00550078</u>	TE NMFI V SW M	ARM Disabled 0x00000000 Disabled Disabled	AFE EE I C	Disabled Little Disabled <mark>Disabled</mark>	TRE RR Z A	Disabled Random Disabled Disabled	- -
•						,	. at

#### 3. Make sure that the core has write access to the flash

NOR flash programming requires that the core has write access to the flash device(s).

The following settings in the **bus configuration** have to be done for each NOR flash device:

- Definition of the base address of the NOR flash device
- Definition of the size of the NOR flash device
- Definition of the bus size that is used to access the NOR flash device
- The write access has to be enabled for the NOR flash device
- Definition of the timing (number of wait states for the write access to the NOR flash device)

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

#### **Example for ColdFire**

#### Bus configuration after reset:

🛹 B::PER.view		
■ EBU External Bus Inte	rface	
⊟ Address Region Ø		
ADDSEL0 0000000	BASE 00000000 ALTSEG 0 MASK [31:27] ALTENAB dis REGEN dis	
BUSCONØ <b>80928000</b>	WRITE dis AGEN demul WAIT off PORTW 32-bit BCGEN ctrl sig WAITINY no PREFETCH no DLOAD always ENDIAN1 little CMULT 32 CTYPE 0 AALIGN chip-sel WEAKPREFETCH no MULTMAP 00	
BUSAPØ FFFFFFF	ADDRC 3 AHOLDC 3 CMDDELAY 7 WAITRDC 7 WAITWRC 7 BURSTC 7 DATAC 3 RDRECOVC 7 WRRECOVC 7 DTARDWR 15 DTACS 15	
⊞ Address Region 1		
		-
	III	► a

In order to have write access to the used off-chip NOR flash device the *Address Region 0* has to be configured for the following characteristics:

- Base address 0xa0000000
- Size 16 MByte
- Bus size 16 bit

PER.Set <address> %<format> <value> Data.Set <address> %<format> <value>

PER.view , /SpotLight	<pre>; Highlight all changed ; configuration registers</pre>
PER.Set 0xf0000080 %Long 0xA0000031	; ADDSEL0
PER.Set 0xf00000c0 %Long 0x00508637	; BUSCONO

Correct bus configuration for NOR flash programming.

SiPER.view , /SpotLight	
EBU External Bus Interface	
Address Region 0	
HUDSELO HUDDOUDI BHSE HUDDOUDO HEISEG O MHSK L31:241 HEIENHB OIS REGEN ENG	
BUSCONØ 00508637 WRITE ena AGEN demul WAIT off	
PORIM <mark>16-bit</mark> BUGEN ctrl sig WAIIINV no PREFEICH no DLOAD <mark>if available</mark> ENDIAN1 little CMULT 32	
CTYPE 1 AALIGN portw fld WEAKPREFETCH no MULTMAP 37	=
BUSAPØ FFFFFFF ADDRC 3 AHOLDC 3 CMDDELAY 7 WAITRDC 7	
WAITWRC 7 BURSTC 7 DATAC 3 RDRECOVC 7	
WRRECOVE 7 DIHRDWR 15 DIHES 15	
Address Region 1	
۲	

FLASH.RESet	Reset the FLASH declaration table
FLASH.CFI <start_address> <data_bus_width></data_bus_width></start_address>	<ul> <li>Generate a FLASH declaration by evaluating the Common Flash Interface description inside the FLASH device.</li> <li>The command FLASH.CFI requires the definition of</li> <li>the <i><start_address></start_address></i> of the FLASH device</li> <li>the <i><data_bus_width></data_bus_width></i> that is used by the core to access the FLASH device</li> </ul>
FLASH.List	List the FLASH declaration table
FLASH.UNLOCK ALL	Unlock the FLASH sectors
FLASH.ReProgram ALL   OFF	Enable/disable the FLASH programming
Data.LOAD. <sub_cmd> <file> I<option></option></file></sub_cmd>	Load code and debug symbols

More details on the concepts of the TRACE32 NOR flash programming can be found in "Onchip/NOR FLASH Programming User's Guide" (norflash.pdf).

If your FLASH device doesn't provide CFI please refer to "**Onchip/NOR FLASH Programming User's Guide**" (norflash.pdf) for details on the FLASH programming procedure.

: Reset the FLASH declaration table FLASH.RESet FLASH.CFI 0xa000000 Word ; Generate a FLASH declaration via : CFI FLASH.List ; Display the FLASH declaration ; table ; Unlock the FLASH device if FLASH.UNLOCK ALL ; required ; e.g. some FLASH devices are ; locked after power on ; Enable the FLASH for programming FLASH.ReProgram ALL Data.LOAD.Elf demo.elf ; Specify the file that contains ; the code and the debug symbols FLASH.ReProgram off ; Program FLASH and disable ; the FLASH programming afterwards Data.LOAD.Elf demo.elf /DIFF ; Verify the FLASH programming IF FOUND() PRINT "Verify error after FLASH programming" ELSE PRINT "FLASH programming completed successfully"

FLASH.RESet	Reset the FLASH declaration table
FLASH.CFI <start_address> <bus_width> /TARG</bus_width></start_address>	ET <code_range> <data_range></data_range></code_range>
	<ul> <li>Generate a FLASH declaration by evaluating the Common Flash Interface description inside the FLASH device.</li> <li>The command FLASH.CFI requires the definition of</li> <li>the <i><start_address></start_address></i> of the FLASH device</li> <li>the <i><data_bus_width></data_bus_width></i> that is used by the core to access the FLASH device</li> <li>the target RAM location <i><code_range></code_range></i> for the flash programming algorithm</li> <li>the target RAM location <i><data_range></data_range></i> for the flash programming data</li> </ul>
FLASH.List	List the FLASH declaration table
FLASH.UNLOCK ALL	Unlock the FLASH sectors
FLASH.ReProgram ALL   OFF	Enable/disable the FLASH programming
Data.LOAD. <sub_cmd> <file> I<option></option></file></sub_cmd>	Load code and debug symbols

Required size for the code is size\_of(file) + 32 byte

Flash programming algorithm 32 byte

Memory mapping for the <code\_range>

**Details on** *<data\_range>* 

The parameter <data\_range> specifies the RAM location for the data, especially

- the <*data\_buffer\_size*> for the programming data. Recommended buffer size is 4 KByte, smaller buffer sizes are also possible. The max. buffer size is 16 KByte
- the argument buffer for the communication between TRACE32 PowerView and the programming algorithm
- the stack

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

	Memory mapping for the <i><data_range< i="">&gt;</data_range<></i>
64 byte argument buffer	Memory mapping for the <uua_range></uua_range>
Data buffer for data transfer between TRACE32 and flash programming algorithm <buffer_size> calculated as described above</buffer_size>	
256 byte stack	

#### 4. Enable RAM Access

Target-controlled Flash Programming requires, that the core has access to the RAM locations specified for <*code\_range>* and *<data\_range>*.

If this is not the case the following settings in the **bus configuration** have to be done for an off-chip RAM:

- Definition of the base address of the RAM
- Definition of the size of the RAM
- Definition of the bus size that is used to access the RAM
- Definition of the timing (number of wait states for the RAM access)

#### Example

```
; reset the FLASH declaration table
FLASH.RESet
; set up the FLASH declaration for target-controlled programming
; target RAM at address 0x2000000
FLASH.CFI 0x0 Word /TARGET 0x2000000++0xfff 0x20001000++0xfff
; display FLASH declaration table
FLASH.List
; unlock the FLASH device if required for a power-up locked device
: FLASH.UNLOCK ALL
; enable the programming for all declared FLASH devices
FLASH.ReProgram ALL
; specify the file that contains the code and the debug symbols
Data.LOAD.Elf demo.elf
; program the file and disable the FLASH programming afterwards
FLASH.ReProgram off
; verify the FLASH contents
Data.LOAD.Elf demo.elf /DIFF
IF FOUND()
    PRINT "Verify error after FLASH programming"
ELSE
    PRINT "FLASH programming completed successfully"
. . .
```

### Configure the TRACE32 OS Awareness

Refer to "The TASK.CONFIG Command", page 9 for details.

The boot loader under debug is running out of a flash e.g. a NAND flash.

In contrast to NOR flash, code can not be executed out of NAND or serial flash. The code has always to be copied to RAM before it can be executed.

# **Typical Boot Sequence**

Before the setup for debug scenario 2 is described, it might be useful to have a look at a typical boot sequence. If the boot loader is running out of flash the system start-up might include the following steps:

#### 1. Reset

At RESET the boot loader is copied from the flash to an on-chip SRAM, which is mapped to the reset vector. The boot loader starts afterwards.

Please be aware, that some core(s) require a correct ECC for this copy procedure.



#### 2. Boot loader is running

The main task of the boot loader is to initialize the SDRAM and to copy the second stage boot loader to SDRAM. When this is done the control is passed to the second stage boot loader.



#### 3. Second stage boot loader is running

The main task of the second stage boot loader is to copy the kernel image to SDRAM. When this is done the control is passed to the kernel.



Flash



**NOTE:** Flash programming requires that data cache and MMU are disabled.

#### The Flash Programming File and the Debug Symbol File

Flash programming can only program binary files. Therefore two output files have to be generated by the compiler:

- A binary file (e.g. boot.bin)
- A file containing the debug symbols (e.g. boot.elf)

Ready-to-run flash programming scripts for most processors/chips can be found in the directory  $\sim\sim/demo/<architecture>/flash$ 

Folder and file name convention:

```
~~/demo/<architecture>/flash/<cpu_name>-<prefix_of_nand_flash_code>.cmm
```

Get <cpu\_name> from the CPU column of the list of "Supported NAND/Serial Flash Controller" on the Lauterbach home page (www.lauterbach.com) if the CONTROLLER column does not indicate generic.

Firefox 🔻					inserile state and sugar	
TRACE32® Supported NAND/SERIAL FL	+			C an	and an and a second	▽
www.lauterbach.com/frames.html	l?home.html			☆ ▽ C 🚼 - englisch wörterbu	ich 🔎	n 🗈 🦗 -
🥝 Disable• 👗 Cookies• 🌶 CSS• 🗋 Form	is* 🔟 Image	s* 🕕 Information* 🧧 Miscellaneous*	🥖 Outline* 🥒 Resize* 💥 Tools*	■ View Source		88 🗸
	[links] [c	ompile] [activate] [refresh] [get files]	[show files] [report error] www	lauterbach.com	sitemap  print	login   impressum
						control panel 💌
	FI	reescale Semiconductor, Inc	5.			▲ ^
Site Search in site	C	PU	CONTROLLER	COMMENT		STATE
Chip Search for chip	1.1	MX21	imx	NAND		
	1.1	MX23	gpmimx23	NAND		
Products	· .	MX25	imx25	NAND		
	1.1	MX25	ecspi	SPI(eCSPI)		_
Tool Chain	1.1	MX27	imx	NAND		III
Supported Compilers	1.1	MX27	imx	SPI(CSPI)		
Supported Host	1.1	MX28	gpmimx28	NAND		
Operating Systems	1.1	MX31	imx	NAND		
Supported Flash Devices	= 1.M	MX31	imx	SPI(CSPI)		
Supported NAND/Serial Flash	1.1	MX35	imx25	NAND		
Controller Controller	1.1	MX35	ecspi	SPI(eCSPI)		
Operating Systems	1.1	VIX51	imx51	NAND		
Supported Tool Integrations	1.1	VIX51	ecspi	SPI(eCSPI)		
Supported Simulators/Virtual	1.1	MX51	imx51	eMMC/SD/MMC		
Prototypes/Target Servers	1.1	MX53	imx51	NAND		
Support	1.1	MX6	ecspi	SPI(eCSPI)		
http://www.lauterbach.com/ylistnand.html		MX6	anmimx6	NAND		•

If the **CONTROLLER** column indicates **generic** refer to "**NAND Flash Programming (generic NAND Flash Controller)**", page 46.

Get the prefix of the <nand\_flash\_code> from the CODE column of the list of "Supported Flash Devices" on the Lauterbach home page (www.lauterbach.com).

Firefox *			Children containing States in-		
TRACES2® Supported FLASH Devices	+			-	
www.lauterbach.com/frames.h	tml?home	e.html	☆ マ C 🚦	¶ ▼ Google	P ▲ ■ ₩
🕗 Disable* 👗 Cookies* 💉 CSS* 일 Fo	orms* 🔛	Images  🕕 Information 📃 Mi	scellaneous* 🥖 Outline* 🥖	Resize* 🎇 Tools* 🔳 View S	ourcer 🔝 Options  🖌
	(lin	iks] [compile] [activate] [refres]	h] [get files] [show files] [re	eport error] www.lauterbac	h.com
		Hynix semiconducto	r		
Site Search in site		TYPE	COMPANY	CODE	COMMENT
Chip Search for chip		H27U518S	HYNIX	NAND1208	
		H8KDS0UN0MER	HYNIX	NAND2G16	NAND Flash
		HY27SS08561A	HYNIX	NAND5608L	NAND Flash
El Products	- Â	HY27SS16561A	HYNIX	NAND5616L	NAND Flash
El Features		HY27UF082G2A	HYNIX	NAND2G08	NAND-Flash
Tool Chain		HY27UF082G2M	HYNIX	NAND2G08L	NAND-Flash
Supported Compilers		HY27UF162G2A	HYNIX	NAND2G16	NAND-Flash
Supported Host     Operating Systems		HY27UF162G2B	HYNIX	NAND2G16	NAND-Flash
Supported Flash Devices	=	HY27UF162G2M	HYNIX	NAND2G16L	NAND-Flash
Supported MAND(A rial Flach		HY27UG082G2B	HYNIX	NAND2G08	NAND-Flash
Controller		HY27US08561A	HYNIX	NAND5608L	NAND Flash
Supported Target		HY27US16561A	HYNIX	NAND5616L	NAND Flash
Operating Systems		HY29DL162	HYNIX	AM29LV100	16-bit mode
Supported Tool Integrations				AM29LV100B	8-bit mode
Supported Simulators/Virtual Prototypes/Target Servers		HY29DL163	HYNIX	AM29LV100	16-bit mode
Support				AM29LV100B	8-bit mode
Support		HY29DS162	HYNIX	AM29LV100	16-bit mode
ttp://www.lauterbach.com/ylist.html			m		E F

Name of flash programming script here ~~/demo/arm/flash/imx31-nand2g08.cmm

To program the code to flash proceed as follows:

1. Start the script appropriate for your processor/chip and appropriate for your flash device.



2. TRACE32 PowerView informs you when all preparations are done. Please confirm that you are ready to choose the boot loader binary to be programmed.



3. Please select the boot loader to be programmed.



Detail on NAND flash programming can be found in "NAND FLASH Programming User's Guide" (nandflash.pdf).

The **CONTROLLER** column of the list of "**Supported NAND/Serial Flash Controller**" on the Lauterbach home page (www.lauterbach.com) indicates "generic", if a processor/chip has a generic NAND flash controller.

Firefox  Fir	+		and the		
www.lauterbach.com/frames.html?	?home.hti	ml		☆ ▽ C 🚼 ▼ engliscl	n wörterbuch 🔎 🍙 💽 🐖 🔹
🥝 Disable* 👗 Cookies* 💉 CSS* 🚺 Forms	s• 🔛 Ima	ages 🕕 Information	🛛 🧧 Miscellaneous* 🥖 Outline* 🥒 Resize* 💥	Tools* 🔳 View Source* 🔝 Opti	ons* 🗸 😵 🗸
	[links]	[compile] [activate]	[refresh] [get files] [show files] [report error]	www.lauterbach.com	sitemap  print  login  impressum
DEVELOPMENT TOOLS					<< >> control panel 💌
		MSM7500	qsd8650	NAND	
Site Search in site		QSC60X5	qsc60x5	NAND	
Chip Search for chip		QSC62XX	qsd8650	NAND	
		QSD8650	qsd8650	NAND	
Products	^	Renesas Techr	nology, Corp.		
Features		CDU	CONTROLLER	CONVENT	
Tool Chain		CPU	CONTROLLER	COMMENT	STATE
Supported Compilers		SH7204	sh7264	INAND	
<ul> <li>Supported Host</li> <li>Operating Systems</li> </ul>	E	311/204	\$11/204	<b>SFI</b>	
Supported Flash Devices		Samsung Semi	iconductor		
Supported NAND/Serial Flash Controller		CPU	CONTROLLER	COMMENT	STATE
Supported Target		S3C24XX	generic	NAND	
Operating Systems		S3C6410	s3c6410	OneNAND	
Supported Tool Integrations		S3C6410	s3c6410	eMMC/SD/MMC	
Supported Simulators/Virtual Prototypes/Target Servers		S3C64XX	generic	NAND	
Support					http://www.lauterbach.com/ylistnand.html
× Suchen: generic 🗸 Ab <u>w</u>	vārts 👚	Aufwärts 🖌 <u>H</u> ervorhe	eben 🔲 <u>G</u> roß-/Kleinschreibung		

Programming script for generic NAND flash controller have to be written by the user.

Programming a flash device with the help of a generic NAND flash controller requires the following steps:



#### 1. Prepare the NAND FLASH Controller and the NAND FLASH for Programming

Programming a flash device requires a proper initialization of the flash controller and the bus interface. The following settings might be necessary:

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

Use the **PER.Set/PER.view** commands for this setup.

Example for s3c2410X (ARM920T):

PER.view , "NAND Flash Controller" /SpotLight

🗢 B::PER.view , "	NAND Flash Co	ntroller" /Sp	otLight			uentointois antein			×
🖻 NAND Flash	Controlle	r							
NFCONF	00004800	ENABLE	Disabled	INIECC	Not initial	ized NFMCE	V Inactive	TACLS 0	
		TWRPH0	0	TWRPH1	0				
NECMD	00000000	COMMAND	00		-				
NEADDR	00000000	ADDRESS	00						
NEDATA	00000000	ΔΤΔ	00						
NESTAT	00008301	RNR	Ready						
NEECC	00559595	ECC2	55	ECC1	95	ECC0	95		
in coc	00555555	2002		LCCI		LCCO			-
									1111
									<ul> <li></li></ul>

; enable and configure NAND flash controller **PER.Set 0x4E000000 %Long 0xE030** 

B::PER.view , "I	NAND Flash Co	ntroller" /Sp	otLight							×
🗆 NAND Flash	Controlle	r								
NFCONF	0000E030	ENABLE TWRPH0	Enabled 3	INIECC TWRPH1	Not 0	initialized	NFMCEN	Active	TACLS 0	
NFCMD	00000000	COMMAND	00							
NFADDR	00000000	ADDRESS	00							
NFDATA	00000004	DATA	04							
NESTAT	00008301	RNB	Ready							
NFECC	00C0C00C	ECC2	C0 -	ECC1	C0		ECC0	0C		
										-
•										►

#### 2. Enable RAM Access

TRACE32 PowerView runs the flash programming algorithm in target RAM. It requires at least 16 KByte of RAM for this purpose.

This requires that the core has access to target RAM.

If the core has no access to target RAM, the access to target RAM has to be set up.

Correct settings in the bus configuration registers are key for the RAM access. The following settings in the bus configuration have to be done:

- Definition of the RAM base address
- Definition of the RAM size
- Definition of the bus size that is used to access the RAM
- The write access has to be enabled for the RAM
- Definition of the timing (number of wait states for the write access to the RAM)

Use the **PER.Set/PER.view** commands for this setup.

#### Example SDRAM configuration on an s3c2410X (ARM920T):

#### Bus configuration after reset:

B::PER.view , /Spo	otLight						
	isters						*
B Memory Contr	oller						
BWSCON 0	0000000	ST7 ST6 ST5 ST4 ST3 ST2 ST1 DW0	No UB/LB No UB/LB No UB/LB No UB/LB No UB/LB No UB/LB No UB/LB Reserved	WS7 WS6 WS5 WS4 WS3 WS2 WS1	Disabled Disabled Disabled Disabled Disabled Disabled Disabled Disabled	DW7 DW6 DW5 DW4 DW3 DW2 DW1	8-bit 8-bit 8-bit 8-bit 8-bit 8-bit 8-bit
BANK CONTROL	REGISTE	R(BANKCONN	N:nGCS0-nGCS5	)	0 clock	TACC	14 clocks
BANKCONO U	0000700	TCOH	0 clock	TCAH	0 clock	TACP	2 clocks
BANKCON1 0	0000700	TACS TCOH	uata 0 clock 0 clock	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON2 0	0000700	TACS TCOH	0 clock 0 clock 1 data	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON3 0	0000700	TACS TCOH PMC	0 clock 0 clock 1 data	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON4 0	0000700	TACS TCOH PMC	0 clock 0 clock 1 data	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON5 0	0000700	TACS TCOH PMC	0 clock 0 clock 1 data	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON6 0	0018008	MT	Sync DRAM	TRCD	4 clocks	SCAN	8-bit
BANKCON7 0	0018008	МТ	Sync DRAM	TRCD	4 clocks	SCAN	8-bit
REFRESH CONT	ROL REGI	STER					
REFRESH 0	0AC0000	REFEN TSRC	Enabled 7 clocks	TREFMD REFCNT	Auto 0000	TRP	4 clocks
BANKSIZE REG	ISTER	DUDCT EN	Dicabled	SCKE EN	Disabled	SCLK EN	Disabled
GANNAILE U		BK76MAP	128MB/128MB	SCRE_EN	UISANIEU	SCEN_EN	DISADIEU
MRSRB6 0	0000030	WBL	Burst	ТМ	Test mode	CL	3 clocks
MRSRB7 0	0000030	BT WBL BT	Sequential Burst Sequential	BL TM BL	1 Test mode 1	CL	3 clocks
1							

```
        PER.Set
        0x4800000
        %Long
        0x2222D222

        PER.Set
        0x4800004
        %Long
        0x0000700

        PER.Set
        0x4800000
        %Long
        0x0000700

        PER.Set
        0x4800000
        %Long
        0x0000700

        PER.Set
        0x4800000
        %Long
        0x000014

        PER.Set
        0x4800014
        %Long
        0x0000700

        PER.Set
        0x48000014
        %Long
        0x0000700

        PER.Set
        0x48000016
        %Long
        0x00000700

        PER.Set
        0x48000012
        %Long
        0x000018005

        PER.Set
        0x48000020
        %Long
        0x00018005

        PER.Set
        0x4800028
        %Long
        0x000032

        PER.Set
        0x4800028
        %Long
        0x0000032

        PER.Set
        0x4800020
        %Long
        0x0000032

        PER.Set
        0x4800020
        %Long
        0x0000032

        PER.Set
        0x4800030
        %Long
        0x0000030

        PER.Set
        0x4800030
        %Long
        0x0000030
```

Correct bus configuration for SDRAM usage:

B::PER.view , /	SpotLight						
⊞ <u>ARM Core R</u>	egisters						^
E Memory Con	troller						
BWSCON	22220220	ST7 ST6 ST5 ST4 ST3 ST2 ST1	No UB/LB No UB/LB No UB/LB No UB/LB UB/LB No UB/LB	WS7 WS6 WS5 WS4 WS3 WS2	Disabled Disabled Disabled Disabled Enabled Disabled Disabled	DW7 DW6 DW5 DW4 DW3 DW2 DW1	32-bit 32-bit 32-bit 32-bit 16-bit 32-bit
		DWO	Reserved	WOI	DISabled	DWI	32-010
BANK CONTR	OL REGISTE	R(BANKCON	N:nGCS0-nGCS5	)		7100	14 1 1
BANKCONU	00000700	TACS TCOH PMC	0 clock 0 clock 1 data	TCAH	0 clock	TACP	2 clocks
BANKCON1	00007FF0	TACS TCOH	4 clocks 4 clocks	TCOS TCAH	4 clocks 4 clocks	TACC TACP	14 clocks 2 clocks
BANKCON2	00000700	TACS TCOH	0 clock 0 clock 1 data	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON3	00001F4C	TACS TCOH	0 clock 1 clock	TCOS TCAH	<mark>4 clocks</mark> 0 clock	TACC TACP	14 clocks <mark>6 clocks</mark>
BANKCON4	00000700	TACS TCOH	0 clock 0 clock	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON5	00000700	TACS TCOH	0 clock 0 clock	TCOS TCAH	0 clock 0 clock	TACC TACP	14 clocks 2 clocks
BANKCON6	00018005	MT	Sync DRAM	TRCD	3 clocks	SCAN	9-bit
BANKCON7	00018005	MT	Sync DRAM	TRCD	3 clocks	SCAN	9-bit
REFRESH CO	NTROL REGI	STER	Epobled	TREEMD	Auto	TRD	
REFRESH	000E0459	TSRC	7 clocks	REFCNT	0459	TKP	2 CTOCKS
BANKSIZE R BANKSIZE	EGISTER 00000032	BURST_EN BK76MAP	Disabled 128MB/128MB	SCKE_EN	Enabled	SCLK_EN	Enabled
SDRAM MODE	REGISTER	SET REGIST	TER				
MRSRB6	0000030	WBL	Burst	TM	Test mode	CL	3 clocks
MRSRB7	0000030	WBL BT	Burst Sequential	TM BL	Test mode 1	CL	3 clocks
•			III				۲. ۲

#### 3. Disable internal and external watchdog

Example (ARM920T):

```
; Display "Watchdog Timer" configuration registers, highlight changes PER.view , "Watchdog Timer" /SpotLight
```

🗢 B::PER.view , "Watchdog Timer" /SpotLight											
■ <u>Watchdog</u> WTCON WTDAT WTCNT	Timer 8021 8000 <mark>6429</mark>	PV CRV CV	80 8000 6429	WT Enabled	CS <b>16</b>	IG Disabled	RE Enabled	^			
•			111								

; Disable Watchdog timer by configuring Watchdog Timer Control Register ; (WTCON)

PER.Set.simple 0x53000000 %Long 0x0

ĺ	🗢 B::PER.view , "Wa	atchdog Timer" /S	ootLight				×
	□ <u>Watchdog Tin</u> WTCON 000 WTDAT 800 WTCNT 445	ner 00 PV 00 00 CRV 8000 58 CV 4458	WT Disabled	CS 16	IG Disabled	RE <mark>Disabled</mark>	^
	•	11	1				►

The following commands are useful, if a generic NAND flash controller is used to program a flash. For details refer to "NAND FLASH Programming User's Guide" (nandflash.pdf).

Reset NAND flash programming to default.
Inform TRACE32 PowerView on the NAND flash registers
Inform TRACE32 PowerView on all details about flash programming algorithm.
Erase NAND flash.
Program binary file to NAND flash.

More details on the FLASHFILE.TARGET command:

The name of the flash programming algorithm depends on the NAND flash to be programmed (e.g. nand5608.bin for the K9F5608 NAND flash from Samsung).

Firefox +								<b>X</b>
TRACE32® Supported FLASH Devices	+	-100-100 (S)	* * 6 * 1		i i i i internetione			~
( www.lauterbach.com/frames.html	l?home.h	tml		☆ ▼ C	🚼 👻 englisch wörterbuch	\$		. 🐢 .
🖉 Dirabler 🖡 Cookiert 🖉 CSSt 📋 Form	cr 🕅 In	agest 🔒 Informations	Miscellaneoust / Outlinet	Regizer Stanler I View So	urcer I Ontionst			- C -
Obsable: S Cookies: Z Cook S D Point	5. 69 11	lages.	Miscellaneous. 2 Outline.	View 30	urce. W obtions.			• • •
	[links	] [compile] [activate]	refresh] [get files] [show files]	[report error] www.lauterbach.	.com   site	map   print	login  impr	essum
							control pa	
Development rools					*	·  »	control pa	nei 💌
		K8P6415	SAMSUNG	AM29LV100				
Site Search in site		K8Q2815	SAMSUNG	AM29LV100				
Chip Search for chip		K8S2815	SAMSUNG	AM29LV100				
		K8S3215	SAMSUNG	AM29LV100				
		K8S5615	SAMSUNG	AM29LV100				
Products	<u> </u>	K8S6415	SAMSUNG	AM29LV100				
		K9F1208	SAMSUNG	NAND1208	NAND Flash			
Tool Chain		K9F1G08	SAMSUNG	NAND1G08	NAND Flash			
Supported Compilers		K9F2G08	SAMSUNG	NAND2G08	NAND Flash			
Supported Host		K9F4G08	SAMSUNG	NAND2G08	NAND Flash			
Operating Systems     Supported Electropy	=	K9F5608	SAMSUNG	NAND5608	NAND Flash			
Supported Plash Devices		K9F5616	SAMSUNG	NAND5616	NAND Flash			
Controller		K9F8G08	SAMSUNG	NAND8G08	NAND Flash, 4KB Pag	ge		
Supported Target		K9G8G08	SAMSUNG	NAND2G08	NAND Flash, 2KB Pag	je		
Operating Systems		K9GAG08U0D	SAMSUNG	NANDLBG08XS	NAND Flash, 4K/218E	1		
Supported Tool Integrations		K9HBG08U	SAMSUNG	NANDLAG08	NAND Flash			
Supported Simulators/Virtual		K9HCG08U	SAMSUNG	NANDLBG08	NAND Flash			
Prototypes/Target Servers		K9HCG08U1M	SAMSUNG	NANDLBG08XS	NAND Flash, 4K/218E	1		
Support		K9K1208X0C	SAMSUNG	NAND1208L	NAND Flash			
Sales				m		http://www.	lauterbach.com	n/ylist.htm
× Suchen: 5608 + Aby	wärts 🕇	Aufwärts 🖌 Hervorhe	ben 🔲 Groß-/Kleinschreibung					

Its location in the TRACE32 demo folder is defined by the number of data I/O pins between the NAND flash controller and the flash device. E.g. is there are 8 data I/O pins between the NAND flash controller and the flash device the algorithm can be found in:

~~/demo/**<architecture>**/flash/**byte**/**<nand\_flash\_code>**.bin

This flash programming algorithm is downloaded to a target RAM when flash programming is performed. Therefore TRACE32 PowerView needs to be informed about an appropriate RAM location by the *<code\_range>* parameter of the **FLASH.TARGET** program

required size for the code is size\_of(file) + 32 byte

FLASH algorithm	Memory mapping for the <code_range></code_range>
32 byte	

The parameter <data\_range> specifies the RAM location for the data, especially

- the <*data\_buffer\_size*> for the programming data. Recommended buffer size is 4 KByte, smaller buffer sizes are also possible. The max. buffer size is 16 KByte
- the argument buffer for the communication between TRACE32 PowerView and the programming algorithm
- the stack

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

64 byte argument buffer	Memory mapping for the < <i>data_range</i> >
Data buffer for data transfer between TRACE32 and NAND FLASH algorithm <buffer_size> calculated as described above</buffer_size>	
256 byte stack	

#### Example

FLASHFILE.RESet

FLASHFILE.CONFIG 0x4E000004 0x4E000008 0x4E00000C

FLASHFILE.TARGET 0x30000000++0x1FFF 0x30002000++0x3FFF
~~/demo/arm/flash/byte/nand5608.bin

FLASHFILE.Erase 0x0--0x1FFFF

FLASHFILE.LOAD boot.bin 0x0

Folder and file name convention:

~~/demo/**<architecture>**/flash/**<cpu\_name>-emmc**.cmm

Get <cpu\_name> from the CPU column of the list of "Supported NAND/Serial Flash Controller" on the Lauterbach home page (www.lauterbach.com).

Firefox TRACE32® Supported NAND/SERIAL FL	+				× •
www.lauterbach.com/frames.htm	l?home.ht	ml	☆ マ C 🛃 - Google	۹ 🗈 ۱	e -
🥝 Disable* 👗 Cookies* 🎽 CSS* 📋 Form	ns* 💷 Im	ages* 🍿 Information* 📃 Miscellaneous	🕶 🥖 Outline= 🥒 Resize= 🎇 T	ools* 🔳 View Source* 👖 Options	× 6
	[links	[compile] [activate] [refresh] [get file:	s] [show files] [report error]	www.lauterbach.com	
		Marvell, Inc.			^
Site Search in site		CPU	CONTROLLER	COMMENT	
Chip Search for chip		88F5082	generic	NAND	
		88F5181	generic	NAND	_
		88F5281	generic	NAND	
		88F6192	generic	NAND	
E Features		88F6192	marvell	SPI	
Iool Chain		88F6281	generic	NAND	
Supported Compilers		88F6281	marvell	SPI	
Operating Systems		MV76100	generic	NAND	
Supported Flash Devices	=	MV76100	marvell	SPI	
Supported NAND/Serial Flash		MV78100	generic	NAND	
Controller (m)		MV78100	marvell	SPI	E
Supported Target		MV78200	generic	NAND	
Operating Systems		MV78200	marvell	SPI	
Supported Tool Integrations		PXA3XX	oxa	NAND	
Supported Simulators/Virtual     Brottoturage (Target Service)		PXA920	рха	eMMC	
El Support		PXA978	рха	eMMC/SD/MMC	
a support		PXA978	рха	eMMC/SD/MMC	-
http://www.lauterbach.com/ylistnand.html		ш			•

#### Name of flash programming script here

~~/demo/**arm**/flash/**pxa920-emmc**.cmm

Detail on eMMC flash programming can be found in "eMMC FLASH Programming User's Guide" (emmcflash.pdf).

It is required to establish the communication between the debugger and the core by **SYStem.Up**. This advise the processor/chip to reset before the communication is established (details can be found on "Establish your Debug Session", page 5).

# Load the Debug Symbols

; load debug symbols for boot loader and second stage boot loader ; address of boot loader: 0x0--0x3fff ; address of second stage boot loader: 0x33f80000++0x3ffff ; symbol mapping has to be accordingly Data.LOAD.Elf boot.elf /NoCODE

If you want to debug only the second stage boot loader you can set an on-chip breakpoint to its start address:

```
Break.Set start_boot2 /Onchip
```

The application (and/or the operating system) under debug are running out of RAM and a ready-to-run boot loader configures the target system and especially the RAM for this debug scenario.



\*Considering the circumstance that a process has to be started manually e.g. via a TERMinal window

The most important command to run the boot loader are:

Go	Start program execution
Break	Stop program execution
WAIT <time></time>	Wait the defined time (for scripts only)
Go <address></address>	Run the program until the specified address is reached
Break.Set <address></address>	Set a breakpoint to the specified address

#### Example 1

Go	; start the program execution
Break	; stop the program execution after ; the target initialization is done

#### Example 2

;	; script example
Go	; start the program execution
WAIT 0.5s	; wait 500. ms
Break	; stop the program execution
;	; continue with other setups

#### Example 3

Go 0xc0001000	; run the program until the
	; complete setup is done

#### Example 4

Break.Set 0xc0001000	; set a breakpoint to the end ; of the boot loader
Go	; start the program execution
WAIT !STATE.RUN()	; wait until the program stops ; at the end of the boot loader
;	; continue with other setups

### Load Application (and/or OS) Code and Debug Symbols

If the boot loader does not load the application (and/or OS) you can perform the loading by the following command:

Data.LOAD.Elf my\_app.elf

# Load Debug Symbols only

If the boot loader loads the application (and/or OS) code to RAM you need only to load the debug symbols.

Data.LOAD.Elf my\_app.elf /NoCODE

# **Configure the TRACE32 OS Awareness**

Refer to "The TASK.CONFIG Command", page 9 for details.

### **Complete Setup Example**

Example for a boot loader that loads the application to RAM.

```
SYStem.CPU

SYStem.Up

Go ; start the program execution

WAIT 0.5s ; wait 500.ms

Break ; stop the program execution

Data.LOAD.Elf my_app /NoCODE
```

The application (and the operating system) under debug are running out of RAM. The target configuration, especially the RAM configuration has to be done by TRACE32 commands, because there is no ready-to-run boot loader.



\*Considering the circumstance that a process has to be started manually e.g. via a TERMinal window

A minimum target configuration has to configure all used memories and the serial interface.

Use the **PER.Set/PER.view** commands for this setup.

# Load Application (and/or OS) Code and Debug Symbols

Data.LOAD.Elf my\_app.elf

# **Configure the TRACE32 OS Awareness**

Refer to "The TASK.CONFIG Command", page 9 for details.

It is strongly recommended to summarize the commands, that are used to set up the debug environment, in a start-up script. The script language PRACTICE is provided for this purpose.

The standard extension for a script file is . cmm.

# Write a Start-Up Script

The debugger provides an ASCII editor, that allows to write, to run and to debug a start-up script.

PEDIT <file></file>	Open < <i>file&gt;</i> with the script editor	
PEDIT my_startup		

The debugger provides two commands, that allow you to convert debugger configuration information to a script.

STOre <file> [<item>]</item></file>	Generate a script that allows to reproduce the current settings		
ClipSTOre [ <item>]</item>	Generate a command list in the clip-text that allows to reproduce the current settings		
STOre system_settings	SYStem	; Generate a script that allows you ; to reproduce the settings of the ; SYStem window at any time	
PEDIT system_settings		; Open the file <pre>system_settings</pre>	

ClipSTOre	SYStem	;	Generate a command list that
		;	allows you to reproduce the
		;	settings of the SYStem window
		;	at any time
		;	The generated command list can be
		;	pasted in any editor



ChDir.DO <file>

Change directory and run script

```
ChDir.DO my_startup.cmm
```

There are two ways to define a start-up script, that is automatically started, when the debugger is started.

#### 1. Define start-up script in conjunction with the executable

The debugger-executable can be started with the start-up script as parameters.

c:\t32\t32arm.exe -s g:\and\arm\**start.cmm** 

#### 2. Use T32Start to define an automated start-up script

If you use T32Start to start the debugger, an automated start-up script can be defined.

1 T32Start V2.2.22 *	<b>— — X</b>			
🔺 🛅 Configuration Tree				
⊳ - 🛅 Settings	Start			
▶ 👘 Example Configuration	Add			
ARM	1.15050111			
🖉 🖉 🖛 1: Podbus Device Chain	Delete			
⊿ · 📶 1: Power Debug II				
🖉 🐨 🐨 ConnectionType: USB	Up			
🛛 USB Settings	Down			
🔺 🐴 1: Core	0.01111			
Target: ARM/XScale/Janus	Instances			
Advanced Settings				
⊳ - 🛅 Paths				
Interfaces				
⊳ - 🛅 Display				
⊿ . CintupScript				
Source: File	Jave			
I File: U:\132_ARM\demo\arm\compiler\arm\arm.cmm	Help			
Harameters:				
	1			
File C:\T32_ARM\demo\arm\compiler\arm\arm.c Link to Edit				
ID: //Configuration2/Podbus Device Chain/Power Debug II/Core/CoreAdvancedOptions/				