LAUTERBACH
DEVELOPMENT TOOLS

CPU32/ColdFire Debugger and Trace

MANUAL

CPU32/ColdFire Debugger and Trace

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .... 🗁

ICD In-Circuit Debugger .... 🗁

Processor Architecture Manuals .... 🗁

CPU32 and ColdFire .... 🗁

CPU32/ColdFire Debugger and Trace 1

History 5

Brief Overview of Documents for New Users 5

Demo and Start-up Scripts 6

Warning 6

Quick Start of the BDM Debugger 7

Quick Start of the ROM Monitor 9

Restrictions 12

Troubleshooting 13

FAQ 13

ROM Monitor 14

Monitor Features 14

Monitor Files 14

Address Layout 15

Vector Table 16

Configuration 17

Break without Hardware Interrupt 17

CPU specific Implementations 18

Hardware Breakpoint for MC68360 18

Memory Classes 19

CPU specific SYStem Commands 20

SYStem.BdmClock	Select BDM-clock	20
SYStem.CPU	Select CPU type	21
SYStem.LOCK	Lock and tristate the debug port	21
SYStem.MemAccess	Select run-time memory access method	22
SYStem.Mode	Establish the communication with the CPU	22

SYStem.CONFIG Configure debugger according to target topology 23
Daisy-Chain Example 26
TapStates 27
SYStem.CONFIG.CORE Assign core to TRACE32 instance 28
SYStem.CONFIG.state Display target configuration 29
SYStem.Option.BASE Select peripheral base address 29
SYStem.Option.CACHE Flush instruction cache on MC68349 29
SYStem.Option.HOOK Compare PC to hook address 29
SYStem.Option.ICFLUSH Flush of instruction cache before step and go 31
SYStem.Option.IMASKASM Disable interrupts while single stepping 31
SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping 31
SYStem.Option.MMUPhysLogMemaccess Memory access preferences 31
SYStem.Option.MMUSPACES Separate address spaces by space IDs 32
SYStem.Option.OTE Ownership trace 33
SYStem.Option.SLOWRESET Slow reset enable 33
SYStem.Option.PST Detect HALT condition of the CPU 33
SYStem.Option.PSTCLKTERM Termination of the PSTCLK pin 34
SYStem.Option.ResetAction Debugger behavior when RESET is detected 34
SYStem.Option.StandbyAction Debugger behavior when power is restored 35
SYStem.Option.TracePULSE Use PULSE instruction 35
SYStem.Option.TraceWDDATA Use WDDATA instruction 35
SYStem.RESetOut Reset target without reset of debug port 35
SYStem.Option.StepFlat Step without exceptions 36

Trace specific SYStem.Option Commands 37
SYStem.Option.BTB Change the width of the address information 37
SYStem.Option.DDC Configure the tracing of data accesses 37
SYStem.Option.TSYNC Send the PC to the trace port 37

CPU specific TrOnchip Commands 38
TrOnchip.ALIGN Enable breakpoint alignment 38
TrOnchip.RESet Set on-chip trigger to default state 38
TrOnchip.state Display on-chip trigger window 38
TrOnchip.SIZE Enable break on SIZE lines 39
TrOnchip.TEnable Set filter for the trace 39
TrOnchip.TOFF Switch the sampling to the trace to OFF 39
TrOnchip.TON Switch the sampling to the trace to “ON” 39
TrOnchip.TTrigger Set a trigger for the trace 40

CPU specific MMU Commands 41
MMU.DUMP Page wise display of MMU translation table 41
MMU.List Compact display of MMU translation table 43
MMU.SCAN Load MMU table from CPU 45

BDM Connector 68K 47

BDM and Trace Connector ColdFire 47

BDM Connectors for ColdFire V1, V2, V3, V4 and ColdFire+ 47
BDM Connector 6 pin ColdFire+/V1 CPUs Debugger 47
BDM Connector 26 pin ColdFire V2 or V3 CPUs Debugger and Trace 48
BDM Connector 26 pin ColdFire V2 or V3 CPUs with ALLPST only Debugger 48
BDM Connector 26 pin ColdFire V4 CPUs Debugger and Trace 49

Technical Data BDM 68K 50
Operation Voltage 50

Technical Data BDM ColdFire 51
Operation Voltage 51

Technical Data Trace ColdFire 52
Operation Voltage 52

CPU32/ColdFire Debugger and Trace

Version 06-Jun-2024

History

20-Jul-22	For the MMU.SCAN ALL command, CLEAR is now possible as an optional second parameter.

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Training Basic Debugging”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **“OS Awareness Manuals”** (rtos_<*os*>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Demo and Start-up Scripts

Lauterbach provides ready-to-run start-up scripts for known CPU32/ColdFire based hardware.

To search for PRACTICE scripts, do one of the following in TRACE32 PowerView:

- Type at the command line: **WELCOME.SCRIPTS**
- or choose **File** menu > **Search for Script**.

  You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.

You can also manually navigate in the `~~/demo/coldfire/` subfolder of the system directory of TRACE32.

Warning

WARNING:	To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF. Recommendation for the software start: 1. Disconnect the Debug Cable from the target while the target power is off. 2. Connect the host system, the TRACE32 hardware and the Debug Cable. 3. Power ON the TRACE32 hardware. 4. Start the TRACE32 software to load the debugger firmware. 5. Connect the Debug Cable to the target. 6. Switch the target power ON. 7. Configure your debugger e.g. via a start-up script. Power down: 1. Switch off the target power. 2. Disconnect the Debug Cable from the target. 3. Close the TRACE32 software. 4. Power OFF the TRACE32 hardware.

Quick Start of the BDM Debugger

Starting up the debugger is done as follows:

1. Reset the system.

```
RESet
```

2. Specify CPU specific settings.

```
SYStem.CPU M68360

SYStem.Option.Base <address>
```

The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Map the EPROM simulator if available (optional).

```
MAP.ROM 0x0--0x1FFFF
```

This command maps a standard 8 bit wide 27x010 EPROM.

4. Enter debug mode.

```
SYStem.Up
```

This command resets the CPU and enters debug mode. After this command is executed, it is possible to access memory and registers.

Some ColdFire V1 derivatives need a Power On Reset to enter debug mode out of reset without executing code. Dependent from their actual memory contents these derivatives might stuck in a reset loop. When you enter SYStem.Up the debugger tries to get the CPU out of such a loop, but if the time between two resets is too small this attempt might fail. Use **SYStem.Mode StandBy** in this case, remove power from the target and switch it on again.

5. Load the program.

```
Data.LOAD.I mcc.abs
```

The format of the **Data.LOAD** command depends on the file format generated by the compiler. The corresponding options for all available compilers are listed in the compiler list. A detailed description of the **Data.LOAD** command is given in the **“General Commands Reference”**.

A typical start sequence without EPROM simulator is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

```
WinCLEAR                            ; Clear all windows

SYStem.Up                           ; Reset the target and enter debug mode

Data.LOAD.i mccp.x /nil             ; Load the application

Register.Set PC main                ; Set the PC to function main

Register.Set USP 0FEFF              ; Set USP to address FEFF

Register.Set SPP 0FFFF              ; Set SSP to address FFFF

List.Mix                            ; Open disassembly window         *)

PER.view                            ; Show clearly arranged peripherals
                                    ; in window                       *)

Register.view /SpotLight            ; Open register window            *)

Frame.view /Locals /Caller          ; Open the stack frame with
                                    ; local variables                 *)

Var.Watch %SpotLight flags ast      ; Open watch window for variables *)

Break.Set 0x1000 /Program           ; Set breakpoint to address 1000
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

Quick Start of the ROM Monitor

Starting up the ROM Monitor is done as follows.

1. Select the device prompt for the ICD Debugger and reset the system.

   ```
   b:
   RESet
   ```

   On all host systems except of the emulator device B is already selected. The **RESet** command is only necessary if you don't start directly after booting.

2. Specify CPU specific settings.

   ```
   SYStem.CPU M68020
   SYStem.Option.Base <address>
   ```

   The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Map the EPROM simulator.

   ```
   MAP.ROM 0x0--0x1FFFF
   ```

   This command maps a standard 8 bit wide 27x010 EPROM.

4. Load the monitor program.

   ```
   Data.LOAD.B rom68.bin /ny
   ```

5. Configure the Monitor program.

   ```
   Data.Set 0x400 0x9f              ; select 68020 CPU
   Data.Set 0x402 1                 ; 16bit EPROM
   ```

   At least the CPU type and EPROM size must be specified.

6. Set the polarity of the Reset and NMI signal according to your target.

   ```
   eXception.RESetPOL -
   eXception.NMIPOL -
   eXception.NMIBREAK ON
   ```

7. Start the ROM Monitor.

```
SYStem.Up
```

If the RESET output of the ESI is not connected you have to reset your target after the SYStem.Up command manually.

8. Load the application.

```
Data.LOAD.i mcc.abs
```

The format of the **Data.LOAD** command depends on the file format generated by the compiler. The corresponding options for all available compilers are listed in the compiler list. A detailed description of the **Data.LOAD** command is given in the **"General Commands Reference"**.

A typical start sequence for a ROM monitor (68332, EPROM 0--1ffff, RAM 40000--5ffff) is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

```
B::                                       ; Select the ICD device prompt

WinCLEAR                                  ; Clear all windows

SYStem.CPU M68332                         ; Set CPU for debugger
                                          ; software

Data.LOAD.i mccp.x /nil                   ; Load the application

MAP.ROM 0x0--0x1FFFF                      ; Map the ESI

Data.LOAD rom68.hex /ny                   ; Load the monitor program

; add patch to disable 332 watchdog

Data.Set sp:0x0 %Long 0x41000             ; Initialize the SSP

Data.Set sp:0x4 %Long 0x0ff0              ; Initialize the reset vector

Data.Ass sp:0x0ff0 move.b #40,0x0fffa21   ; Extra code to disable
                                          ; watchdog

Data.Ass, jmp 0x440                       ; Jump to monitor start

; initialize the monitor configuration table

d.s sp:0x400 %byte 10.                    ; Set CPU for monitor program

d.s sp:0x402 %byte 0x0                    ; Set the EPROM bus width

; set the polarity of RESET and NMI

eXception.RESetPOL -                      ; Negative polarity for Reset
eXception.NMIPOL -                          ; Negative polarity for NMI

eXception.RESBREAK ON                       ; Enable Reset activation

; start debugging

SYStem.Up                                   ; Start monitor

Data.LOAD.rof rof 0x4000 0x40000 /col       ; Load application

Register.Set PC main                        ; Set the PC to function main

Register.Set USP 0x0FEFF                    ; Set USP to address FEFF

Register.Set SPP 0x0FFFF                    ; Set SSP to address FFFF

List.Mix                                    ; Open disassembly window *)

Register                                    ; Open register window    *)

Var.Frame /Locals /Caller                   ; Open the stack frame with
                                            ; local variables         *)

Var.Watch <var1> <var2>                     ; Add variables to watch
                                            ; window                  *)

Break.Set 0x10000 /p                        ; Set breakpoint to address
                                            ; 10000
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

Stack Memory (only ROM Monitor)	All 68000-type ROM debuggers need memory in the supervisor stack area (SSP) to break correctly. If you get an invalid PC value after stopping the program, the SSP register may be outside the memory area. This must be considered especially when debugging the startup code of an application. The ROM Monitor needs up to 40 bytes space on the stack for step, go and EPROM modification (Hot Patch). The stack is not required for starting the Monitor and memory read or modify commands. BDM debuggers need no stack.
Register Setup	The SR register trace flag should not be set to 1.

Troubleshooting

No information available.

FAQ

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

ROM Monitor

Monitor Features

The monitor requires no stack during startup and memory operations. A valid stack is only required for modifications in the EPROM while the monitor is running (Hot Patch) and for single step and go commands. This allows to use the monitor for testing not fully functional hardware, as only the EPROM access must work correctly to operate the monitor. The position independent code of the monitor allows to relocate the monitor during debugging. The NMI pin of the EPROM simulator can be used to manually stop the target program. On serial linked ROM monitors the NMI line can be controlled by the RTS or DTR lines.

Monitor Files

The “rom68” monitor is for EPROM simulator solutions, while the “rom68e” monitor is used as foreground monitor for emulators. By using a foreground monitor the target program can be single stepped without stopping the target processors interrupts or DMA transfers.

The monitor “rom68s'”is the serial line monitor. It requires linking with a serial line driver module. See the example files in the monitor directory “…\files\demo\m68k\monitor” for details. All monitors have the same source file “rom68.asm”. This source file should not be modified, it is only included for reference purposes. There are two possibilities to include the monitor in the application: loading the “.bin” by the Eprom Simulator or linking the “.src” file together with the application. The “.src” files contain only the monitor code, a corresponding configuration table has to be included in the target program.

Address Layout

The ROM monitor is freely relocatable in the whole address space. The communication area for the EPROM simulator is located at the fixed address 1000 to 1FFF of the first EPROM. The CPU address depends on the bus width of the EPROMs. The following table shows the address ranges occupied by the communication port:

Bus Width	Start Address	End Address
8 bit	EPROM_BASE+1000	EPROM_BASE+1FFF
16 bit	EPROM_BASE+2000	EPROM_BASE+3FFF
32 bit	EPROM_BASE+4000	EPROM_BASE+7FFF

The monitor program consists of three parts:

- Vector Table
- Configuration Table
- Monitor Program Code

The '.bin' and '.asm' files contain all three parts of the monitor. The address layout of the default monitor is as follows:

0000--03FF	**Vector Table**
0400--041F	Configuration Table
0420--=FFF	Monitor Code

Vector Table

For the first tests of a software, the “.bin” files can be loaded with vector and configuration table. When the vector table becomes part of the application, it is not loaded with the monitor. Instead the table is setup according to the application (the table may also reside in RAM). Some vectors must be set up to point into the monitor program code. The entry points are located at the beginning of the monitor.

vec	offs	ent	Usage
00	000	-	Reset Stack (optional)
01	004	+20	Reset PC (optional, can also go to application)
02	008	+50	Bus Error (optional, when Bus Errors should enter monitor)
03	00C	+50	Address Error (optional, when they should enter monitor)
04	010	+30	Illegal Instruction (used for breakpoints)
09	024	+30	Trace (used for single step)
NMI	XXX	+30	Manual Break (optional)
…	…	+40	Any unused vector may be handled by the monitor

Configuration

The configuration table of the monitor must always be located directly before the monitor code. The default location used in the binary files is 400 (hex).

- Processor core type (byte at offset 00H):

  00 = 6800x, 6830x, 68322, 68356 (default)

  03 = 68010, CPU32

  9F = 68020, 68030 no MMU

  A7 = 68040 no MMU

  A3 = 68060 no MMU

  01 = 68070, 93Cxx

- EPROM Bus Width (byte at offset 02H):

  0 = 8 bit (default)

  1 = 16 bit

  2 = 32 bit

- Monitor Interrupt Level (byte at offset 04H)

  0 = all interrupts enabled in monitor

  …

  7 = all interrupts disabled in monitor (default)

- Relative Monitor Location (long at offset 0CH).

  This is the offset from the start of the EPROM to the monitor configuration table. It is **not** the absolute address of the monitor.

Break without Hardware Interrupt

If no hardware interrupts are free for the implementation of the **Break** command, it is possible to implement a software solution. An interrupt of the target program (usually the timer interrupt) polls the address of the communication area to determine when a break has been entered. If bit zero of the status byte (at location 1400H) is set, the interrupt should enter the monitor through the breakpoint entry point. The stack should be set in the same way as if an NMI has been executed.

```
Clock_Interrupt:

btst #0,$1400                   ; clear hardware bits here, if required

beq Normal_Interrupt

jmp $420+$30                    ; enter monitor if bit 0 is set

Normal_Interrupt:               ; continue with normal interrupt
```

Hardware Breakpoint for MC68360

The built-in hardware breakpoint of the 68360 can be used by the debugger. It can be used to stop the debugger on a read or write to a variable or a fetch in the ROM area. They are set with the regular breakpoint command as read or write breakpoints. Note that the MBAR-Register and the **SYStem.Option.BASE** value must be equal before setting the breakpoints. When a program breakpoint is set in a read-only mapped area, it is automatically converted into a hardware read breakpoint.

The behavior of the hardware breakpoints can be controlled with the **TrOnchip** commands.

Memory Classes

Memory Class	Description
FC0	Function-Code 0
FC1	USER-DATA
UD	USER-DATA
FC2	USER-PROGRAM
UP	USER-PROGRAM
FC3	Function-Code 3
FC4	Function-Code 4
FC5	SUPERVISOR-DATA
SD	SUPERVISOR-DATA
FC6	SUPERVISOR-PROGRAM
SP	SUPERVISOR-PROGRAM
FC7	Function-Code 7
CPU	CPU Function-Code
U	User
S	Supervisor
D	Data
P	Program
C	Memory access by CPU
E	Emulation memory access

SYStem.BdmClock   Select BDM-clock

Format:	**SYStem.BdmClock** <*rate*>
<*rate*>:	**4** | **8** | <*fixed*>
<*fixed*>:	**1000.** … **5000000.**

Either the clock frequency divided by 4 or 8 is used as the BDM clock or a fixed clock rate. The fixed clock rate must be used when the operation frequency is very slow or the clock is turned off or the target clock line is not connected. The default is a fixed rate of 1 MHz.

There is an additional plug on the debug cable on the debugger side. This plug can be used as an external clock input. With setting **EXT/x** the external clock input (divided by **x**) is used as BDM port frequency.

The **ColdFire+/V1** offers two clock sources for the communication between debugger and CPU:

Format:	**SYStem.BdmClock BusClock** | **Async**

BusClock	Time base for BDM communication is the bus frequency of the CPU. This allows a faster download if the data rate is increased by configuring the FLL. On the other hand moving the communication frequency can cause problems, because the debugger has to synchronize again after each change of frequency.
Async	The lower clock of the FLL is fixed time base for BDM communication. The bus frequency can be modified without affecting the BDM channel. This selection requires no resynchronization on bus clock changes.

This command tells the debugger how to configure the CPU when you start your debug session. If you have to download big files you can use “BusClock” to get the highest available bus frequency and as a result the highest download performance.

SYStem.CPU — Select CPU type

Format:	**SYStem.CPU** <*mode*>
<*mode*>:	**000** | **010** | **020** | **030** | **040** | **060** **302** | **LC302** | **PM302** | **EN302** | **356** | **306** | **307** **330** … **68336** | **340** | **341** | **349** | **360**

Selects the processor type.

NOTE:	ROM debuggers require also a modification in the debug monitor for different processor types.

SYStem.LOCK — Lock and tristate the debug port

Format:	**SYStem.LOCK** [**ON** | **OFF**]

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the **SYStem.LOCK** command is to give debug access to another tool.

SYStem.MemAccess — Select run-time memory access method

Format:	**SYStem.MemAccess Enable | StopAndGo | Denied** **SYStem.ACCESS** (deprecated)

Enable **CPU** (deprecated)	Memory access during program execution to target is enabled.
Denied (default)	Memory access during program execution to target is disabled.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

SYStem.Mode — Establish the communication with the CPU

Format:	**SYStem.Mode** *<mode>* **SYStem.Down** (alias for SYStem.Mode Down) **SYStem.Up** (alias for SYStem.Mode Up)
<mode>:	**Down** **NoDebug** **Go** **Up**

Default: Down.

Selects the target operating mode.

“Debug mode is active” means the communication channel via debug port (JTAG) is established. The features of the “on-chip debug support” (OCDS) are enabled and available.

Down	The CPU is in reset. Debug mode is not active. Default state and state after fatal errors.
NoDebug	The CPU is running. Debug mode is not active. Debug port is tristate. In this mode the target should behave as if the debugger is not connected.

Go	The CPU is running. Debug mode is active. After this command the CPU can be stopped with the break command or if any break condition occurs.
Up	The CPU is not in reset but halted. Debug mode is active. In this mode the CPU can be started and stopped. This is the most typical way to activate debugging.
Attach	Not supported.
StandBy	Not supported.

If the mode “Go” is selected, this mode will be entered, but the control button in the SYStem window jumps to the mode “UP”.

The “Emulate” LED on the debug module is ON when the debug mode is active and the CPU is running.

SYStem.CONFIG   Configure debugger according to target topology

Format:	**SYStem.CONFIG** *<parameter> <number_or_address>* **SYStem.MultiCore** *<parameter> <number_or_address>* (deprecated)
<parameter>:	**CORE** *<core>*
<parameter>: (JTAG):	**DRPRE** *<bits>* **DRPOST** *<bits>* **IRPRE** *<bits>* **IRPOST** *<bits>* **TAPState** *<state>* **TCKLevel** *<level>* **TriState** [**ON** | **OFF**] **Slave** [**ON** | **OFF**]

The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. Arm + DSP). The information is required before the debugger can be activated e.g. by a **SYStem.Up**. See **Daisy-chain Example**.
For some CPU selections (**SYStem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

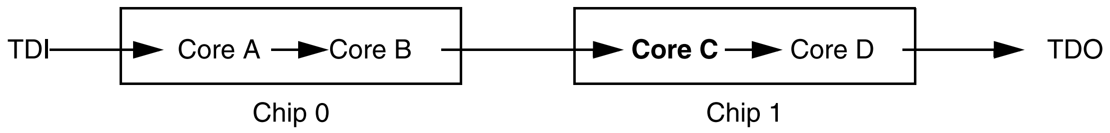
TriState has to be used if several debuggers ("via separate cables") are connected to a common JTAG port at the same time in order to ensure that always only one debugger drives the signal lines. TAPState and TCKLevel define the TAP state and TCK level which is selected when the debugger switches to tristate mode. Please note: nTRST must have a pull-up resistor on the target, TCK can have a pull-up or pull-down resistor, other trigger inputs need to be kept in inactive state.

⚠	Multicore debugging is not supported for the DEBUG INTERFACE (LA-7701).

CORE	For multicore debugging one TRACE32 PowerView GUI has to be started per core. To bundle several cores in one processor as required by the system this command has to be used to define core and processor coordinates within the system topology. Further information can be found in **SYStem.CONFIG.CORE**.
DRPRE	(default: 0) *<number>* of TAPs in the JTAG chain between the core of interest and the TDO signal of the debugger. If each core in the system contributes only one TAP to the JTAG chain, DRPRE is the number of cores between the core of interest and the TDO signal of the debugger.
DRPOST	(default: 0) *<number>* of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.
IRPRE	(default: 0) *<number>* of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.
IRPOST	(default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.
TAPState	(default: 7 = Select-DR-Scan) This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.
TCKLevel	(default: 0) Level of TCK signal when all debuggers are tristated.
TriState	(default: OFF) If several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.

Slave	(default: OFF) If more than one debugger share the same debug port, all except one must have this option active. JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET).

Daisy-Chain Example



Below, configuration for core C.

Instruction register length of

- Core A: 3 bit
- Core B: 5 bit
- Core D: 6 bit

```
SYStem.CONFIG.IRPRE  6.                ; IR Core D

SYStem.CONFIG.IRPOST 8.                ; IR Core A + B

SYStem.CONFIG.DRPRE  1.                ; DR Core D

SYStem.CONFIG.DRPOST 2.                ; DR Core A + B

SYStem.CONFIG.CORE 0. 1.               ; Target Core C is Core 0 in Chip 1
```

0	Exit2-DR
1	Exit1-DR
2	Shift-DR
3	Pause-DR
4	Select-IR-Scan
5	Update-DR
6	Capture-DR
7	Select-DR-Scan
8	Exit2-IR
9	Exit1-IR
10	Shift-IR
11	Pause-IR
12	Run-Test/Idle
13	Update-IR
14	Capture-IR
15	Test-Logic-Reset

Format:	**SYStem.CONFIG.CORE** *<core_index> <chip_index>* **SYStem.MultiCore.CORE** *<core_index> <chip_index>* (deprecated)
<chip_index>:	**1 … i**
<core_index>:	**1 … k**

Default *core_index*: depends on the CPU, usually 1. for generic chips

Default *chip_index*: derived from CORE= parameter of the configuration file (config.t32). The CORE parameter is defined according to the start order of the GUI in T32Start with ascending values.

To provide proper interaction between different parts of the debugger, the systems topology must be mapped to the debugger's topology model. The debugger model abstracts chips and sub cores of these chips. Every GUI must be connect to one unused core entry in the debugger topology model. Once the **SYStem.CPU** is selected, a generic chip or non-generic chip is created at the default *chip_index.*

Non-generic Chips

Non-generic chips have a fixed number of sub cores, each with a fixed CPU type.

Initially, all GUIs are configured with different *chip_index* values. Therefore, you have to assign the *core_index* and the *chip_index* for every core. Usually, the debugger does not need further information to access cores in non-generic chips, once the setup is correct.

Generic Chips

Generic chips can accommodate an arbitrary amount of sub-cores. The debugger still needs information how to connect to the individual cores e.g. by setting the JTAG chain coordinates.

Start-up Process

The debug system must not have an invalid state where a GUI is connected to a wrong core type of a non-generic chip, two GUIs are connected to the same coordinate or a GUI is not connected to a core. The initial state of the system is valid since every new GUI uses a new *chip_index* according to its CORE= parameter of the configuration file (config.t32). If the system contains fewer chips than initially assumed, the chips must be merged by calling **SYStem.CONFIG.CORE**.

SYStem.CONFIG.state Display target configuration

Format:	**SYStem.CONFIG.state** [**/**<*tab*>]
<*tab*>:	**DebugPort** | **Jtag**

Opens the **SYStem.CONFIG.state** window, where you can view and modify most of the target configuration settings. The configuration settings tell the debugger how to communicate with the chip on the target board and how to access the on-chip debug and trace facilities in order to accomplish the debugger's operations.

SYStem.Option.BASE Select peripheral base address

Format:	**SYStem.Option.BASE** <*address*>

Defines the base address of the internal IO of some 683xx processors. This should be set to the value used by the target system.

SYStem.Option.CACHE Flush instruction cache on MC68349

Format:	**SYStem.Option.CACHE** [**ON** | **OFF**]

Flushes instruction cache on MC68349.

SYStem.Option.HOOK Compare PC to hook address

Format:	**SYStem.Option.HOOK** <*address*> | <*address_range*>

The command defines the hook address. After program break the hook address is compared against the program counter value.

If the values are equal, it is supposed that a hook function was executed. This information is used to determine the right break address by the debugger.

Defines the location of the PC after a break in the hook function. The hook function allows to insert a piece of code in the execution of breakpoints. When the option is active (nonzero) the BGND breakpoint command is replaced by an undefined instruction. The undefined instruction handler should then execute the required code and then stop with a BGND instruction. After the code after the BGND instruction is executed with the next Step or Go command. The code in the hook function should not modify the SSP register.

SYStem.Option.ICFLUSH Flush of instruction cache before step and go

Format:	**SYStem.Option.ICFLUSH** [**ON** | **OFF**]

Default: OFF.

If this option is enabled, the instruction cache will be invalidated before debug mode will be left (in case of a **Step** or **Go**).

SYStem.Option.IMASKASM Disable interrupts while single stepping

Format:	**SYStem.Option.IMASKASM** [**ON** | **OFF**]

Default: OFF.

If enabled, the bit responsible for ignoring pending interrupts during assembler single-step operations of the CPU will be set. The interrupt routine is not executed during single-step operations.

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

Format:	**SYStem.Option.IMASKHLL** [**ON** | **OFF**]

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during HLL single-step operations. The interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step.

SYStem.Option.MMUPhysLogMemaccess Memory access preferences

Format:	**SYStem.Option.MMUPhysLogMemaccess** [**ON** | **OFF**]

Default: ON.

Controls whether TRACE32 prefers a cached logical memory access over a (potentially uncached) physical memory access to keep caches updated and coherent.

NOTE:	This option should usually not be changed.

ON	A cached logical memory access is used.
OFF	A (potentially uncached) physical memory access is used.

SYStem.Option.MMUSPACES   Separate address spaces by space IDs

Format:	**SYStem.Option.MMUSPACES** [**ON** | **OFF**] **SYStem.Option.MMUspaces** [**ON** | **OFF**] (deprecated) **SYStem.Option.MMU** [**ON** | **OFF**] (deprecated)

Default: OFF.

Enables the use of space IDs for logical addresses to support **multiple** address spaces.

For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Concepts"** (trace32_concepts.pdf).

NOTE:	**SYStem.Option.MMUSPACES** should not be set to **ON** if only one translation table is used on the target. If a debug session requires space IDs, you must observe the following sequence of steps: 1. Activate **SYStem.Option.MMUSPACES**. 2. Load the symbols with **Data.LOAD**. Otherwise, the internal symbol database of TRACE32 may become inconsistent.

Examples:

```
;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x012A:
Data.dump D:0x012A:0xC00208A

;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x0203:
Data.dump D:0x0203:0xC00208A
```

SYStem.Option.OTE — Ownership trace

Format: **SYStem.Option.OTE** [**ON** | **OFF**]

Enables/disables ASID trace.

SYStem.Option.SLOWRESET — Slow reset enable

Format: **SYStem.Option.SlowReset** [**ON** | **OFF**]

Has to be switched **ON** if the reset line of the debug connector is not(!) connected direct to the CPU reset pin.

Problem: At system-up the debugger has to enable the CPUs debug mode first. This is done by a certain sequence of the debug signals. This sequence becomes faulty if the target includes a reset-circuit which hold the reset line for a unknown period.

If **SlowReset** is switched “ON” the debugger accepts a reset-hold period of up to 1 s. A system up needs about 3 s then!

SYStem.Option.PST — Detect HALT condition of the CPU

Format: **SYStem.Option.PST** [**ON** | **OFF**]

Default: OFF.

Setting this option to **ON** enables the hardware-based check method for the HALT-condition of the CPU. For this setting to work correctly the signals PST[3..0] (V2 and V3 ColdFire cores), ALLPST (some pin-limited CPU's) or PSTDDATA[7..0] have to be properly connected to the debug connector.

If these signals are located on pins sharing some other functions, the PST mode of these CPU pins has to be enabled first. The recommended setting is **ON**, because otherwise the HALT state of the CPU cannot be reliably detected.

SYStem.Option.PSTCLKTERM Termination of the PSTCLK pin

Format:	**SYStem.Option.PSTCLKTERM** [**ON** | **OFF**]

Default: ON.

Turns the termination of the PSTCLK pin on the debug connector **ON**/**OFF**.
Works only with the COLDFIRE-HS whisker.

SYStem.Option.ResetAction Debugger behavior when RESET is detected

Format:	**SYStem.Option.ResetAction** [**NOTHING** | **HALT** | **GO**]

Default: HALT.

This setting changes the behavior of the debugger when a RESET is detected on the target board.

NOTHING	Do nothing at all; the CPU keeps running without the debugger influencing the CPU after a RESET.
HALT	Break immediately after reset.
GO	After setting up basic registers (on-chip breakpoints, etc.), a GO is issued automatically.

SYStem.Option.StandbyAction Debugger behavior when power is restored

Format:	**SYStem.Option.StandbyAction** [**HALT** | **GO**]

Default: GO.

This setting changes the behavior of the debugger when the power is restored on the target board.

HALT	Break immediately after power comes back.
GO	After setting up basic registers (on-chip breakpoints, etc.), a GO is issued automatically.

SYStem.Option.TracePULSE Use PULSE instruction

Format:	**SYStem.Option.TracePULSE** [**ON** | **OFF**]

When set to ON, the PULSE instruction is used to trigger the trace.

SYStem.Option.TraceWDDATA Use WDDATA instruction

Format:	**SYStem.Option.TracePULSE** [**ON** | **OFF**]

When set to ON, the WDDATA instruction is used to turn on or off the trace similar to TraceON/TraceOFF breakpoints (bit 0 of data).

SYStem.RESetOut Reset target without reset of debug port

Format:	**SYStem.RESetOut**

If possible (nRESET is open collector), this command asserts the nRESET line on the debug connector. This will reset the target including the CPU but not the debug port. The function only works when the system is in **SYStem.Mode.Up**.

Format:	**SYStem.Option.StepFlat** [**HALT** | **GO**]

Avoids stepping into TLB miss handlers.

Trace specific SYStem.Option Commands

SYStem.Option.BTB — Change the width of the address information

Format: **SYStem.Option.BTB** [**16** | **24** | **32** | **OFF**]

Default without trace: OFF.
Default with trace: 32.

Changes the width of the address information sent to the trace. The default setting of 32 bits gives the best trace decoding results.

SYStem.Option.DDC — Configure the tracing of data accesses

Format: **SYStem.Option.DDC** [**Write** | **Read** | **ReadWrite** | **OFF**]

Default: OFF.

Configures the tracing of data accesses. Only accesses that leave the cache can be traced.

SYStem.Option.TSYNC — Send the PC to the trace port

Format: **SYStem.Option.TSYNC** [**ON** | **OFF**]

Default: OFF.

Forces the CPU to send the current PC to the trace port every few milliseconds (a) when set to ON and (b) if the CPU supports the SYNC_PC BDM instruction.

Set to ON when you have long-running code-sequences without any indirect branches, which confuse the trace display.

CPU specific TrOnchip Commands

TrOnchip.ALIGN — Enable breakpoint alignment

```
Format:        TrOnchip.ALIGN [ON | OFF]
```

Some 68360 chips cannot set the breakpoint correctly in 8 and 16 bit chip select areas. This option tries to work around this bug by setting the breakpoint to the next quad aligned address.

TrOnchip.RESet — Set on-chip trigger to default state

```
Format:        TrOnchip.RESet
```

Sets the TrOnchip settings and trigger module to the default settings.

TrOnchip.state — Display on-chip trigger window

```
Format:        TrOnchip.state
```

Opens the **TrOnchip.state** window.

TrOnchip.SIZE   Enable break on SIZE lines

Format: **TrOnchip.SIZE** [**ON** | **OFF**]

If activated, the SIZE lines of the processor are also used as a breakpoint criteria. The debugger will only be stopped when the SIZE lines match the breakpoint size. Breakpoint ranges can have a size of 1,2,3 or 4 bytes. Breakpoints on a single address have no size. The following example shows the difference.

```
TrOnchip.SIZE ON

Break.Set 0x1000 /WRITE         ; ignores the SIZE lines

Break.Set 0x1000--0x1000 /W     ; break only on BYTE access

Break.Set 0x1000--0x1003 /W     ; break only on LONG access
```

TrOnchip.TEnable   Set filter for the trace

Format: **TrOnchip.TEnable** <*par*> (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TOFF   Switch the sampling to the trace to OFF

Format: **TrOnchip.TOFF** (deprecated)

Refer to the **Break.Set** command to set trace filters.

TrOnchip.TON   Switch the sampling to the trace to “ON”

Format: **TrOnchip.TON EXT** | **Break** (deprecated)

Refer to the **Break.Set** command to set trace filters.

Format:	**TrOnchip.TTrigger** <*par*> (deprecated)

Refer to the **Break.Set** command to set a trigger for the trace.

MMU.DUMP

Page wise display of MMU translation table

Format:	**MMU.DUMP** *<table>* [*<range>* | *<address>* | *<range> <root>* | *<address> <root>*] **MMU.***<table>***.dump** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*

Displays the contents of the CPU specific MMU translation table.

- If called without parameters, the complete table will be displayed.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: displays the translation table of the specified process • else, this command displays the table the CPU currently uses for MMU translation.

KernelPageTable	Displays the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>*:**0x0**	Displays the MMU translation table entries of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

ITLB	Displays the contents of the ITLB translation table. Deprecated command syntax: MMU.ITLB.
DTLB	Displays the contents of the DTLB translation table. Deprecated command syntax: MMU.DTLB.

MMU.List — Compact display of MMU translation table

Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.List** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**

Lists the address translation of the CPU-specific MMU table.

- If called without address or range parameters, the complete table will be displayed.
- If called without a table specifier, this command shows the debugger-internal translation table. See **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: list the translation table of the specified process • else, this command lists the table the CPU currently uses for MMU translation.

KernelPageTable	Lists the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and lists its address translation.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>*:**0x0**	Lists the MMU translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and lists its address translation. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

Format:	**MMU.SCAN** *<table>* [*<range>* *<address>*] **MMU.***<table>***.SCAN** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** [**Clear**] *<cpu_specific_tables>*

Loads the CPU-specific MMU translation table from the CPU to the debugger-internal static translation table.

- If called without parameters, the complete page table will be loaded. The list of static address translations can be viewed with **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, page table entries will only be loaded if their **logical** address matches with the given parameter.

Use this command to make the translation information available for the debugger even when the program execution is running and the debugger has no access to the page tables and TLBs. This is required for the real-time memory access. Use the command **TRANSlation.ON** to enable the debugger-internal MMU table.

PageTable	Loads the entries of an MMU translation table and copies the address translation into the debugger-internal static translation table. • if *<range>* or *<address>* have a space ID: loads the translation table of the specified process • else, this command loads the table the CPU currently uses for MMU translation.

KernelPageTable	Loads the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Loads the MMU address translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and copies its address translation into the debugger-internal static translation table. • For information about the first three parameters, see **“What to know about the Task Parameters”** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manual**.
ALL [**Clear**]	Loads all known MMU address translations. This command reads the OS kernel MMU table and the MMU tables of all processes and copies the complete address translation into the debugger-internal static translation table. See also the appropriate **OS Awareness Manual**. **Clear:** This option allows to clear the static translations list before reading it from all page translation tables.

CPU specific tables:

-- No CPU specific tables --

BDM Connector 68K

Signal	Pin	Pin	Signal
DS-	1	2	BERR-
GND	3	4	BKPT-
GND	5	6	FREEZE
RESET-	7	8	DSI (IFETCH-)
VCCS	9	10	DSO (IPIPE-)

BDM and Trace Connector ColdFire

BDM Connectors for ColdFire V1, V2, V3, V4 and ColdFire+

- The signals needed for debugging and tracing are combined in a single connector.
- Some of the pins in the schematics below have multiple alternate signal names. Please check the "BDM Connector Pinout" chapter of the reference manual for the exact CPU you are using for further details or contact our support team.

BDM Connector 6 pin ColdFire+/V1 CPUs Debugger

Signal	Pin	Pin	Signal
BKGD	1	2	GND
N/C	3	4	RESET-
N/C	5	6	VCC

BDM Connector 26 pin ColdFire V2 or V3 CPUs Debugger and Trace

Signal	Pin	Pin	Signal
N/C	1	2	BKPT-
GND	3	4	DSCLK
GND	5	6	N/C
RESET-	7	8	DSI
VDD_IO 1.8...5.0 V	9	10	DSO
GND	11	12	PST3
PST2	13	14	PST1
PST0	15	16	DDATA3
DDATA2	17	18	DDATA1
DDATA0	19	20	GND
N/C	21	22	N/C
GND	23	24	PSTCLK/CLKOUT/CPUCLK
VDD_CPU 1.8-5.0V	25	26	TEA-/TA-/DTACK-/BERR-

If the tracing capability is not needed, DDATA3..0 can be connected or pulled down to GND.

BDM Connector 26 pin ColdFire V2 or V3 CPUs with ALLPST only Debugger

Signal	Pin	Pin	Signal
N/C	1	2	BKPT-
GND	3	4	DSCLK
GND	5	6	N/C
RESET-	7	8	DSI
VDD_IO 1.8-5.0V	9	10	DSO
GND	11	12	ALLPST
ALLPST or pull-up resistor	13	14	ALLPST or pull-up resistor
ALLPST or pull-up resistor	15	16	GND or pull-down resistor
GND or pull-down resistor	17	18	GND or pull-down resistor
GND or pull-down resistor	19	20	GND
N/C	21	22	N/C
GND	23	24	PSTCLK/CLKOUT/CPUCLK
VDD_CPU 1.8...5.0 V	25	26	TEA-/TA-/DTACK-/BERR-

This connector cannot be used for tracing, because the CPU variants with the ALLPST signal miss the necessary PST3..0 and DDATA3..0 signals.

Signal	Pin	Pin	Signal
N/C	1	2	BKPT-
GND	3	4	DSCLK
GND	5	6	N/C
RESET-	7	8	DSI
VDD_IO 1.8...5.0 V	9	10	DSO
GND	11	12	PSTDDATA7
PSTDDATA6	13	14	PSTDDATA5
PSTDDATA4	15	16	PSTDDATA3
PSTDDATA2	17	18	PSTDDATA1
PSTDDATA0	19	20	GND
N/C	21	22	N/C
GND	23	24	PSTCLK
VDD_CPU 1.8-5.0V	25	26	TA-

Technical Data BDM 68K

Operation Voltage

Adapter	OrderNo	Voltage Range
BDM Debugger for 68K (ICD)	LA-7710	3.0 .. 5.5 V

Technical Data BDM ColdFire

Operation Voltage

Adapter	OrderNo	Voltage Range
BDM Debugger for ColdFire+/V1 (ICD)	LA-3746	3.0 .. 5.5 V
BDM Debugger for ColdFire HS (ICD)	LA-3757	3.0 .. 5.5 V

Technical Data Trace ColdFire

Operation Voltage

Adapter	OrderNo	Voltage Range
Preprocessor for ColdFire family HS	LA-3759	3.0 .. 5.5 V