LAUTERBACH
DEVELOPMENT TOOLS

General Commands Reference Guide M

Release 02.2024

MANUAL

General Commands Reference Guide M

General Commands Reference Guide M

Version 04-Mar-2024

History

27-Nov-2023	Command MCDS.TimeStamp has been marked as deprecated and replaced by the new command MCDS.TImeMode.
27-Nov-2023	New command MCDS.Option.RESetBehavior.
27-Nov-2023	Command MCDS.Option.TTRESet has been removed.
23-Aug-2023	New command MCDS.TargetAgents.CLEAR.
09-Aug-2022	New command MAP.BUS64.
05-Aug-2022	For the MMU.SCAN ALL command, CLEAR is now possible as an optional second parameter.
11-Apr-2022	Commands MCDS.BusTrace.Agent, MCDS.BusTrace.Mode, MCDS.DataTrace.Agent, MCDS.DataTrace.Mode, MCDS.PERipheralTrace, MCDS.ProgramTrace.Agent, and MCDS.ProgramTrace.Mode marked as Modern.
11-Apr-2022	Command MCDS.Source.Set marked as Classic.
21-Feb-2022	New commands: MCDS.BusTrace.Agent, MCDS.BusTrace.Mode, MCDS.DataTrace.Agent, MCDS.DataTrace.Mode, MCDS.PERipheralTrace, MCDS.ProgramTrace.Agent, and MCDS.ProgramTrace.Mode.
21-Feb-2022	Marked the old description of MCDS.state as deprecated and added new description for MCDS.state.

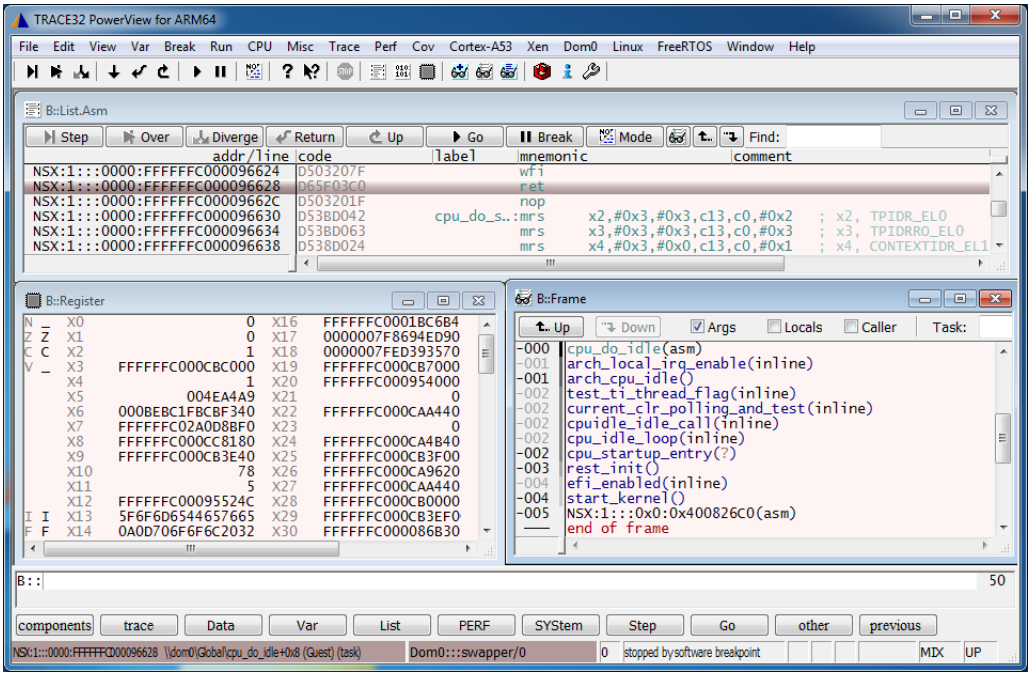
MACHINE.select Display context of specified virtual machine

Format: **MACHINE.select** *<machine_number>* | "*<machine_name>*" [*<vcpu_number>*]

If the specified virtual machine is active, the currently selected core is changed to the core running the specified machine. As a result, the debugger view is changed to this core and all TRACE32 commands without the **/CORE** *<number>* option apply to it.

If the specified machine is inactive, TRACE32 reads the context of the first VCPU assigned to the machine. If a *<vcpu_number>* is specified, the context of this VCPU is read.

The TRACE32 state line changes to a reddish look-and-feel (see screenshot below) to indicate that the context of an inactive virtual machine is displayed. TRACE32 window commands such as **List.auto**, **Register.view**, **Frame.view** or **Var.Local** apply to this machine. Whereas all other commands switch back to the currently running machine before they are executed.



See also

- Mode
- CORE.select

MAP

Mapping memory attributes

See also

- MAP.ADelay
- MAP.BE
- MAP.BOnchip
- MAP.BUS16
- MAP.BUS24
- MAP.BUS32
- MAP.BUS64
- MAP.BUS8
- MAP.BUS<x>
- MAP.BYTE
- MAP.CacheInhibit
- MAP.COMSTART
- MAP.CONST
- MAP.DenyAccess
- MAP.DenyBurst
- MAP.DMUX
- MAP.FRAG
- MAP.GAP
- MAP.InitVar
- MAP.LE
- MAP.List
- MAP.MONITOR
- MAP.NoBE
- MAP.NoBOnchip
- MAP.NoCacheInhibit
- MAP.NoCONST
- MAP.NoDenyAccess
- MAP.NoDenyBurst
- MAP.NoDMUX
- MAP.NOFRAG
- MAP.NOGAP
- MAP.NoInitVar
- MAP.NoLE
- MAP.NoOPFetch
- MAP.NOPAGE
- MAP.NOROM
- MAP.NOSWAP
- MAP.NoUpdateOnce
- MAP.NoVMREAD
- MAP.NoVOLATILE
- MAP.OPFetch
- MAP.PAGE
- MAP.RESet
- MAP.ROM
- MAP.state
- MAP.SWAP
- MAP.UpdateOnce
- MAP.VMREAD
- MAP.VOLATILE
- MAP.WORD

▲ 'MAP Functions' in 'General Function Reference'

Overview MAP

Mapping the EPROM Simulator for BDM/ROM

The monitor/EPROM-simulator can support two 8-bit or one 16-bit EPROM. The combination of several modules allows 32- and 64-bit configuration to be supported.

During the simulation the EPROM configuration of the target system is imitated by software. Using this technique paged and banked EPROM's can be simulated.

The imitation of the EPROM configuration is done as follows:

1. Reset the mapping system (**MAP.RESet** command).
2. Map the EPROM simulator within the specified range (**MAP.ROM** command).
3. Set the EPROM bus size (**MAP.BUSXX** command). The default bus size is 8 bit.

4. Set the EPROM width (**MAP.BYTE** or **MAP.WORD** command). By default an 8 bit organized EPROM is assumed.

```
;-------------------------------------------------
; maps one 8K x 8 EPROM
; 8 bit adapter low
b:
MAP.RESet
MAP.ROM 0x0--0x01fff

;-------------------------------------------------------------
; maps two 8K x 8 EPROMS in parallel
; 8 bit adapter low and high
b:
MAP.RESet
MAP.ROM 0x0--0x03fff
MAP.BUS16 0x0--0x03fff

;-------------------------------------------------------------
; maps one 4K x 16 EPROM
; 16 bit adapter
b:
MAP.RESet
MAP.ROM 0x0--0x01fff
MAP.BUS16 0x0--0x01fff
MAP.WORD 0x0--0x01fff
```

```
;-------------------------------------------------------------
; maps one paged addressed EPROM with 4 pages (4 x 16K x 8)
; 8 bit adapter low
b:
MAP.RESet

MAP.ROM 0x00000--0x03fff
MAP.ROM 0x04000--0x07fff
MAP.ROM 0x08000--0x0bfff
MAP.ROM 0x0c000--0x0ffff

MAP.PAGE 0 0x00000--0x03fff
MAP.PAGE 1 0x04000--0x07fff
MAP.PAGE 2 0x08000--0x0bfff
MAP.PAGE 3 0x0c000--0x0ffff

;-------------------------------------------------------------
; maps two fragments in one 8 bit EPROM
; 8 bit adapter low
b:
MAP.ROM 0x0--0x7fff
MAP.ROM 0x10000--0x17fff
MAP.FRAG 1 0 0x0--0x7fff
MAP.FRAG 1 8000 0x10000--0x17fff
```

```
;--------------------------------------------------------------------
; relocates one 128K x 8 EPROM mapped from 0x0--0x1ffff to 0x40000
; while the system is up
b:
MAP.RELOCate 0x40000 0x0--0x1ffff

;--------------------------------------------------------------------
; maps four 64K x 8 EPROMs for a bus size of 32 bit
; two EPROM simulators
; for each 8 bit adapter high and low
MAP.ROM 0x0--0x3ffff
MAP.BUS32 0x0--0x3ffff

;--------------------------------------------------------------------
; maps two 64K x 16 EPROMs for a bus size of 32 bit
; two EPROM simulators
; for each 16 bit adapter
MAP.ROM 0x0--0x3ffff
MAP.BUS32 0x0--0x3ffff
MAP.WORD
```

MAP.ADelay — Set analyzer delay

Format: **MAP.ADelay** *<delay>*

The command defines for RISC traces (PPC 500/800) the difference between clock strobe and the corresponding valid bus trace record moment.

See also

- MAP
- MAP.state

MAP.BE — Define big endian area

Format: **MAP.BE** [*<range>*]

Defines the memory address area where the variable value display is switched to big endian word memory interpretation.

See also

- MAP
- MAP.LE
- MAP.state

Format: **MAP.BOnchip** *<addressrange>*

This definition will be used for setting breakpoints. Any breakpoints that touch the defined area will be implemented using on-chip resources. This allows program breakpoints in read only memories or data breakpoints that also consider CPU internal operations. The capabilities of the on-chip breakpoints are CPU dependent.

See also

- MAP
- MAP.state

The command **MAP.BUSx** constrains the debugger to read/write data in the specified access width.

```
MAP.BUS8 0x0++1FFFFF                    ; constrain the debugger to 8-bit
                                        ; reads/writes

Data.dump 0x1000 /Long                  ; display a memory dump in 32-bit
                                        ; format

                                        ; the debugger reads the required
                                        ; information 8-bit wise

Data.Set 0x300 %Long 0xAAAAAAAA         ; write 32-bit data to memory

                                        ; the debugger writes the data
                                        ; 8-bit wise to the memory
```

```
MAP.BUS32 0x060000000++1FFFF            ; constrain the debugger to 32-bit
                                        ; reads/writes

Data.dump 0x06000003C /Byte             ; display a memory dump in 8-bit
                                        ; format

                                        ; the debugger reads the required
                                        ; information 32-bit wise

Data.Set 0x0600007A9 %Byte 0xAA         ; write 8-bit data to memory

                                        ; the debugger reads the relevant
                                        ; data 32-bit wise, modifies the
                                        ; byte and write the data back to
                                        ; memory 32-bit wise
```

For debuggers of some processor architectures, the command **MAP.BUS32** doesn't affect the access width for patching software breakpoint codes. In these cases, the option **SYStem.Option.SOFTLONG** is still required to patch the breakpoint code 32-bit wise.

See also

- MAP.BUS16
- MAP.BUS24
- MAP.BUS32
- MAP.BUS64
- MAP.BUS8
- MAP
- MAP.state

MAP.BUS8 Bus width mapping

Format: **MAP.BUS8** [*<addressrange>*]

Constrains TRACE32 to 8-bit reads/writes for the specified target memory block.

See also

■ MAP.BUS<x> ■ MAP ■ MAP.state

▲ 'Release Information' in 'Legacy Release History'

MAP.BUS16 Bus width mapping

Format: **MAP.BUS16** [*<addressrange>*]

Constrains TRACE32 to 16-bit reads/writes for the specified target memory block.

See also

■ MAP.BUS<x> ■ MAP ■ MAP.state

▲ 'Release Information' in 'Legacy Release History'

MAP.BUS24 Bus width mapping

Format: **MAP.BUS24** [*<addressrange>*]

Constrains TRACE32 to 24-bit reads/writes for the specified target memory block.

See also

■ MAP.BUS<x> ■ MAP ■ MAP.state

MAP.BUS32 Bus width mapping

Format: **MAP.BUS32** [*<addressrange>*]

Constrains TRACE32 to 32-bit reads/writes for the specified target memory block.

See also

■ MAP.BUS<x> ■ MAP ■ MAP.state

▲ 'Release Information' in 'Legacy Release History'

MAP.BUS64 Bus width mapping

Format: **MAP.BUS64** [*<addressrange>*]

Constrains TRACE32 to 64-bit reads/writes for the specified target memory block.

See also

■ MAP.BUS<x> ■ MAP ■ MAP.state

MAP.BYTE Set EPROM width

Format: **MAP.BYTE** [*<addressrange>*]

The EPROM is organized by 8 bits per word in the specified range.

See also

■ MAP ■ MAP.state ■ MAP.WORD

Format: **MAP.CacheInhibit** [*<addressrange>*]

Disable CTS cache simulation for selected address range. This setup has only an effect if MMU architecture **NONE** has been selected (**CTS.CACHE.MMUArchitecture**).

Refer to **CTS.CACHE** for more information.

See also

- MAP
- MAP.state

MAP.COMSTART Offset for ROM monitor

Format: **MAP.COMSTART** *<offset>*

If the ROM Monitor is not located at the begin of the EPROM, this command defines the communication area of monitor.

See also

- MAP
- MAP.state

Format: **MAP.CONST** [*<range>*]

The defined address range contains constants. The address range for the constants can be declared in two ways:

- The compiler provides a constant section

```
MAP.CONST                          ; map the section \.sdata2 as
sYmbol.SECRANGE(\.sdata2)          ; address range for constants
```

- The constants are merged into the code

```
MAP.CONST 0x0--0x3ffff             ;map the address range of a FLASH
                                   ;as address range for constants
```

This command is closely related to the command **CTS.UseConst**.

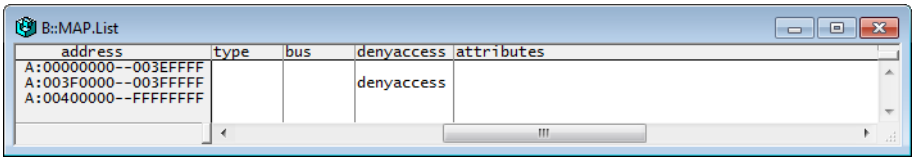
See also

- MAP
- MAP.NoCONST
- MAP.state

Format: **MAP.DenyAccess** [*<physical_addressrange>*]

The TRACE32 software can't access the specified target address range. This command can be used if accesses by the development tool to specific target memory address ranges cause problems (e.g. "debug port fail").

The address ranges that cannot be accessed by the TRACE32 software can be displayed by the **MAP.List** command.



MAP.DenyAccess is switched off by the command **MAP.NoDenyAccess**.

NOTE: Use **MAP.DenyAccess** to protect physical address ranges from debugger access.
Use **TRANSlation.Protect.ADD** to protect logical address ranges from debugger access.

See also

- MAP
- MAP.state
- TRANSlation.Protect.ADD

Format: **MAP.DenyBurst** [*<address>* | *<addressrange>*]

MPC7441, MPC7445, MPC7447, MPC7447A, MPC7450, MPC7451, MPC7455 and MPC7457 only, because for these processors, memory is access is performed through 32-byte burst access per default.

In order to access peripherals which do not support burst accesses (e.g. register and flash address space of MV64xxx), use this command to prevent burst accesses. Please note that the non-burst memory access on these processors is very slow.

MAP.DenyBurst is switched off by the command **MAP.NoDenyBurst**.

See also

- MAP
- MAP.state

MAP.DMUX Define DRAM area

Format: **MAP.DMUX** [*<range>*]

Defines DRAM area. PowerPC only.

See also

- MAP
- MAP.NoDMUX
- MAP.state

MAP.FRAG Form fragment

Format: **MAP.FRAG** *<frag>* *<address>* [*<addressrange>*]

<frag>: **1.** … **255.**

Combines two ROM areas to a fragment. One fragment can be simulated by one EPROM simulator.

Example:

```
MAP.ROM 0x0--0x7fff
MAP.ROM 0x10000--0x17fff

MAP.FRAG 1 0 0x0--0xfff
MAP.FRAG 1 0x8000 0x10000--0x17fff
```

See also

■ MAP ■ MAP.NOFRAG ■ MAP.state

MAP.GAP Define gap

Format: **MAP.GAP** *<frag>* *<address>* [*<addressrange>*]

<frag>: **1.** … **255.**

Defines a gap in the ROM area simulated by an EPROM simulator. This could be useful e.g. if this area is used by internal peripherals

Example:

```
MAP.FRAG 1. 0 0x0--0x7aff
MAP.GAP  1. 0x7b00-0xfff
MAP.FRAG 1. 0x8000 0x8000--0x8fff
```

See also

■ MAP ■ MAP.NOGAP ■ MAP.state

MAP.InitVar CTS initial variable mapping

Format: **MAP.InitVar** [*<addressrange>*]

Maps the selected data address as “read” for **CTS**. This can prevent in CTS a "read-before-write" error from being reported by **Go.TillViolation** when the variable was initialized by the startup code (outside the trace).

See also

■ MAP ■ MAP.state

▲ ’Release Information’ in ’Legacy Release History’

Format: **MAP.LE** [*<range>*]
MAP.LittleEndian [*<range>*] (deprecated)

Defines the memory address area where the variable value display is switched to little endian word memory interpretation.

See also

- MAP
- MAP.BE
- MAP.state

MAP.List List allocation

Format: **MAP.List** [*<address>* | *<addressrange>*] [**/***<option>* …]

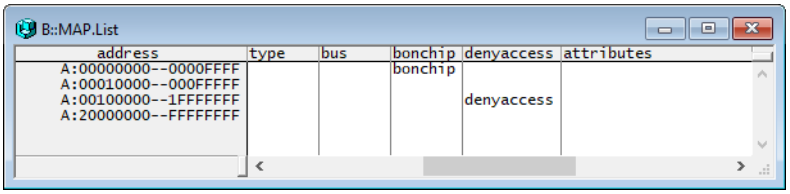
<option>:
DEFault
ALL
Break
Data
Flag
Ack
BANK
BURST
BUS8
Cache
DMA
Internal
NoCache
Onchip
OPFetch
Protect
Wait

The display of the mapper configuration for the logical workspace. The information can be reduced by the options. With no option set everything is displayed, except the mapping of the breakpoint memory (option DEFault).

DEFault	All information, except the breakpoint memory, is displayed.
ALL	All information is displayed.
Data, **Break**, **Flag**	Only the selected memory type is displayed.

Ack	The areas in which the bus cycles would be acknowledged are displayed.
BANK	The mapping of banked and non-banked memory is displayed.
BUS8, **Onchip**, **DMA**, **OPFetch**, …	These are special mapper flags solely used for special emulation probes. For further information, refer to your **Processor Architecture Manual**.
Intern	The internal/external mapping is displayed.
Protect	The memory-write protection is displayed.
Wait	The number of wait states is displayed.

MAP.List Window for BDM/ROM



See also

■ MAP ■ MAP.state ❑ MAP.ROMSIZE()

MAP.MONITOR MONITOR address range

Format: **MAP.MONITOR** [*<range>*]

The defined monitor address range will be excluded from an analyzer trace.

See also

■ MAP ■ MAP.state

MAP.NoBE Switch off big endian

Format: **MAP.NoBE** [*<range>*]

Undoes the settings made by using the **MAP.BE** command.

See also

- MAP
- MAP.state

MAP.NoBOnchip Use on-chip breakpoints

Format: **MAP.NoBOnchip** [*<range>*]

Undoes the settings made by using the **MAP.BOnchip** command.

See also

- MAP
- MAP.state

MAP.NoCacheInhibit CTS cache simulation

ARM

Format: **MAP.CacheInhibit** [*<addressrange>*]

Undoes the settings made by using the **MAP.CacheInhibit** command.

See also

- MAP
- MAP.state

Format: **MAP.NoCONST** [*<range>*]

Undoes the settings made by using the **MAP.CONST** command.

See also

- MAP
- MAP.CONST
- MAP.state

MAP.NoDenyAccess Switch off deny access for TRACE32

Format: **MAP.NoDenyAccess** [*<addressrange>*]

Switches off the deny access for the TRACE32 software that was specified for a specific target memory range.

See also

- MAP
- MAP.state

MAP.NoDenyBurst Undo MAP.DENYBURST settings

Format: **MAP.NoDenyBurst** [*<address>* | *<addressrange>*]

Undoes the settings made by using the **MAP.DenyBurst** command.

See also

- MAP
- MAP.state

Format: **MAP.NoDMUX** [*<range>*]

Undoes the settings made by using the **MAP.DMUX** command.

See also

- MAP
- MAP.DMUX
- MAP.state

MAP.NOFRAG Switch off fragmentation

Format: **MAP.NOFRAG** [*<addressrange>*]

By this command the fragmentation in the specified range is switched off.

See also

- MAP
- MAP.FRAG
- MAP.state

MAP.NOGAP Switch off gap

Format: **MAP.NOGAP** [*<addressrange>*]

By this command the gap in the specified range is switched off.

See also

- MAP
- MAP.GAP
- MAP.state

Format: **MAP.InitVar** [*<addressrange>*]

Undoes the settings made by using the **MAP.InitVar** command.

See also

■ MAP ■ MAP.state

▲ 'Release Information' in 'Legacy Release History'

MAP.NoLE Switch off little endian

Format: **MAP.NoLE** [*<range>*]
MAP.NoLittleEndian [*<range>*] (deprecated)

Undoes the settings made by using the **MAP.LittleEndian** command.

See also

■ MAP ■ MAP.state

MAP.NoOPFetch Switch off opfetch area mapping

Format: **MAP.NoOPFetch** [*<addressrange>*]

Undoes the settings made by using the **MAP.OPFetch** command.

See also

■ MAP ■ MAP.state

Format: **MAP.NOPAGE** *<addressrange>*

By this command the page in the specified range is switched off.

See also

- MAP
- MAP.PAGE
- MAP.state

MAP.NOROM Unmap ESI

Format: **MAP.NOROM** [*<addressrange>*]

By this command the EPROM simulator is unmapped in the specified range.

See also

- MAP
- MAP.ROM
- MAP.state

MAP.NOSWAP Keep byte order

Format: **MAP.NOSWAP** *<addressrange>*

No changes are made to the byte order determined by the target CPU.

See also

- MAP
- MAP.state
- MAP.SWAP

Format: **MAP.NoUpdateOnce** [*<address>* | *<addressrange>*]

Undoes the settings of the **MAP.UpdateOnce** command.

See also

- MAP
- MAP.state

MAP.NoVMREAD Undo MAP.VMREAD settings

Format: **MAP.NoVMREAD** [*<addressrange>*]

Undoes the setting of the **MAP.VMREAD** command.

See also

- MAP
- MAP.state
- MAP.VMREAD

MAP.NoVOLATILE Undo MAP.VOLATILE settings

Format: **MAP.NoVOLATILE** [*<range>*]

Undoes the settings of the **MAP.VOLATILE** command.

See also

- MAP
- MAP.state
- MAP.VOLATILE

Format: **MAP.OPFetch** [*<addressrange>*]

See also

- MAP
- MAP.state

MAP.PAGE Define pages

Format: **MAP.PAGE** *<page_number> <addressrange>*

<page_number>: **1.** … **256.**

This command is used for paged EPROMs. The appropriate page is set by the value in the page register of the EPROM.

Example:

```
; maps one paged addressed EPROM with 4 pages (4 x 16K x 8)
; 8 bit adapter low
b:
MAP.RESet

MAP.ROM 0x00000--0x03fff
MAP.ROM 0x04000--0x07fff
MAP.ROM 0x08000--0x0bfff
MAP.ROM 0x0c000--0x0ffff

MAP.PAGE 0. 0x00000--0x03fff
MAP.PAGE 1. 0x04000--0x07fff
MAP.PAGE 2. 0x08000--0x0bfff
MAP.PAGE 3. 0x0c000--0x0ffff
```

See also

- MAP
- MAP.NOPAGE
- MAP.state

Format: **MAP.RESet**

The mapping system is reset to its default state. The EPROM simulator is switched off.

See also

- MAP
- MAP.state

MAP.ROM Map ESI

Format: **MAP.ROM** *<addressrange>*

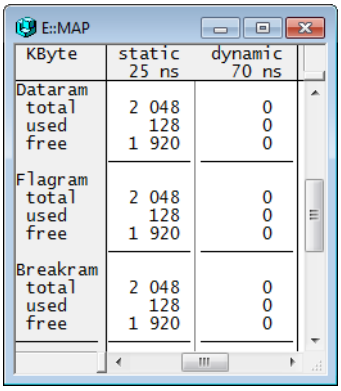
The EPROM Simulator is mapped within the specified range.

See also

- MAP
- MAP.NOROM
- MAP.state
- MAP.ROMSIZE()

Format: **MAP.state**

General view of the total available and the actually activated memory.



See also

- MAP
- MAP.ADelay
- MAP.BE
- MAP.BOnchip
- MAP.BUS16
- MAP.BUS24
- MAP.BUS32
- MAP.BUS64
- MAP.BUS8
- MAP.BUS<x>
- MAP.BYTE
- MAP.CacheInhibit
- MAP.COMSTART
- MAP.CONST
- MAP.DenyAccess
- MAP.DenyBurst
- MAP.DMUX
- MAP.FRAG
- MAP.GAP
- MAP.InitVar
- MAP.LE
- MAP.List
- MAP.MONITOR
- MAP.NoBE
- MAP.NoBOnchip
- MAP.NoCacheInhibit
- MAP.NoCONST
- MAP.NoDenyAccess
- MAP.NoDenyBurst
- MAP.NoDMUX
- MAP.NOFRAG
- MAP.NOGAP
- MAP.NoInitVar
- MAP.NoLE
- MAP.NoOPFetch
- MAP.NOPAGE
- MAP.NOROM
- MAP.NOSWAP
- MAP.NoUpdateOnce
- MAP.NoVMREAD
- MAP.NoVOLATILE
- MAP.OPFetch
- MAP.PAGE
- MAP.RESet
- MAP.ROM
- MAP.SWAP
- MAP.UpdateOnce
- MAP.VMREAD
- MAP.VOLATILE
- MAP.WORD

MAP.SWAP Change byte order

Format: **MAP.SWAP**

Changes the byte order from little endian to big endian or vice versa depending on the target CPU.

See also

- MAP
- MAP.NOSWAP
- MAP.state

Format: **MAP.UpdateOnce** [*<address>* | *<addressrange>*]

Configures the debugger to limit accesses to the specified memory address range to a single access per address. The debugger will store the accessed data in an internal buffer and will use the buffered data for all following accesses.

The data in the internal buffer will be discarded every time the CPU stops for the debugger, e.g. after single step, hitting a breakpoint or a manual break. Discarding the buffered data can be enforced by calling **Data.UPDATE**.

See also

- MAP
- MAP.state

Format: **MAP.VMREAD** *<addressrange>*

Redirects memory reads in the given address range to the TRACE32 virtual memory (VM:).

MAP.VMREAD can be used when target memory cannot be accessed (but is constant) or to avoid "unnecessary" memory reads to constant memory in order to speed up debugging. Usually used for program addresses.

If an address within the VMREAD address space is written by the debugger (e.g. **Data.Set**, **Data.LOAD**, **Data.Assemble** etc., but not using access class VM:), then the debugger will write the data to the target and update TRACE32 virtual memory accordingly.

See also

■ MAP ■ MAP.NoVMREAD ■ MAP.state

▲ 'Release Information' in 'Legacy Release History'

MAP.VOLATILE Mapped address range is volatile

Format: **MAP.VOLATILE** [*<range>*]

The defined memory range is not only changed by the processor core, this means that not all changes to this memory range are sampled to the trace buffer. E.g.: peripherals, dual-ported memory etc. Due to this attribute this memory range can not be used for CTS.

See also

■ MAP ■ MAP.NoVOLATILE ■ MAP.state ■ CTS.UseFinalMemory

MAP.WORD Set EPROM width

Format: **MAP.WORD** *<addressrange>*

The EPROM is organized by 16 bits per word in the specified range.

See also

■ MAP ■ MAP.BYTE ■ MAP.state

MCDS

Multicore debug solution

Overview

The MCDS (MultiCore Debug Solution) is an on-chip trigger and trace solution from Infineon, available for the Infineon TriCore, PCP, GTM, XC2000 and C166 architectures.

There are two basic MCDS concepts:

- The regular MCDS is part of the Emulation Extension Chip (EEC). It supports a big feature set for trace and trigger of the CPUs and buses.
- The miniMCDS is part of the Product Chip (PC). It has a reduced feature set only, e.g. only one core is traceable and no buses. Also the trace buffer size is limited.

Classic vs Modern Commands

With the TRACE32 Release 09/2022 the following commands were introduced to control the generation of trace data for the AURIX TC2xx and subsequent TriCore core generations:

MCDS.BusTrace.Agents	Set bus trace agents.
MCDS.BusTrace.Mode	Set bus trace mode.
MCDS.DataTrace.Agents	Set data trace agents.
MCDS.DataTrace.Mode	Set data trace mode.
MCDS.PERipheralTrace	Control peripheral trace.
MCDS.ProgramTrace.Agents	Set program trace agents.
MCDS.ProgramTrace.Mode	Set program trace mode.
MCDS.TraceAgents.CLEAR	Clear all trace (program, data, bus, and peripheral) agents.

These commands tagged as **Modern** replace the commands of the **MCDS.SOURCE** command group. The replaced commands are therefore marked as **Classic**.

Scripts created for the AURIX TC2xx/TC3xx can continue to be used in the long term. For the AURIX TC4x and subsequent TriCore core generations the new commands have to be used. A 1:1 conversion of the classic to the modern syntax is not possible, because the modern syntax abstracts stronger from the physical MCDS resources.

Further Documentation

The **MCDS** commands described in this document are for reference only, so please refer to the:

- **"MCDS User's Guide"** (mcds_user.pdf) for understanding the MCDS concept and the TRACE32 MCDS support
- **"AURIX Trace Training"** (training_aurix_trace.pdf) for examples of how to use the MCDS on TriCore AURIX (classic MCDS commands only)
- The Infineon documentation for devices specific information

See also

- MCDS.BusTrace.Agents
- MCDS.BusTrace.Mode
- MCDS.CLEAR
- MCDS.CLOCK
- MCDS.DataTrace.Agents
- MCDS.DataTrace.Mode
- MCDS.INFO
- MCDS.Init
- MCDS.OFF
- MCDS.ON
- MCDS.Option
- MCDS.PERipheralTrace
- MCDS.PortSIZE
- MCDS.PortSPEED
- MCDS.ProgramTrace.Agents
- MCDS.ProgramTrace.Mode
- MCDS.Register
- MCDS.RESet
- MCDS.RM
- MCDS.SessionKEY
- MCDS.Set
- MCDS.SOURCE
- MCDS.state
- MCDS.TImeMode
- MCDS.TraceBuffer
- MCDS.MODULE.NAME()
- MCDS.MODULE.TYPE()
- MCDS.STATE()

- 'MCDS Functions' in 'General Function Reference'
- 'Introduction' in 'MCDS User's Guide'

MCDS.BusTrace.Agents — Set bus trace agents

Format:	**MCDS.BusTrace.Agents** [{*<agent>*}]
<agent>:	**SPB** | **LMU0** | **OLDA** | **SPU0** | **DMA** | **SPU1**

Set a list of bus trace agents.

<agent>	The available agents depend on the selected CPU. An empty list will clear all agents.

```
MCDS.BusTrace.Agents SPU0 SPU1
```

See also

- MCDS.BusTrace.Mode
- MCDS.DataTrace.Agents
- MCDS.PERipheralTrace
- MCDS.ProgramTrace.Mode
- MCDS
- MCDS.DataTrace.Mode
- MCDS.ProgramTrace.Agents
- MCDS.state

MCDS.BusTrace.Mode — Set bus trace mode

Modern [build 139589 - DVD 09/2022]

Format:	**MCDS.BusTrace.Mode** *<mode>*
<mode>:	**Read** | **Write** | **ReadWrite**

Set possible bus trace modes.

Read	Trace address and data value of read accesses.
Write	Trace address and data value of write accesses.
ReadWrite	Trace address and data value of read and write accesses (default).

See also

- MCDS.BusTrace.Agents
- MCDS.DataTrace.Agents
- MCDS
- MCDS.DataTrace.Mode

MCDS.CLEAR Clear programming and initialize MCDS registers

Format: **MCDS.CLEAR**

The **MCDS.CLEAR** command performs the following actions:

- Performs **MCDS.Init**:
  - Enables the MCDS (**MCDS.ON**)
  - Initializes all counters e.g., used by the **BMC** commands.
  - Reprograms the entire MCDS breakpoint, trigger and trace configuration. All MCDS registers are re-written to ensure coherency between the setting assumed by TRACE32 and the target.
- Initializes the MCDS related traces (**Trace.Init**)
- Clears all settings made by the command **MCDS.Set**

See also

■ MCDS ■ MCDS.state

The **MCDS.CLOCK** command group is used for functionality related to the MCDS clocks and clock system:

- Inform TRACE32 about the MCDS clock configuration. This is required for a correct decoding of the MCDS timestamps and to calculate the CPU clock cycles. There are two different strategies:
  - Use the CLOCK feature for an automatic detection of the on-chip clock programming.
  - If the on-chip clock programming can not be used, e.g. for post-mortem analysis, the clock configuration can be specified manually.
- For C166 and XC2000ED, the programming of the MCDS on-chip clocks can be done by TRACE32 based on the manual configuration.
- Configuration of a timer to generate a periodic trigger.

For more information on the concept and usage of the **MCDS.CLOCK** commands, please refer to **"Clock System"** in MCDS User's Guide, page 62 (mcds_user.pdf).

See also

- MCDS.CLOCK.DEPRECATED
- MCDS.CLOCK.Frequency
- MCDS.CLOCK.REFDIV
- MCDS.CLOCK.SYStem
- MCDS.CLOCK.TimeStamp
- MCDS.state
- MCDS.CLOCK.EXTern
- MCDS.CLOCK.MCDSDIV
- MCDS.CLOCK.REFerence
- MCDS.CLOCK.TIMER
- MCDS

Format:	**MCDS.CLOCK.DEPRECATED** [**ON** | **OFF**] (not for GTM)

Configures the usage of the deprecated commands for specifying and programming the MCDS clocks.

- The command is available for TriCore AUDO devices only to enable deprecated historic functionality. In order to avoid programming conflicts between the application and TRACE32 users are strongly recommended not to use the deprecated functionality any more. Migration to the recommended clock system configuration may require changing the application.
- This command and the commands enabled by it are not supported for TriCore AURIX devices and newer, including GTM.
- This command is not available for C166 and XC2000ED. For these architectures, the deprecated commands are the default.

OFF (default TriCore, PCP)	Get MCDS clock configuration by reading target registers or manual configuration. Reading the target's clock configuration requires **CLOCK.ON**. The **CLOCK** feature is not available for all architectures, e.g. not for C166 and XC2000ED.
ON (default C166, XC2000ED)	Use the deprecated clock configuration method.

For more information on the MCDS clock system and configuration, refer to **"Clock System"** in MCDS User's Guide, page 62 (mcds_user.pdf).

See also

- MCDS.CLOCK

Format:	**MCDS.CLOCK.EXTern** *<frequency>* (deprecated for TriCore, PCP)

Default: Device dependent

Specifies the frequency of the external clock f_{EXT}. This is required for decoding absolute timestamps and for configuring the periodic trigger event **MCDS.CLOCK.TIMER**.

For TriCore and PCP this command has to be enabled using **MCDS.CLOCK.DEPRECATED**.

For more information on the MCDS specific clock generation of your device, see **"Device Specific Details"** in MCDS User's Guide, page 64 (mcds_user.pdf).

See also

- MCDS.CLOCK

If the MCDS related frequencies cannot be evaluated reading the target registers, e.g. in case of post-mortem analysis, the frequencies for timestamp decoding have to be specified manually. Another use case is configuring the periodic trigger event **MCDS.CLOCK.TIMER**.

The manually configured frequencies are used if **CLOCK.OFF** or the **CLOCK** feature is not available, and **MCDS.CLOCK.DEPRECATED** is **OFF**.

For more information on the MCDS clock system see **"Clock System"** in MCDS User's Guide, page 62 (mcds_user.pdf).

See also

- MCDS.CLOCK.Frequency.McdsClock
- MCDS.CLOCK.Frequency.ReferenceClock
- MCDS.CLOCK

MCDS.CLOCK.Frequency.McdsClock Specify the MCDS clock

Format: **MCDS.CLOCK.Frequency.McdsClock** *<frequency>*

Default: 0.Hz

Specifies the frequency of the MCDS clock f_{MCDS}. This is required for decoding relative timestamps.

See also

- MCDS.CLOCK.Frequency

MCDS.CLOCK.Frequency.ReferenceClock Specify the reference clock

Format: **MCDS.CLOCK.Frequency.ReferenceClock** *<frequency>*

Default: 0.Hz

Specifies the frequency of the reference clock f_{REF} This is required for decoding absolute timestamps and for configuring the periodic trigger event **MCDS.CLOCK.TIMER**.

See also

- MCDS.CLOCK.Frequency

Format: **MCDS.CLOCK.MCDSDIV** *<divider>* (deprecated for TriCore, PCP)
MCDS.CLOCK.SYStemDIV *<divider>* (deprecated)

Default: Device dependent, minimum possible value.

Configures the divider for generating the MCDS clock. The legal divider values are dependent on the device. TRACE32 knows about the limitations and auto-adjusts the user value in case the specified setting is not applicable in the current context.

For TriCore and PCP devices this command has to be enabled using **MCDS.CLOCK.DEPRECATED**.

For more information on the MCDS specific clock generation of your device, see **"Device Specific Details"** in MCDS User's Guide, page 64 (mcds_user.pdf).

See also

- MCDS.CLOCK

MCDS.CLOCK.REFDIV Set divider for generating the reference clock

Format: **MCDS.CLOCK.REFDIV** *<divider>*
MCDS.CLOCK.EXTernDIV *<divider>* (deprecated)

Default: Device dependent, minimum possible value.

Configures the divider for generating the reference clock. The legal divider values are dependent on the device. TRACE32 knows about the limitations and auto-adjusts the user value in case the specified setting is not applicable in the current context.

For more information on the MCDS specific clock generation of your device, see **"Device Specific Details"** in MCDS User's Guide, page 64 (mcds_user.pdf).

See also

- MCDS.CLOCK

Format: **MCDS.CLOCK.REFerence** [**USB** | **PLL** | **ERAY** | **BACKUP**]

Default: Device dependent.

Selects which clock is input for the reference clock f_{REF}. **USB** and **ERAY** is the external clock f_{EXT}, **PLL** is the System Clock f_{SYS} and **BACKUP** the internal Backup Clock f_{BACK}.

For more information on the MCDS specific clock generation of your device, see **"Device Specific Details"** in MCDS User's Guide, page 64 (mcds_user.pdf).

See also

- MCDS.CLOCK

MCDS.CLOCK.SYStem Set the system clock frequency

Format: **MCDS.CLOCK.SYStem** *<frequency>* (deprecated for TriCore, PCP)

Default: Device dependent.

Specifies the frequency of the system clock f_{SYS}. This is required for calculating the MCDS clock f_{MCDS}. f_{MCDS} is used for sampling the trace data generated by the cores and buses. All relative timestamp messages, including TICK messages are generated depending on f_{MCDS}.

For TriCore and PCP devices this command has to be enabled using **MCDS.CLOCK.DEPRECATED**.

See also

- MCDS.CLOCK

Format:	**MCDS.CLOCK.TIMER** [*<frequency>* | *<period>*]
<frequency>:	**1.Hz** … *<maximum_frequency>*
<period>:	**1.0** … *<maximum_period>*

Default: 0. (disabled)

MCDS has a timer driven by the reference clock. It can be used to generate a periodic trigger signal. Periods from micro seconds up to minutes are possible depending on the available clock source.

Not all values can be entered as a frequency or period. The time base for the period is seconds, but the unit “s” must not be specified on the command line. Not all frequencies are possible, an appropriate one is chosen. Entering a frequency higher than the reference clock disables the trigger generation.

The trigger signal is not available immediately. It must be connected to an event for becoming effective. For more information, refer to **“Periodic Trigger”** in MCDS User’s Guide, page 69 (mcds_user.pdf).

See also

- MCDS.CLOCK

Format: **MCDS.CLOCK.TimeStamp** [**AUTO** | **OFF** | **Relative** | **Absolute**]

Controls the decoding of timestamps. It only makes sense to change the default setting **AUTO** in a few cases only:

- Avoid long processing times by disabling the timestamp decoding. Timestamp decoding can be re-enabled at any point of time if necessary.
- When timestamps are generated manually, TRACE32 does not know that there are timestamps to be generated. Use **Relative** and **Absolute** for telling TRACE32 which timestamps to decode.

Absolute and relative timestamps can be generated simultaneously, but only one kind of them can be displayed at a time. Switching between both methods is possible, there is no need to perform a new recording.

TRACE32 only configures relative timestamps. Absolute timestamps are required for special use cases only and require manual configuration. Manual configuration requires expert knowledge. See **"Guarded MCDS Programming"** in MCDS User's Guide, page 92 (mcds_user.pdf) for more information.

AUTO (default)	Decode timestamps according to configuration made by TRACE32. The settings of **MCDS.TimeStamp** is evaluated for determining whether timestamps have to be decoded or not.
OFF	Do not decode any timestamps, even if generated. This option is useful to increase the decoding speed in case of big trace recordings.
Relative	Decode relative timestamps based on the MCDS clock.
Absolute	Decode absolute timestamps based on the reference clock.

See also

- MCDS.CLOCK

Format:	**MCDS.DataTrace.Agents** [{*<agent>*}]
<agent>:	**Core0 | Core1 | Core2 | Core3 | Core4 | Core5**

Set a list of logical cores the data should be traced for.

<agent> The number of available agents correspond to the number of cores controlled by the TRACE32 PowerView instance.
An empty list will clear all agents.

```
MCDS.DataTrace.Agents Core0 Core3 Core4
```

To disable the data trace clear all agents.

```
MCDS.DataTrace.Agents
```

See also

- MCDS.DataTrace.Mode
- MCDS.BusTrace.Agents
- MCDS.PERipheralTrace
- MCDS.ProgramTrace.Mode
- MCDS
- MCDS.BusTrace.Mode
- MCDS.ProgramTrace.Agents
- MCDS.state

Format: **MCDS.DataTrace.Mode** *<mode>*

<mode>: **Read** | **Write** | **ReadWrite**

Set possible program trace modes.

Read	Address and data reads are traced.
Write	Address and data writes are traced.
ReadWrite	Address and data reads and writes are traced

See also

- MCDS.DataTrace.Agents
- MCDS.BusTrace.Agents
- MCDS.PERipheralTrace
- MCDS.ProgramTrace.Mode
- MCDS
- MCDS.BusTrace.Mode
- MCDS.ProgramTrace.Agents
- MCDS.state

MCDS.INFO Information on MCDS and usage

Format: **MCDS.INFO**

Opens a window to provide detailed information about the MCDS of the current device:

- MCDS ID and module version.
- Emulation memory usage.
- Which MCDS features, e.g. actions, watchpoints or cross-triggers, are available and how many of them are already in use. For example, this supports an advanced or expert user writing trigger programs.

See also

- MCDS
- MCDS.state

Format: **MCDS.Init**

The **MCDS.Init** command performs the following:

- Enable MCDS (perform **MCDS.ON**).
- Initializes all counters e.g., used by the **BMC** commands.
- Reprogram the entire MCDS breakpoint, trigger and trace configuration. All MCDS registers are re-written to ensure coherency between the setting assumed by TRACE32 and the target.

See also

■ MCDS ■ MCDS.state

MCDS.OFF Disable MCDS programming

Format: **MCDS.OFF**

Default: MCDS enabled.

Disables all MCDS related debugger functionality. TRACE32 will stop programming MCDS registers. When there are no other GUIs attached to the same Emulation Device that have the MCDS enabled, TRACE32 will disable the MCDS hardware.

See also

■ MCDS ■ MCDS.state ❑ MCDS.STATE()

MCDS.ON Enable MCDS programming

Format: **MCDS.ON**

Default: MCDS enabled.

Enables all MCDS related debugger functionality, such as onchip trace, additional breakpoints, …

See also

■ MCDS ■ MCDS.state ❑ MCDS.STATE()

Format: **MCDS.Option** *<option>* {*<parameter>*}

With the **MCDS.Option** commands, the user can control the behavior of the MCDS programming.

See also

- MCDS.Option.CoreBreak
- MCDS.Option.eXception
- MCDS.Option.QuickOFF
- MCDS
- MCDS.Option.DataAssign
- MCDS.Option.FlowControl
- MCDS.Option.RESetBehavior
- MCDS.state

MCDS.Option.CoreBreak Break when BREANK_OUT becomes active

Format: **MCDS.Option.CoreBreak** [**ON** | **OFF**] (TriCore, XC2000 and GTM only)

Default: OFF.

When enabled, the core(s) stop execution as soon as the MCX action BREAK_OUT becomes active. Depending on the chip, the core break is not cycle accurate.

NOTE: This command is only relevant for the users of the **MCDS.Set** command.

See also

- MCDS.Option

MCDS.Option.DataAssign Data assignment in trace listing

Format: **MCDS.Option.DataAssign** [**ON** | **OFF**] (XC2000ED only)

Default: ON.

When enabled, the debugger tries to assign the data cycles to the associated program cycles. The not assigned data cycles are shown in red in the trace listing. When disabled, no data assignment is used.

See also

- MCDS.Option

MCDS.Option.eXception — Exception identification in trace decoder

Format:	**MCDS.Option.eXception.DCU** [**ON** | **OFF**] (TriCore only) **MCDS.Option.eXception.TABLE** *<table_config>* (TriCore only) **SYStem.Option.INTSTART** *<address>* (TriCore only) (deprecated) **SYStem.Option.INTUSE** *<value>* (TriCore only) (deprecated) **SYStem.Option.TRAPSTART** *<address>* (TriCore only) (deprecated)
<table_config>:	[**OFF** | **AUTO** | **Interrupt** {*<range>* [*<size>*]} | **Trap** {*<range>* [*<size>*]}]

Default: DCU OFF, TABLE AUTO.

MCDS.Option.eXception is a command group that configures how the MCDS trace decoder identifies the occurrence of interrupt and trap events. For TriCore it replaces the obsolete **SYStem.Option.[INTSTART | INTUSE | TRAPSTART]** commands.

In case of multicore up to six address ranges can be specified, one for each core starting with core 0. Note that all cores must be specified in ascending order, even if they are not configured for tracing (no gaps allowed). Only the last cores may be omitted if they are not configured for tracing.

Looking up the exception handler entries in the table can be disabled using OFF. It is not possible to detect only parts of the configuration automatically.

<size>	Size of an exception handler entry, default is 32 B.

See also

- MCDS.Option

Format: **MCDS.Option.FlowControl** [**OFF** | **GAP** | **STALL**]

Default: GAP.

The MCDS can generate more trace messages than the AGBT is able to transfer via the trace port. When too much trace data is generated, there is an internal AGBT FIFO overflow. In this case, trace data is lost, and the MCDS trace decoder looses synchronization. This results in corrupted trace decoding and hard errors.

The following cases are known to potentially overflow the AGBT FIFO:

- Tight loops in the code that are executed massively, e.g. the idle loop. Here, a huge number of trace messages is generated in a short period.
- Massive memory accesses to random addresses and wide data. In this case, the MCDS message encoder is not able to compress address and data efficiently.
- Too many configured trace sources.

TRACE32 provides different options to avoid the loss of trace information and to avoid trace data corruption:

OFF	Disable any flow control. It is up to the users to • Modify their applications. • Choose a configuration that will not overflow the AGBT FIFO. • Accept the consequences.
GAP	Generation of program flow trace messages is suppressed as long as the AGBT is likely to overflow. This results in gaps in the program flow and FIFOFULL messages.
STALL	The CPU execution is stalled as long as the AGBT FIFO is likely to overflow. This impacts the real-time behavior of your application.

NOTE: The AGBT FIFO full situation is due to a chip limitation, not a restriction of the TRACE32 trace hardware.

See also

- MCDS.Option

Format: **MCDS.Option.QuickOFF** [**ON** | **OFF**] (TriCore and PCP only)

Default: OFF.

When enabled, the debugger uses a hardware signal to disable the trace recording in case the CPU stops the application execution. This avoids the generation of additional messages, e.g. timestamp messages, and improves the trace buffer usage.

NOTE: **MCDS.Option.QuickOFF:**
- Only has an effect when **<trace>.AutoArm** is enabled.
- Is disabled when the **Break Action WATCH** is selected.
- Uses many trigger resources, especially in multicore scenarios. Enabling this option will reduce the number of available triggers.

See also

- MCDS.Option

MCDS.Option.RESetBehavior Configure Onchip behavior after chip reset

[build 164807 - DVD 02/2024]

Format: **MCDS.Option.RESetBehavior** [**OnchipOFF** | **OnchipArm**]

The MCDS on-chip trace buffer does not allow appending newly generated trace data to a trace recording that already exists in the on-chip trace buffer. Instead, the MCDS hardware always overwrites any existing trace data. As a chip reset will always stop any trace recording, it is thus not possible to trace through a reset event, keeping the trace data before and after the chip reset at the same time.

In case of Onchip, the command **MCDS.Option RESetBehavior** allows the user to choose which trace recording, pre- or post-reset, is more important to him.

OnchipOFF	Stop Onchip on chip reset event. TRACE32 will reconstruct any recorded trace data for inspection after the chip comes out of reset.
OnchipArm	If Onchip is in state Arm on a chip reset event, re-arm Onchip after chip comes out of reset. Trace data before chip reset gets overwritten.

See also

- MCDS.Option

MCDS.PERipheralTrace — Control peripheral trace

Modern [build 139589 - DVD 09/2022]

Format: **MCDS.PERipheralTrace** [*<control>*]

<control>: **ON** | **OFF**

Control OTGB agent for using peripheral trace. See also **"Peripheral Trace"** in MCDS User's Guide, page 48 (mcds_user.pdf) or `~~/demo/tricore/etc/trace_trigger/peripheraltrace`.

See also

- MCDS.ProgramTrace.Agents
- MCDS
- MCDS.BusTrace.Mode
- MCDS.DataTrace.Mode
- MCDS.ProgramTrace.Mode
- MCDS.BusTrace.Agents
- MCDS.DataTrace.Agents
- MCDS.state

▲ 'MCDS Special Features' in 'MCDS User's Guide'

MCDS.PortSIZE — Set number of used Aurora lanes

Format: **MCDS.PortSIZE** [*<lanes>*]

<lanes>: **1Lane**

Default: 1Lane.

The Aurora serial trace protocol supports the use of several serial data streams (lanes) in parallel. This command allows to select how many lanes are to be used in this setup. Changing the port size will result in a **Trace.Init**.

This command is only available for the AGBT (Aurora GigaBit Trace) off-chip trace feature.

See also

- MCDS
- MCDS.state

MCDS.PortSPEED — Set Aurora lane speed

Format:	**MCDS.PortSPEED** [*<speed>*]
<speed>:	**625Mbps** | **1250Mbps** | **2500Mbps**

Default: 2500Mbps.

This command defines the transfer rate of one Aurora lane. In case more than one lane is used, all lanes will operate with the same transfer rate. Using more than one lane at the same time may require reducing the port speed. Changing the port speed will result in a **Trace.Init**.

This command is only available for the AGBT (Aurora GigaBit Trace) off-chip trace feature.

See also

- MCDS
- MCDS.state

Format:	**MCDS.ProgramTrace.Agents** [{*<agent>*}]
<agent>:	**Core0 | Core1 | Core2 | Core3 | Core4 | Core5**

Set a list of logical cores the program flow should be traced for.

<agent> The number of available agents correspond to the number of cores controlled by the TRACE32 PowerView instance.
An empty list will clear all agents.

```
MCDS.DataTrace.Agents CORE0 CORE3 CORE4
```

To disable the program flow trace clear all agents.

```
MCDS.DataTrace.Agents
```

See also

- MCDS.ProgramTrace.Mode
- MCDS.BusTrace.Mode
- MCDS.PERipheralTrace
- MCDS.DataTrace.Agents
- MCDS
- MCDS.DataTrace.Mode
- MCDS.BusTrace.Agents
- MCDS.state

Format:	**MCDS.ProgramTrace.Mode** *<mode>*
<mode>:	**FlowTrace** | **SyncTrace** | **CFT**

Set possible program trace modes.

FlowTrace Program Flow Trace.
Trace information is only generated on a discontinuity of the program flow, e.g. branch or jump instructions. FlowTrace offers the best trace buffer usage but does not provide timestamp information on every executed instruction.

SyncTrace SYNC Trace mode.
Trace information is generated on every MCDS clock cycle. Timestamps are generated for almost all instructions.

CFT Compact Function Trace.
Trace information is only generated for call and return instructions. Information about function call hierarchy may be lost with advanced compiler optimation.

See also

- MCDS.ProgramTrace.Agents
- MCDS
- MCDS.BusTrace.Mode
- MCDS.DataTrace.Mode
- MCDS.PERipheralTrace
- MCDS.BusTrace.Agents
- MCDS.DataTrace.Agents
- MCDS.state

MCDS.Register — Open window with MCDS registers

Format:	**MCDS.Register** [*<file>* [*/<options>*]]
<option>:	**SpotLight** | **DualPort** | **Track** | **AlternatingBackGround** **CORE** *<core_number>*

Opens a peripheral window showing all MCDS related registers. By default, the register file of the currently selected devices is opened.

<file>	Name of the register file or comma for default.
<option>	For a description of the options, see **PER.view**.

See also

- MCDS
- MCDS.state

MCDS.RESet — Reset the MCDS unit in the debug tool

Format:	**MCDS.RESet**

The **MCDS.RESet** command performs the following actions:

- Reset all MCDS settings to their defaults
- Clears the MCDS related traces (**Trace.Init**)
- Clears all settings made by the command **MCDS.Set**
- Resets all counters e.g., used by the **BMC** commands.
- All MCDS registers are re-written to ensure coherency between the setting assumed by TRACE32 and the target.

See also

- MCDS
- MCDS.state

Commands for controlling the MCDS Resource Management. These commands are mainly for diagnostic purpose and not necessary for normal operation.

The MCDS Resource Management is a data structure containing the MCDS register configuration for maintaining coherency between multiple PowerView instances connected to the same Emulation Device and the register programming of the Emulation Device itself. This avoids conflicting register accesses and trigger setups.

The MCDS Resource Management also acts as cache to improve performance.

See also

- MCDS.RM.ReSTore
- MCDS.RM.WriteTarget
- MCDS
- MCDS.state

MCDS.RM.ReSTore Restore MCDS registers

Format: **MCDS.RM.ReSTore**

Re-writes all MCDS registers.

All modified MCDS registers are re-written, overwriting any manual change by the user. If an MCDS register has an internal reset value, the register will be reset to this value.

See also

- MCDS.RM

MCDS.RM.WriteTarget Flush MCDS register cache

Format: **MCDS.RM.WriteTarget**

Writes internally cached MCDS register settings to target.

All modified MCDS registers are re-written. If a register was changed by the user but not by TRACE32, the user's setting will not be overwritten. **MCDS.Set** modifications are considered to be TRACE32 related.

See also

- MCDS.RM

Format:	**MCDS.SessionKEY** *<64_bit_value>*

Default: 0x0000000000000000.

Provides a 64-bit MCDS session key for unlocking the MCDS in case it is locked by the application. This is normally only required in very late stages of the development phase.

See also

- MCDS
- MCDS.state

MCDS.Set Program MCDS on hardware level

[Example]

Format:	**MCDS.Set** *<unit>*.*<feature>* [*<setting>*] {*<setting>*} [*/<option>*]
<unit>:	**MCX** | **CpuMux0** | **CpuMux1** | **CpuMux2** | **TriCore** | **PCP** | **C166** | **SPB** | **RPB** | **LMB** | **SRI**
<option>:	**Default** | **Cached** | **WriteThru**

The **MCDS.Set** commands provide an interface to program an MCDS feature from a logical point of view. Although the commands are quite comfortable and more or less self-explaining a detailed understanding of the MCDS implementation is mandatory. See the Infineon MCDS documentation for details.

<unit>	*<unit>* is a core, bus or the MCX
<feature>	Source and device dependent.
<setting>	Source, feature and device dependent.
MCX	Program a feature of the Multi-core Cross-connect.
CpuMux0	Program a feature of the processor connected to CPU multiplexer 0.
CpuMux1	Program a feature of the processor or the OTGM connected to CPU multiplexer 1.
CpuMux2	Program a feature of the processor or the OTGM connected to CPU multiplexer 2.

TriCore	Program a feature of the TriCore processor.
PCP	Program a feature of the PCP processor.
C166	Program a feature of the C166 processor.
SPB	Program a feature of the System Peripheral Bus.
RPB	Program a feature of the Remote Peripheral Bus.
LMB	Program a feature of the Local Memory Bus.
SRI	Program a feature of the Shared Resource Interconnect.

An **MCDS.Set** command programs all registers belonging to the selected feature, e.g. an IP pretrigger programs the bound and the ranges value at the same time. Implicit information is added automatically.

Example: This script enables the Program Flow Trace of a TriCore AUDO as long as the CPU executes code from the function `sieve()`. This example is equivalent to the **Break Action TraceEnable**.

```
MCDS.Set TriCore IP0                    ; Pretrigger IP0 is active as long
Var.RANGE(sieve)                        ; as TriCore executes code within
                                        ; the function sieve()

MCDS.Set TriCore EVT0 IP0               ; enable event EVT0 as long as
                                        ; pretrigger IP is active

MCDS.Set TriCore ACT.PTU_EN 0. EVT0     ; enable TriCore Program Flow Trace
/Normal /High                           ; as long as event EVT0 is active
```

All common MCDS use cases are available as **Trace Triggers and Filters** via the **Break.Set** command. The **MCDS.Set** commands allow setting up filters and triggers for special use cases. They can be used stand alone, in parallel or as an extension to the **Trace Triggers and Filters**. The last one of course requires a detailed knowledge of how the debugger programs the MCDS. This knowledge is not documented and may change without prior notice.

For performance reason all MCDS register accesses are cached by the TRACE32 software and written to the hardware when necessary, e.g. when resuming program execution, see the **MCDS.RM** command. As this can impact a currently active trigger configuration, the user can specify whether the **MCDS.Set** command is to be executed immediately (option **WriteThru**) or delayed (option **Cached**) until the next automatic write back. Default is the standard behavior (Cached for TriCore and C166/ XC2000).

For more information, see **"Guarded MCDS Programming"** in MCDS User's Guide, page 92 (mcds_user.pdf).

See also

■ MCDS ■ MCDS.state

▲ 'Introduction' in 'Application Note for Complex Trigger Language'

MCDS.SOURCE — Set MCDS trace sources

Classic

The **MCDS.SOURCE** command group controls which on-chip modules (sources) generate which kind of trace data. In general, there are three basic kind of trace sources:

- **Core trace**: Trace data generated by an execution unit, e.g. program flow but also memory accesses
- **Bus trace**: Trace data generated by a bus unit, e.g. destination address, data and meta information.
- **Peripheral trace**: Trace date generated by peripherals, e.g. DMA or Interrupt Router, or special execution units, e.g. GTM.

See also

- MCDS.SOURCE.ALL
- MCDS.SOURCE.DEFault
- MCDS.SOURCE.NONE
- MCDS.SOURCE.Set
- MCDS
- MCDS.state

MCDS.SOURCE.ALL — Enable all MCDS trace sources

Classic

Format: **MCDS.SOURCE.ALL**

The virtual trace sources **ALL** enables all available trace sources and types in one step.

See also

- MCDS.SOURCE

MCDS.SOURCE.DEFault — Set default MCDS trace sources

Classic

Format: **MCDS.SOURCE.DEFault**
MCDS.SOURCE.RESet (deprecated)

Sets all **MCDS.SOURCE** configurations to their default values.

See also

- MCDS.SOURCE

Format:	**MCDS.SOURCE.NONE**

The virtual trace sources **NONE** disable all available trace sources and types in one step.

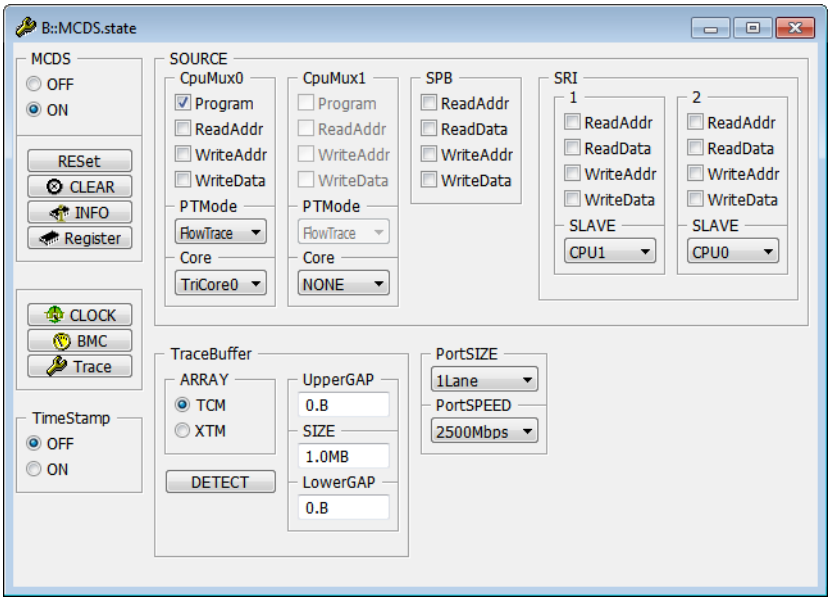
See also

- MCDS.SOURCE

Format 1: **core trace**	**MCDS.SOURCE.Set** *<cpu>*.*<parameter>* **MCDS.SOURCE.Set** *<cpu>*.*<parameter>* (deprecated for TC3x)
<cpu>: **CPU source**	**CpuMux0** | **CpuMux1** | **CpuMux2** | **TriCore** | **PCP** | **C166**
<parameter>: **MUX source**	**Core NONE** **Core TriCore0** **Core TriCore1** **Core TriCore2** **Core OTGM**
<parameter>: **CPU trace messages**	**Program** [**ON** | **OFF**] **ReadAddr** [**ON** | **OFF**] **ReadData** [**ON** | **OFF**] **WriteAddr** [**ON** | **OFF**] **WriteData** [**ON** | **OFF**] **OwnerShip** [**ON** | **OFF**] **Debug** [**ON** | **OFF**]
<parameter>: **program trace mode**	**PTMode FlowTrace** **PTMode SyncTrace** **PTMode CFT**
<parameter>: **cft trace mode filter**	**LeafFctLength.***<function length>*
Format 2: **bus trace**	**MCDS.SOURCE.Set** *<bus>*.*<parameter>*
<bus>: **bus sources**	**SPB** | **RPB** | **LMB**
<parameter>: **bus trace messages**	**ReadAddr** [**ON** | **OFF**] **ReadData** [**ON** | **OFF**] **WriteAddr** [**ON** | **OFF**] **WriteData** [**ON** | **OFF**] **Debug** [**ON** | **OFF**]
Format 3: **SRI trace**	**MCDS.SOURCE.Set SRI.**(**1** | **2**).*<parameters>* **MCDS.SOURCE.Set SRI.Debug** [**ON** | **OFF**]

<parameter>:	**SLAVE CPU0 | CPU1 | CPU2**
SRI slave	**SLAVE CPU1_PMI | CPU1_DMI | CPU2_PMI | CPU2_DMI**
	SLAVE PMI | DMI
	SLAVE PMU0 | PMU1
	SLAVE PMU0_PFLASH0 | PMU0_PFLASH1
	SLAVE PMU0_PFLASH2 | PMU0_PFLASH3 | PMU0_DFLASH
	SLAVE EBU | LMU | SFI | XBAR
<parameter>:	**ReadAddr [ON | OFF]**
SRI messages	**ReadData [ON | OFF]**
	WriteAddr [ON | OFF]
	WriteData [ON | OFF]

The **MCDS.Source.Set** commands are used till Aurix1G inclusively Aurix2G Astep. Since Aurix2G more abstracted User Interface, similar to that of other Architectures are introduced. See also **MCDS.ProgramTrace**, **MCDS.DataTrace**, **MCDS.BusTrace** and **MCDS.PERipheralTrace**



<cpu> defines the core for which type of trace *<message>* is to be generated. For single-core systems and TriCore AUDO, *<cpu>* directly addresses the core. For TriCore AURIX *<cpu>* addresses a multiplexer which allows to choose a core or the **OTGM**. For TriCore AURIX only up to two cores or up to one core and the **OTGM** can be traced.

Available <cpu> sources:

CpuMux0	Generate trace data for the core selected by CPU multiplexer 0. Required for TriCore AURIX and later.
CpuMux1	Generate trace data for the core or the OTGM selected by CPU multiplexer 1. Required for TriCore AURIX and later.
CpuMux2	Generate trace data for the core or the OTGM selected by CPU multiplexer 2. Required for TriCore AURIX and later.
TriCore	Generate trace data for the TriCore core. Required for TriCore AUDO only.
PCP	Generate trace data for the PCP core. Required for TriCore AUDO only.
C166	Generate trace data for the C166 core. Required for C166 and XC2000ED only.

Available multiplexer sources for MCDS.SOURCE.<cpu>.Core command:

NONE	Disable trace data generation for this multiplexer. This will also disable the trigger generation (**Break.Set**) for this multiplexer.
TriCore0 | **TriCore1** | **TriCore2**	Select core as input for **CpuMux0**, **CpuMux1** or **CpuMux2**. Not all cores are valid inputs for all multiplexers.
OTGM	Select the peripheral trace as input for **CpuMux1**. Only write data trace is available for this *<source>*. OTGM is not only used for tracing dedicated peripherals, e.g. DMA or Interrupt Router, but also the core trace of GTM.

For more information on GTM and peripheral trace, refer to:

- **"Special Trace Sources via OTGM"** in MCDS User's Guide, page 46 (mcds_user.pdf)
- **"GTM Debugger and Trace"** (debugger_gtm.pdf)

Available CPU trace message types:

Program	Configure generation of program trace messages. Different modes of generating the program trace are available, see the description of the **PTrace** option below.
ReadAddr	Configure generation of trace messages (data address) on read accesses. Not on all architectures and chips.
ReadData	Configure generation of trace messages (data value) on read accesses. Not on all architectures and chips.
WriteAddr	Configure generation of trace messages (data address) on write accesses.
WriteData	Configure generation of trace messages (data value) on write accesses.
OwnerShip	Configure generation of ownership trace messages. Depending on the core, different type of information is generated: • PCP: current channel ID. • TriCore: active memory protection set. The availability of this trace type depends on the architecture and the chip.
Debug	Configure generation of debug and status related trace messages. Generated trace messages provide additional information, e.g. halted or whether the exception handler is active. If not explicitly supported by TRACE32, e.g. for exceptions, this information can only be displayed as a value.

The default core trace source is program flow trace for the first core of the chip.

Any trace filters programmed using the **Break.Set** command only have an effect on the enabled trace sources.

Example: The following configuration will only generate write trace messages if TriCore core 0 writes to the variable magic, but not if TriCore core 1 accesses it:

```
Break.Set magic /Write /TraceEnable

MCDS.SOURCE.Set CpuMux0.Core TriCore0
MCDS.SOURCE.Set CpuMux0.WriteAddr ON
MCDS.SOURCE.Set CpuMux0.WriteData ON

MCDS.SOURCE.Set CpuMux1.Core TriCore1
MCDS.SOURCE.Set CpuMux1.WriteAddr OFF
MCDS.SOURCE.Set CpuMux1.WriteData OFF
```

Available program trace modes for PTMode:

FlowTrace	Program Flow Trace. Trace information is only generated on a discontinuity of the program flow, e.g. branch or jump instructions. FlowTrace offers the best trace buffer usage but does not provide timestamp information on every executed instruction.
SyncTrace	SYNC Trace mode. Trace information is generated on every MCDS clock cycle. Timestamps are generated for almost all instructions.
CFT	Compact Function Trace. Trace information is only generated for call and return instructions. Information about function call hierarchy may be lost with advanced compiler optimation.

Filter for CFT program trace modes:

LeafFctLength. <function length>	The minimum length of leaf functions which will be traced in CFT program trace mode.

Trace Source Configuration for Buses

<bus> defines the bus system for which generation of trace data of *<type>* is to be configured. Bus trace is not available for C166 and XC2000ED.

Available <bus> sources:

SPB	Configure trace data generation for the System Peripheral Bus.
RPB	Configure trace data generation for the Remote Peripheral Bus. Only available for TC1796ED devices.
LMB	Configure trace data generation for the Local Memory Bus. LMB bus trace is not available for AUDO-NG devices.

Available bus trace message types:

ReadAddr	Configure generation of trace messages (data address and meta information) on read accesses.
ReadData	Configure generation of trace messages (data value) on read accesses.
WriteAddr	Configure generation of trace messages (data address and meta information) on write accesses.

WriteData	Configure generation of trace messages (data value) on write accesses.
Debug	Configure generation of debug and status related trace messages. Generated trace messages provide additional information on the bus, e.g. sleeping, reset, error. If not explicitly supported by TRACE32, e.g. for reset, this information can only be displayed as a value.

Meta information provides information, e.g. on the bus master, DMA channel or on the access mode.

Trace Source Configuration for SRI

The **SRI** is a fabric that connects the cores and on-chip memories on recent TriCore devices (TriCore core architecture v1.6 and later). **SRI** can handle multiple transactions in parallel. The SRI trace can only observe the transactions to up to two bus slaves (the destination of the data transfer). The availability of these slaves is device dependent.

The debug trace messages for SRI are generated for the entire SRI and not independently for each slave.

Available <sri> sources:

SRI	Configure trace data generation for the Shared Resource Interconnect.

Observable SRI slaves:

The availability of the SRI slaves which can be observed is device and slave dependent.

Available SRI trace message types:

ReadAddr	Configure generation of trace messages (data address and meta information) on read accesses.
ReadData	Configure generation of trace messages (data value) on read accesses.
WriteAddr	Configure generation of trace messages (data address and meta information) on write accesses.
WriteData	Configure generation of trace messages (data value) on write accesses.
Debug	Configure generation of debug and status related trace messages. Generated trace messages provide additional information on the bus, e.g. sleeping, reset, error. If not explicitly supported by TRACE32, e.g. for reset, this information can only be displayed as a value. Debug message generation is independent of the slaves.

Meta information provides information, e.g. on the bus master, DMA channel or on the access mode.

See also

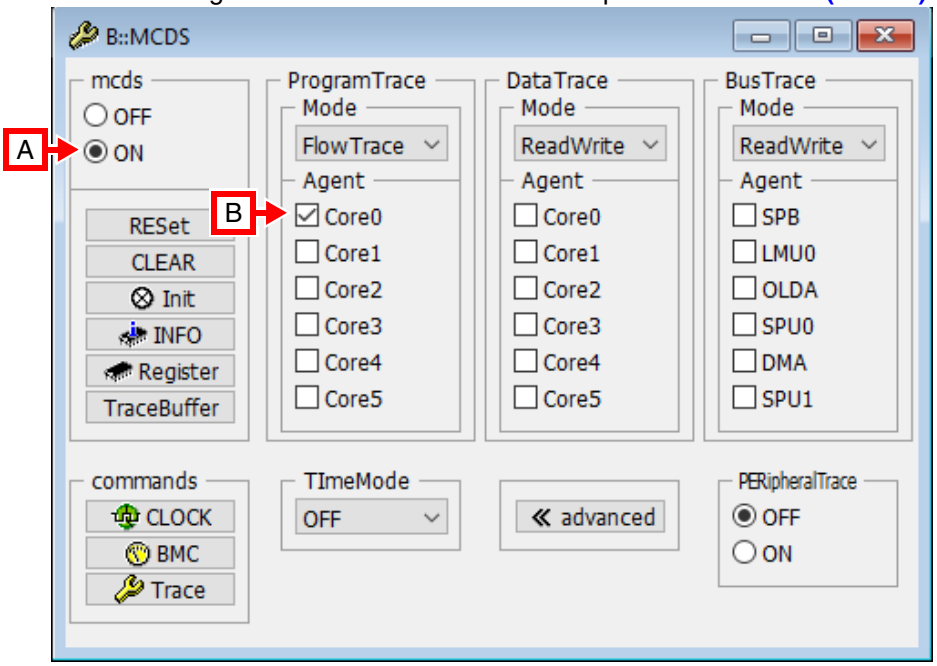
- MCDS.SOURCE

Modern

[build 139493 - DVD 09/2022]

Format:	**MCDS.state**

Opens the MCDS configuration window. The MCDS.State window below shows the actual view since Aurix2G. For older generations inclusive Aurix2G Astep see **MCDS.State (classic)**. ^^

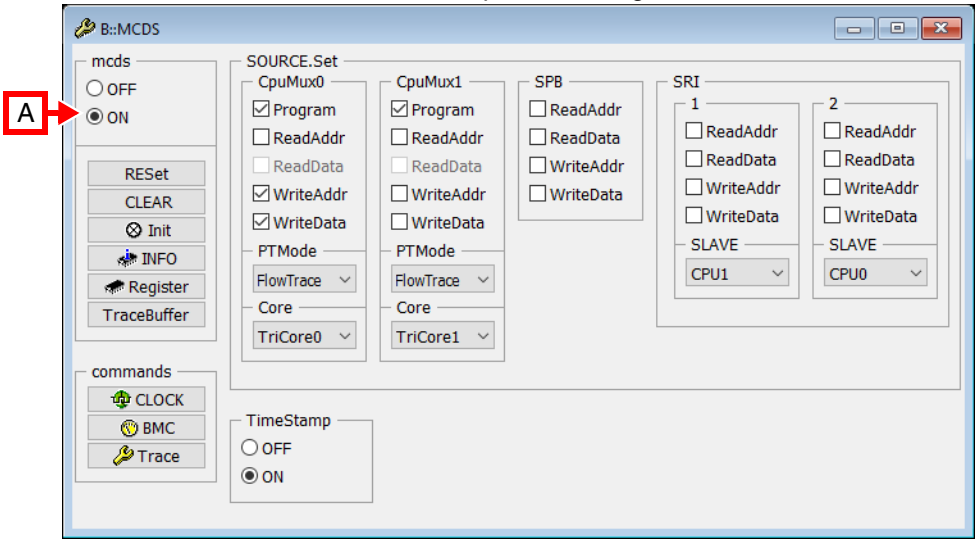


A For a description of the commands in the **MCDS.state** window, refer to the **MCDS.*** commands in this chapter.
Example 1: For information about **ON**, see **MCDS.ON**.

B **Example 2**: For setting program trace agents see **MCDS.ProgramTrace.Agent**.

Format:	**MCDS.state**

Opens the MCDS configuration window. The MCDS.State window below shows the deprecated view which is still shown till Aurix1G and Aurix2G AStep. For newer generations see **MCDS.State**.



A For a description of the commands in the **MCDS.state** window, refer to the **MCDS.*** commands in this chapter.
Example 1: For information about **ON**, see **MCDS.ON**.

See also

- MCDS
- MCDS.BusTrace.Mode
- MCDS.CLOCK
- MCDS.DataTrace.Mode
- MCDS.Init
- MCDS.ON
- MCDS.PERipheralTrace
- MCDS.PortSPEED
- MCDS.ProgramTrace.Mode
- MCDS.RESet
- MCDS.SessionKEY
- MCDS.SOURCE
- MCDS.TraceBuffer
- MCDS.BusTrace.Agents
- MCDS.CLEAR
- MCDS.DataTrace.Agents
- MCDS.INFO
- MCDS.OFF
- MCDS.Option
- MCDS.PortSIZE
- MCDS.ProgramTrace.Agents
- MCDS.Register
- MCDS.RM
- MCDS.Set
- MCDS.TImeMode

▲ 'TRACE32 Support for Emulation Devices' in 'MCDS User's Guide'

Format: **MCDS.TImeMode** [**OFF** | **External** | **MCDS** | **MCDS+External**]
MCDS.TimeStamp (deprecated)

Timestamps are used to analyze the temporal behavior of a program or signal. They can be generated on-chip (Onchip: **MCDS**), or off-chip by the tool (**Analyzer**, **CAnalyzer**, …).

MCDS timestamps are generated in the instance of the event that creates the related trace message. Thus, they are the most accurate timing information. However, they consume onchip buffer, trace bandwidth, and have an impact on the decoding performance. They can not be correlated with any other trace recording.

Tool timestamps are generated in the instance when a trace packet arrives in the tool. Due to long signal paths and chip-internal buffers, there is a non-deterministic, varying delay. Multiple trace messages will get an identical tool timestamp. Thus, tool timestamps are inaccurate.

TRACE32 allows correlating the MCDS timestamps with the tool timestamps. This improves the decoder performance and allows compensation of chip-internal delays and buffers. It also allows correlation with other trace sources sharing a common timestamp base.

OFF	Do not generate any MCDS-based timestamps. If tool timestamps exist, they are used for decoding.
External	Do not generate any MCDS-based timestamps. Use the tool timestamps for decoding. **External** is only available, if a TRACE32 trace tool is connected.
MCDS	Generate MCDS-based timestamps and use them for decoding. If a TRACE32 trace tool is connected, ignore the tool timestamps.
MCDS+External	Generate MCDS-based timestamps, and correlate them with the tool timestamps for a better resolution. This allows correlating the trace recording with any other TRACE32 tool recordings, e.g., the Mixed-Signal Probe, the Power Integrator or any other TRACE32 trace tool connected to the same PodBus device chain. **MCDS+External** is only available, if a TRACE32 trace tool is connected.

See also

- MCDS
- MCDS.state

Format:	**MCDS.TraceAgents.CLEAR**

Clear the list of all trace agents (logical cores, bus agents and OTGB agent) previously set via the commands **MCDS.ProgramTrace.Agents**, **MCDS.DataTrace.Agents**, **MCDS.BusTrace.Agents**, and **MCDS.PERipheralTrace**.

The **MCDS.TraceBuffer** commands allow to configure the EMEM for being used as on-chip trace buffer or AGBT FIFO. A correct setup is not only required for the operation of the trace modes but also for cooperation with third-party applications such as calibration tools, or when using parts of the EMEM for application.

TriCore miniMCDS, C166 and XC2000ED do not allow to configure the trace buffer.

NOTE:
- All **MCDS.TraceBuffer** commands influence each other. Especially pay attention to the **MCDS.TraceBuffer.NoStealing** setting.
- When switching the trace method, the current trace buffer configuration (array, size, lower and upper gap) will be remembered when switching back to this method.
- When switching the memory arrays within the same trace method, the trace buffer configuration (size and gap) will be reset to the default values according to the newly selected trace method.
- Always check the results of your configuration to avoid unwanted effects.

For an overview and details, see chapter **"Emulation Memory"** in MCDS User's Guide, page 73 (mcds_user.pdf).

See also

- MCDS.TraceBuffer.ARRAY
- MCDS.TraceBuffer.LowerGAP
- MCDS.TraceBuffer.SIZE
- MCDS.TraceBuffer.UpperGAP
- MCDS.state
- MCDS.TraceBuffer.DETECT
- MCDS.TraceBuffer.NoStealing
- MCDS.TraceBuffer.state
- MCDS

MCDS.TraceBuffer.ARRAY Select MCDS trace buffer array

Format: **MCDS.TraceBuffer.ARRAY** [**TCM** | **XTM**]

Selects which memory array is to be used as on-chip trace buffer or memory array. Not all memory arrays are available for all devices. Memory arrays that cannot be used as trace buffer, e.g. XCM (calibration only), cannot be configured.

TCM Use TCM (Trace- and Calibration Memory) as trace buffer.
Huge trace tiles. Default for all onchip traces.

XTM Use XTM (Extended Trace Memory) as trace buffer.
Small trace tiles for use as FIFO. Default for all off-chip traces, if available.

See also

- MCDS.TraceBuffer

Format: **MCDS.TraceBuffer.DETECT**

Reads the EMEM configuration from the device and tries to detect which memory array and which tiles can be used as trace buffer. This feature is useful if a third-party tool or the application also uses the emulation memory to allow a concurrent use.

NOTE: Ensure that the third-party tool or application already configured the EMEM for its purpose when using this command. Strange effects will occur when the EMEM configuration is changed by application while tracing. TRACE32 will not be able to access the EMEM, also the trace recording is stopped and the trace data stored in the on-chip trace buffer will be destroyed.

The first suitable trace buffer configuration found will be used for tracing. For on-chip trace, TCM is the preferred memory array, for off-chip trace XTM is preferred. If the preferred memory array does not contain a suitable trace buffer configuration, another array is selected. If no array contains a suitable configuration, the trace buffer size is set to zero.

NOTE: Always check the results of the detection to avoid any unwanted setup.

The first suitable trace buffer configuration is not necessarily the largest possible configuration. The search for the trace buffer array starts at tile 0 and stops, when after the first range of suitable tiles a non-suitable tile is found.

See also

- MCDS.TraceBuffer

Format:	**MCDS.TraceBuffer.LowerGAP** *<size>*

Default: 0 bytes.

Configures which EMEM tiles at the lower boundary of the currently selected memory array are not used as trace buffer, starting with tile 0. Some devices, e.g. TriCore AUDO-NG and XC2000, do not support configuration of a lower gap.

<size>	The *<size>* of the trace buffer lower gap can be entered in Bytes, KB or MB and will be rounded up by the software to match a multiple of the tile size. The configuration of the upper gap will be adjusted accordingly. The trace buffer size is only adjusted in case a further reduction of the upper gap is not possible or the device does not support an upper gap.

See also

- MCDS.TraceBuffer
- MCDS.TraceBuffer.LowerGAP()

Format:	**MCDS.TraceBuffer.NoStealing** [**ON** | **OFF**]

Default: ON.

ON	Do not destroy the EMEM configuration of another tool or application. Instead, TRACE32 tries to find another suitable configuration. If this is not possible, the size of the trace buffer is either set to zero (on-chip trace) or the trace method is disabled at all (off-chip trace).
OFF	Force using EMEM tiles for tracing even if already assigned to a third-party tool or application. This allows TRACE32 to destroy a configuration of another tool or an application in order to use the assigned memory tiles (all or some) for tracing. A warning message is printed in this case.

Always check the result of your trace buffer configuration to avoid any unwanted setup.

NOTE:	Devices that do not support the unused mode for trace tiles should be handled with care. For these devices unused mode is identical with calibration mode, so NoStealing should only be enabled in case a third-party tool or the application always maps tiles not used for their purpose to trace mode.

See also

- MCDS.TraceBuffer

MCDS.TraceBuffer.SIZE Set MCDS trace buffer size

Format:	**MCDS.TraceBuffer.SIZE** *<size>* **MCDS.SIZE** *<size>* (deprecated)

Default: maximum possible trace buffer size.

Configures how many EMEM tiles of the currently selected memory array are used as trace buffer. The value of the trace buffer size can be entered in Bytes, KB or MB, the debugger automatically adjusts to a possible value. Depending on the device, not all Emulation Memory can be used as trace buffer.

Some devices, e.g. XC2000, do not support configuration of the trace buffer size. When using off-chip trace, the trace buffer is used as AGBT FIFO. In this case the trace buffer size cannot be changed.

The values of the lower and upper gap are adjusted accordingly. Use **MCDS.TraceBuffer.LowerGAP** and **MCDS.TraceBuffer.UpperGAP** to align the trace buffer within the EMEM. Always check the result of your trace buffer configuration to avoid any unwanted setup.

See also

■ MCDS.TraceBuffer ❏ MCDS.TraceBuffer.SIZE()

▲ 'Trace Configuration within TRACE32' in 'Training AURIX Tracing'

MCDS.TraceBuffer.state Show trace buffer state window

Format:	**MCDS.TraceBuffer.state**

Opens the **MCDS.TraceBuffer.state** window for configuring the trace buffer settings.

See also

■ MCDS.TraceBuffer

MCDS.TraceBuffer.UpperGAP Set MCDS trace buffer upper gap

Format:	**MCDS.TraceBuffer.UpperGAP** *<size>* **MCDS.GAP** *<size>* (deprecated)

Default: 0 bytes.

Configures which EMEM tiles at the upper boundary of the currently selected memory array are not used as trace buffer, starting with the highest tile. Some devices, e.g. XC2000, do not support configuration of an upper gap.

<size>	The *<size>* of the trace buffer upper gap can be entered in Bytes, KB or MB and will be rounded up by the software to match a multiple of the tile size. The configuration of the lower gap will be adjusted accordingly. The trace buffer size is only adjusted in case a further reduction of the lower gap is not possible or the device does not support a lower gap.

See also

■ MCDS.TraceBuffer ❏ MCDS.TraceBuffer.UpperGAP()

Format: **MCDSBase***<trace>* (diagnostic use only)

<trace>: **Analyzer** | **Onchip**

MCDSBaseAnalyzer and **MCDSBaseOnchip** process the MCDS trace data recorded by the **Analyzer** or **Onchip** trace without any optimization or fine tuning. The purpose of this command is to find issues related to trace decoder optimizations.

MCDSBaseAnalyzer and **MCDSBaseOnchip** are used as *<trace>* aliases.

Example:

```
MCDSBaseAnalyzer.List ; display non-optimized trace content
```

NOTE: TRACE32 automatically detects which optimizations are necessary.

MCDSDCA<trace> MCDS trace processing with data cycle assignment

Format: **MCDSDCA***<trace>* (diagnostic use only)

<trace>: **Analyzer** | **Onchip**

MCDSDCAAnalyzer and **MCDSDCAOnchip** process the MCDS trace data recorded by the **Analyzer** or **Onchip** trace after DDTU reordering and data cycle assignment optimizations. The purpose of this command is to find issues related to trace decoder optimizations.

Data cycle assignment is an optimization where TRACE32 assigns recorded core data cycles (read, write) to the corresponding recorded program cycles. A requirement is the correct order of the core's data cycles.

MCDSDCAAnalyzer and **MCDSDCAOnchip** are used as *<trace>* aliases.

Example:

```
MCDSDCAAnalyzer.List ; display trace content after data cycle assignment
```

NOTE: TRACE32 automatically detects which optimizations are necessary.

Format:	**MCDSDDTU***<trace>* (diagnostic use only)
<trace>:	**Analyzer** | **Onchip**

MCDSDDTUAnalyzer and **MCDSDDTUOnchip** process the MCDS trace data recorded by the **Analyzer** or **Onchip** trace after DDTU reordering. The purpose of this command is to find issues related to trace decoder optimizations.

DDTU (Duplex Data Trace Unit) reordering is an optimization where TRACE32 reorders core- and bus data cycles into their correct temporal order. Timestamps is a requirement for DDTU reordering.

MCDSDDTUAnalyzer and **MCDSDDTUOnchip** are used as *<trace>* aliases.

Example:

```
MCDSDDTUAnalyzer.List ; display trace content after DDTU reordering
```

NOTE: TRACE32 automatically detects which optimizations are necessary.

MIPS

Number of instructions per second

See also

- MIPS.List
- MIPS.ListNesting
- MIPS.PROfileChart
- MIPS.PROfileSTATistic
- MIPS.STATistic

▲ 'Release Information' in 'Legacy Release History'

Overview MIPS

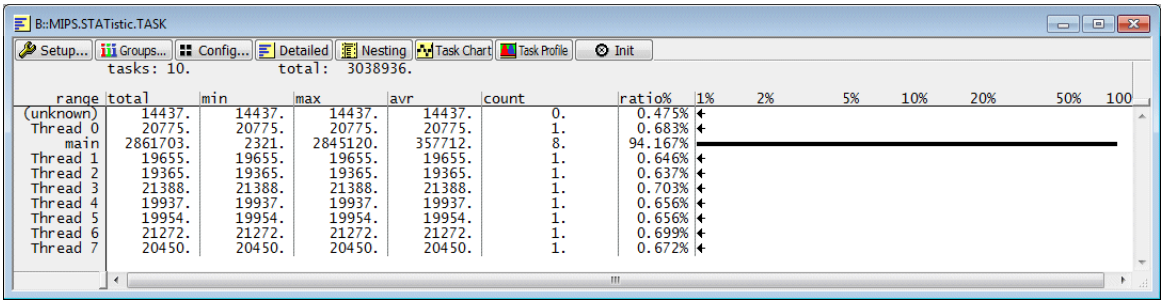
The **MIPS** command group can be used to analyze the workload (MIPS) of your systems. The source for this analysis is the trace information recorded into the selected trace sink (**Trace.METHOD** command).

The system can be analyzed under different aspects: workload per task, workload per high-level language line, workload per specified functional group etc.

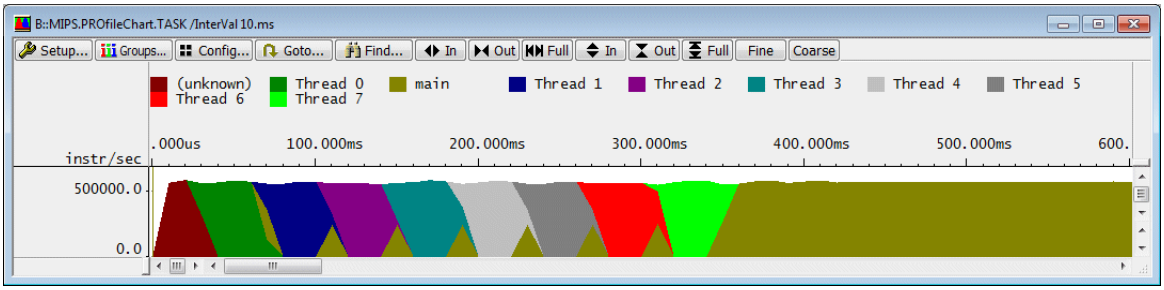
The following results are provided (workload per task as example):

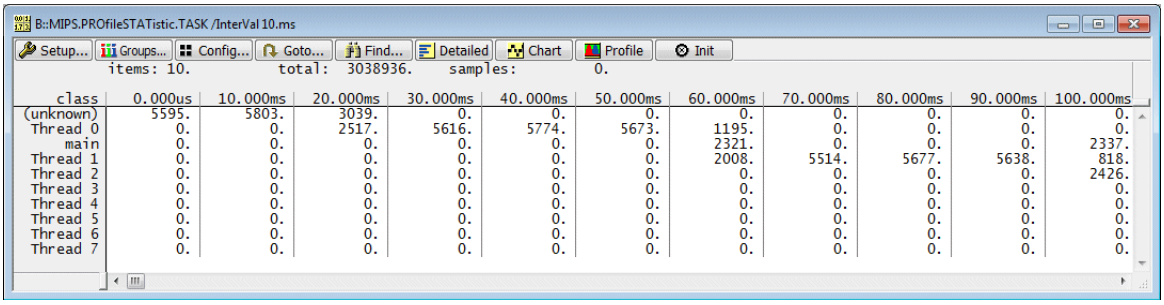
```
; the trace information recorded to the PowerTrace is analyzed by the
; MIPS commands
Trace.METHOD Analyzer

MIPS.STATistic.TASK /InterVal 10.ms

MIPS.PROfileChart.TASK /InterVal 10.ms

MIPS.PROfileSTATistic.TASK /InterVal 10.ms
```

Numeric statistical analysis of the MIPS per task for the recording time.



Graphical analysis of the MIPS per task for the recording time.



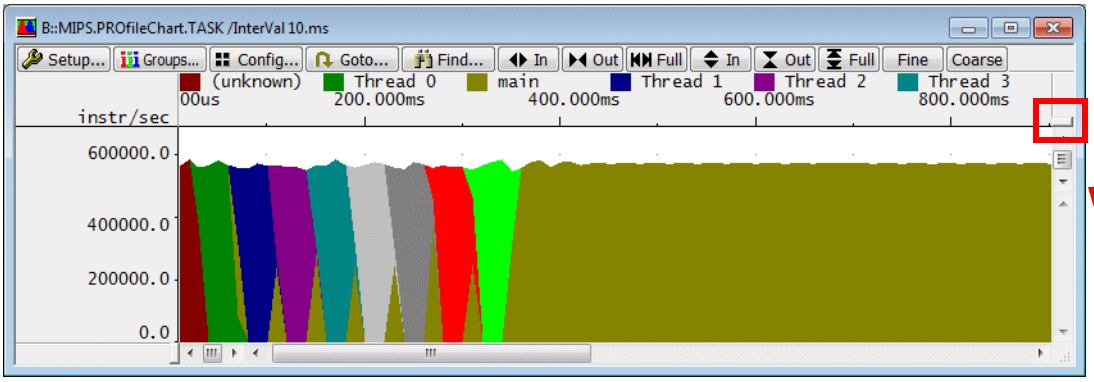


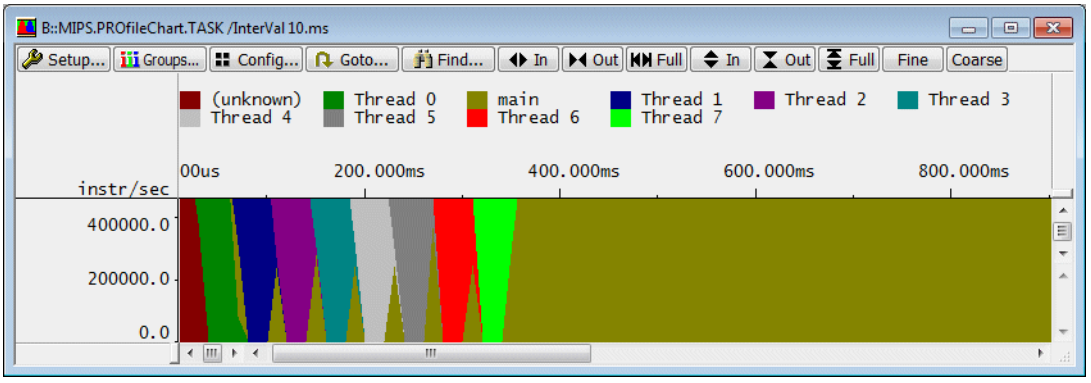
The observation time is cut into segments in order to provide the graphical analysis. The default segment size is 10.us. The **/InterVal** *<time>* option allows to specify a different segment time.

For each segment the MIPS per task are calculated. Based on this calculation the workload for the different tasks is displayed in a graphic.

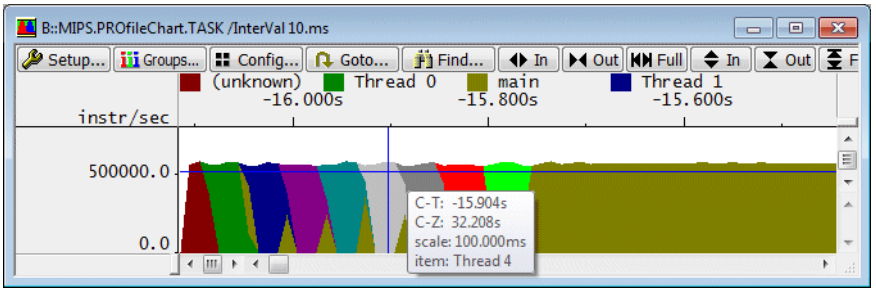
The command subgroup **MIPS.PROfileSTATistic.***<item>* is provided to enable the export of the results as CSV file (Comma-Separated Value).

```
; specify the file name and select Comma-Separated Values
; as output format
PRinTer.FILE Mips  CSV

; send the result to the file
WinPrint.MIPS.PROfileSTATistic.TASK /InterVal 10.ms
```



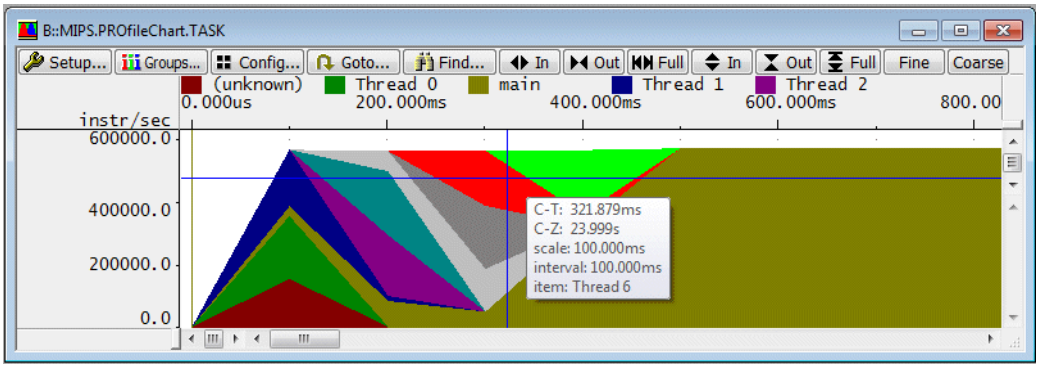


In order to see the color assignment for all tasks drag the slider.



The color assignment is also displayed in the tool tip.

Fine	Decrease segment size by factor 10.
Coarse	Increase segment size by factor 10.



The current interval is displayed in the tool tip.

MIPS.List — List the MIPS trace data

[build 135171 - DVD 09/2021]

Format: **MIPS.List** [*<record>* | *<record_range>* | *<time>* | *<time_range>* | *<bookmark>*] [*<items>*...] | [**/***<options>*]

<items>:
%*<formats>*
DEFault | **ALL** | **CPU** | **LINE** | **PORT**
Run
CYcle | **Data**[**.***<subitem>* | **BDATA** | **List**[**.***<subitem>*]
Address | **BAddress** | **FAddress** |**sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** | **Var**
TIme[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*]
FUNC | **FUNCR** | **FUNCVar** | **IGNORE**
LeVel | **MARK**[**.***<marker>*
SPARE

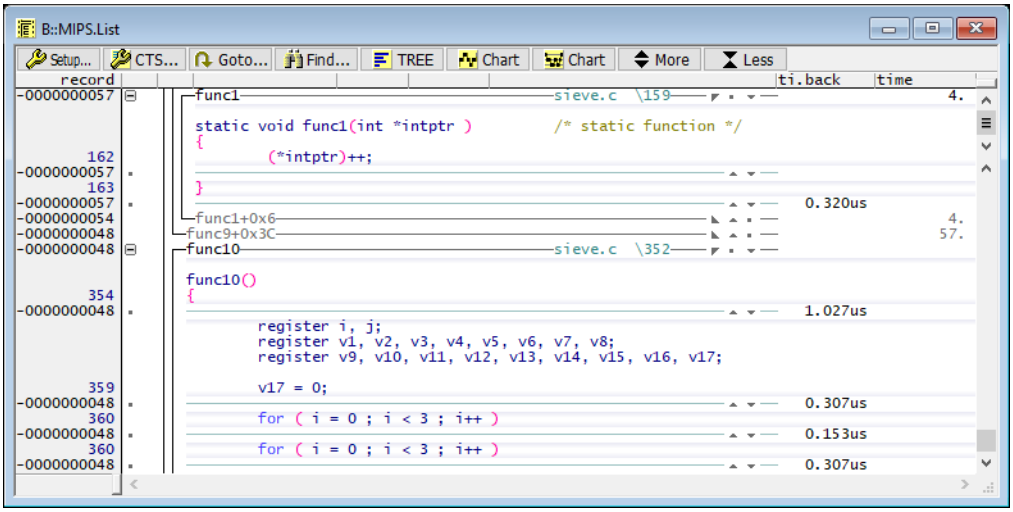
<formats>:
DEFault | **LEN** | **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii**
Signed | **Unsigned** | **TimeAuto** | **TimeFixed**

<options>:
FILE | **FlowTrace** | **BusTrace** | **MACHINE** | **NorthWestGravity** | **Mark** | **Track**

Opens a window showing the recorded trace data starting at the specified *<record>* or for a range of trace records *<record_range>* (e.g. `(-10000.)--(-2000.)`).

The columns of the window can be defined using the *<list_items>*. The order of the columns in the window is according to the order of the *<list_item>* parameters given.

<options> For a detailed description of all other parameters and options, refer to the **<trace>.List** command.



See also

- MIPS

Format:	**MIPS.ListNesting** [*<record>* | *<record_range>* | *<time>* | *<time_range>* | *<bookmark>*] [*<items>* …] [**/***<options>*]
<items>:	**%***<formats>* **DEFault** | **ALL** | **CPU** | **LINE** | **PORT** **Run** **CYcle** | **Data**[**.***<subitem>* | **BDATA** | **List**[**.***<subitem>*] **Address** | **BAddress** | **FAddress** |**sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** | **Var** **TIme**[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*] **FUNC** | **FUNCR** | **FUNCVar** | **IGNORE** **LeVel** | **MARK**[**.***<marker>* **SPARE**
<formats>:	**DEFault** | **LEN** | **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**
<options>:	**FILE** | **FlowTrace** | **BusTrace** | **MACHINE** | **NorthWestGravity** | **Mark** | **Track**

Shows program nesting with instruction.

<options>	For a detailed description of all other parameters and options, refer to the **<trace>.List** command.

See also

- MIPS

See also

- <trace>.PROfileChart.TASKVSINTERRUPT
- MIPS.PROfileChart.ALL
- MIPS.PROfileChart.DistriB
- MIPS.PROfileChart.Line
- MIPS.PROfileChart.PROGRAM
- MIPS.PROfileChart.sYmbol
- MIPS.PROfileChart.TASKINFO
- MIPS.PROfileChart.TASKKernel
- MIPS.PROfileChart.TASKSRV
- MIPS.PROfileSTATistic
- MIPS.STATistic
- MIPS.PROfileChart.AddressGROUP
- MIPS.PROfileChart.DatasYmbol
- MIPS.PROfileChart.GROUP
- MIPS.PROfileChart.MODULE
- MIPS.PROfileChart.RWINST
- MIPS.PROfileChart.TASK
- MIPS.PROfileChart.TASKINTR
- MIPS.PROfileChart.TASKORINTERRUPT
- MIPS.PROfileChart.TASKVSINTR
- MIPS

MIPS.PROfileChart.AddressGROUP MIPS profile chart for address groups

Format: **MIPS.PROfileChart.AddressGROUP** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The time for of the items pooled by the **GROUP.Create** command is displayed as MIPS chart. The results include groups for both program and data.

<trace_area> Refer to **Trace.PROfileChart.AddressGROUP**.
<option>

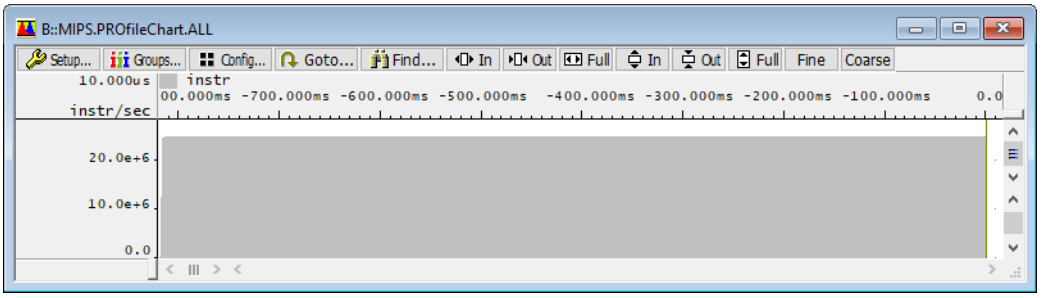
See also

- MIPS.PROfileChart
- MIPS.PROfileChart.GROUP

Format: **MIPS.PROfileChart.ALL** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The time for the program execution is displayed as MIPS chart.



See also

- MIPS.PROfileChart

MIPS.PROfileChart.DatasYmbol MIPS profile chart for pointer

Format: **MIPS.PROfileChart.DatasYmbol** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays a MIPS chart for debug symbols with addresses corresponding to the data accessed in the trace.

See also

- MIPS.PROfileChart

Format:	**MIPS.PROfileChart.DistriB** [*<trace_area>*] [*/<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows a graphical representation of the specified trace item as a percentage of a time slice.

See also

- MIPS.PROfileChart

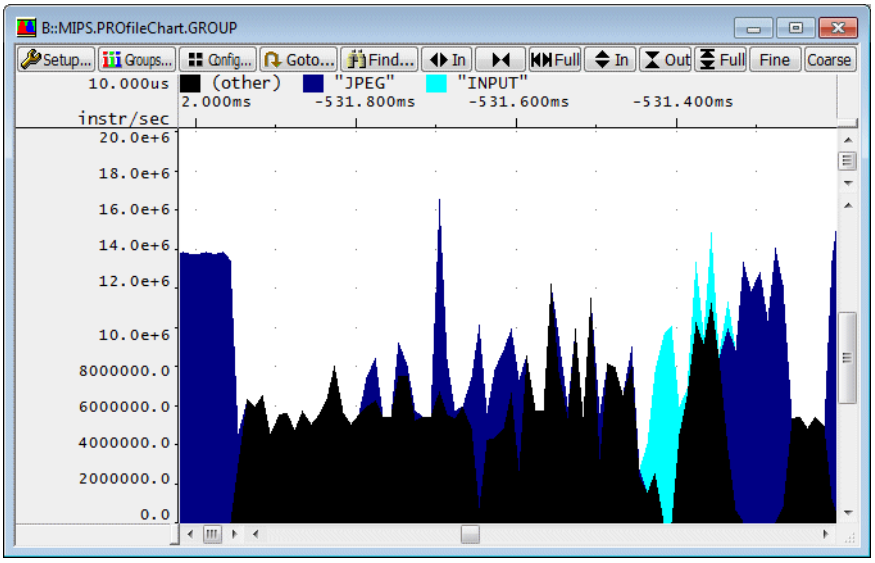
Format: **MIPS.PROfileChart.GROUP** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws MIPS profile charts for **groups**. The results only include groups within the program range. Groups for data addresses are not included.

<trace_area> Refer to **Trace.PROfileChart.GROUP**.
<option>

```
; create group "INPUT", add the modules \jquant2 \jdinput \jidctred and
; assign the color AQUA to this group
GROUP.CreateModules "INPUT" \jquant2 \jdinput \jidctred /AQUA

; create another group
GROUP.CreateModules "JPEG" …

; start and stop the program execution

; display the result
MIPS.PROfileChart.GROUP
```



See also

- MIPS.PROfileChart
- MIPS.PROfileChart.AddressGROUP

MIPS.PROfileChart.Line MIPS per high-level language line graphically

Format: **MIPS.PROfileChart.Line** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws a MIPS graph for all executed high-level code lines.

<trace_area> Refer to **Trace.PROfileChart.Line**.

<option>

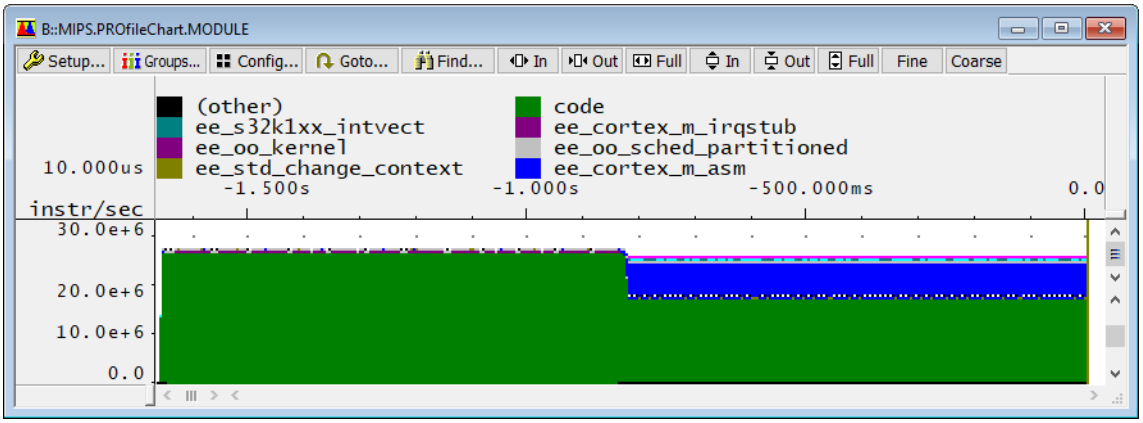
See also

- MIPS.PROfileChart

Format: **MIPS.PROfileChart.MODULE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws MIPS profile charts for symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.



<trace_area> Refer to **Trace.PROfileChart.MODULE**.

<option>

See also

- MIPS.PROfileChart

Format: **MIPS.PROfileChart.MODULE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws MIPS profile charts for loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area> Refer to **Trace.PROfileChart.PROGRAM**.
<option>

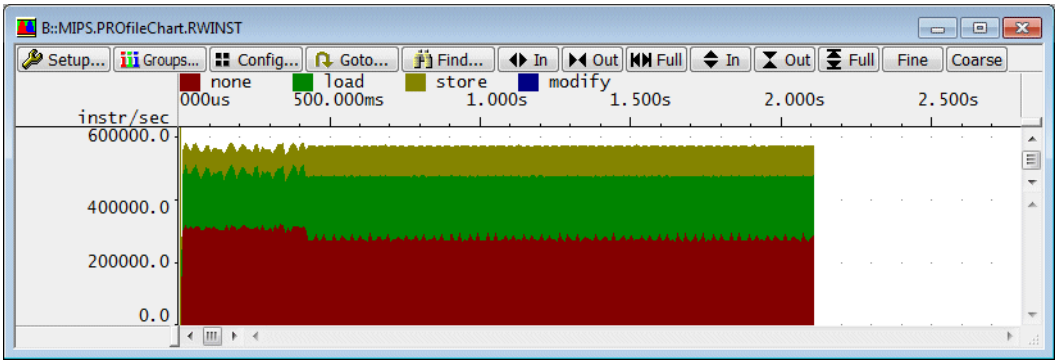
See also

- MIPS.PROfileChart

MIPS.PROfileChart.RWINST MIPS per cycle type graphically

Format: **MIPS.PROfileChart.RWINST**

Draws a MIPS graph for read, write, modify instruction and all others (none) instructions.



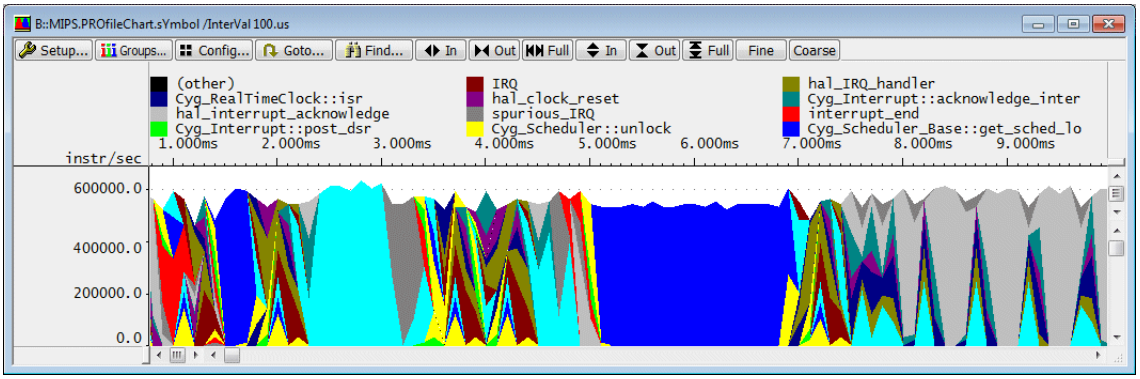
See also

- MIPS.PROfileChart

Format:	**MIPS.PROfileChart.sYmbol** [*<trace_area>*] [*/<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws a MIPS graph for all program symbols.

<trace_area> Refer to **Trace.PROfileChart.sYmbol**.
<option>



See also

- MIPS.PROfileChart

Format: **MIPS.PROfileChart.TASK** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws a MIPS graph for all tasks. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASK**.
<option>

See also

- MIPS.PROfileChart

MIPS.PROfileChart.TASKINFO MIPS for data trace via context ID

Format: **MIPS.PROfileChart.TASKINFO** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws a MIPS graph for special messages written to the Context ID register for ETM trace. Please refer to **Trace.PROfileChart.TASKINFO** for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINFO**.
<option>

See also

- MIPS.PROfileChart

Format: **MIPS.PROfileChart.TASKINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays a MIPS profile chart for ORTI based ISR2. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINTR**.
<option>

See also

- MIPS.PROfileChart

MIPS.PROfileChart.TASKKernel MIPS profile chart with kernel marker

Format: **MIPS.PROfileChart.TASKKernel** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Refines the command **MIPS.PROfileChart.TASK** for RTOS systems that don't assign a task ID to the kernel. Refer to **Trace.STATistic.TASKKernel** for more information.

This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASKKernel**.
<option>

See also

- MIPS.PROfileChart

Format: **MIPS.PROfileChart.TASKORINTERRUPT** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Draws a MIPS graph for all tasks and interrupts. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASKORINTERRUPT**.
<option>

See also

- MIPS.PROfileChart

MIPS.PROfileChart.TASKSRV MIPS profile chart for OS service routines

Format: **MIPS.PROfileChart.TASKSRV** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The time spent in OS service routines and different tasks is displayed as MIPS profile chart. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKSRV**.
<option>

See also

- MIPS.PROfileChart

Format: **MIPS.PROfileChart.TASKVSINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays a MIPS profile chart for task-related interrupt service routines. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINTR**.

<option>

See also

- MIPS.PROfileChart

See also

- <trace>.PROfileSTATistic.TASKVSINTERRUPT
- MIPS.PROfileChart
- MIPS.PROfileSTATistic.AddressGROUP
- MIPS.PROfileSTATistic.DatasYmbol
- MIPS.PROfileSTATistic.GROUP
- MIPS.PROfileSTATistic.Line
- MIPS.PROfileSTATistic.PROGRAM
- MIPS.PROfileSTATistic.RWINST
- MIPS.PROfileSTATistic.TASK
- MIPS.PROfileSTATistic.TASKINTR
- MIPS.PROfileSTATistic.TASKORINTERRUPT
- MIPS.STATistic
- MIPS
- MIPS.PROfileSTATistic.Address
- MIPS.PROfileSTATistic.ALL
- MIPS.PROfileSTATistic.DistriB
- MIPS.PROfileSTATistic.INTERRUPT
- MIPS.PROfileSTATistic.MODULE
- MIPS.PROfileSTATistic.RUNNABLE
- MIPS.PROfileSTATistic.sYmbol
- MIPS.PROfileSTATistic.TASKINFO
- MIPS.PROfileSTATistic.TASKKernel
- MIPS.PROfileSTATistic.TASKSRV

MIPS.PROfileSTATistic.Address MIPS per address as profile statistic

Format: **MIPS.PROfileSTATistic.Address** [*<trace_area>*] *<address1>* [*<address2>* ...] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic for addresses.

<trace_area> Refer to **Trace.PROfileSTATistic.Address**.

<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.AddressGROUP** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic for the items pooled by the **GROUP.Create** command. The results include groups for both program and data.

<trace_area> Refer to **Trace.PROfileSTATistic.AddressGROUP**.
<option>

See also

- MIPS.PROfileSTATistic
- ETA.PROfileSTATistic.GROUP

MIPS.PROfileSTATistic.ALL MIPS profile statistic for program run

Format: **MIPS.PROfileSTATistic.ALL** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic for the program run.

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.DatasYmbol** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays a MIPS profile statistic for debug symbols with addresses corresponding to the data accessed in the trace.

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.DistriB Distribution statistical analysis

Format: **MIPS.PROfileSTATistic.DistriB** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic for or the statistical distribution of a selected item or based on the symbolic addresses if no item is specified.

<trace_area> Refer to **Trace.PROfileSTATistic.DistriB**.
<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.GROUP** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic for **groups**. The results only include groups within the program range. Groups for data addresses are not included.

<trace_area> Refer to **Trace.PROfileSTATistic.GROUP**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.INTERRUPT MIPS per interrupt as table

Format: **MIPS.PROfileSTATistic.INTERRUPT** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for interrupts. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.INTERRUPT**.
<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.Line** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for high-level language lines.

<trace_area> Refer to **Trace.PROfileSTATistic.Line**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.MODULE MIPS per module as profile statistic

Format: **MIPS.PROfileSTATistic.MODULE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<trace_area> Refer to **Trace.PROfileSTATistic.MODULE**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.PROGRAM MIPS per program as profile statistic

Format: **MIPS.PROfileSTATistic.PROGRAM** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area> Refer to **Trace.PROfileSTATistic.PROGRAM**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.RUNNABLE MIPS per runnable as table

Format: **MIPS.PROfileSTATistic.RUNNABLE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for AUTOSAR runnables. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.RUNNABLE**.
<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.RWINST**

MIPS for read, write, modify instruction and all others (none) instructions as table.

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.sYmbol MIPS for all program symbols as table

Format: **MIPS.PROfileSTATistic.sYmbol** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for all program symbols.

<trace_area> Refer to **Trace.PROfileSTATistic.sYmbol**.
<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.TASK** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for tasks. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASK**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.TASKINFO MIPS for data trace via context ID

Format: **MIPS.PROfileSTATistic.TASK** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for special messages written to the Context ID register for ETM trace. Refer to **Trace.PROfileSTATistic.TASKINFO** fore more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKINFO**.
<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.TASKINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for ORTI based ISR2. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKINTR**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.TASKKernel MIPS per task as table

Format: **MIPS.PROfileSTATistic.TASKKernel** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Refines the command **MIPS.PROfileSTATistic.TASK** for RTOS systems that don't assign a task ID to the kernel. Refer to **Trace.STATistic.TASKKernel** for more information. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKKernel**.
<option>

See also

- MIPS.PROfileSTATistic

Format: **MIPS.PROfileSTATistic.TASKORINTERRUPT** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays MIPS as profile statistic table for tasks and interrupts. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKORINTERRUPT**.
<option>

See also

- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.TASKSRV MIPS per OS service routine as table

Format: **MIPS.PROfileSTATistic.TASKSRV** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The time spent in OS service routines and different tasks is displayed as MIPS profile statistic table. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKSRV**.
<option>

See also

- MIPS.PROfileSTATistic

See also

- <trace>.STATistic.TASKVSINTERRUPT
- MIPS.PROfileChart
- MIPS.STATistic.ALL
- MIPS.STATistic.DistriB
- MIPS.STATistic.GROUP
- MIPS.STATistic.MODULE
- MIPS.STATistic.PROGRAM
- MIPS.STATistic.sYmbol
- MIPS.STATistic.TASKINFO
- MIPS.STATistic.TASKKernel
- MIPS.STATistic.TREE
- MIPS
- MIPS.PROfileSTATistic
- MIPS.STATistic.ChildTREE
- MIPS.STATistic.Func
- MIPS.STATistic.LINKage
- MIPS.STATistic.ParentTREE
- MIPS.STATistic.RWINST
- MIPS.STATistic.TASK
- MIPS.STATistic.TASKINTR
- MIPS.STATistic.TASKSRV

MIPS.STATistic.ALL MIPS for the program run

Format: **MIPS.STATistic.ALL** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays MIPS for the program run numerically.

See also

- MIPS.STATistic

MIPS.STATistic.ChildTREE MIPS for the callee context of a function

Format: **MIPS.STATistic.ChildTREE** *<address>* [*<list_items>*] [**/***<option>*]

Displays MIPS for all functions called by the specified function numerically as call tree.

Refer to **<trace>.STATistic.ChildTREE** for a description of the parameters and options.

See also

- MIPS.STATistic

Format: **MIPS.STATistic.DistriB** [**%***<format>*] [*<items>* …] [**/***<option>*]

Displays MIPS for the statistic distribution of any data is displayed if *<item>* is specified. Displayed are the without *<item>* the statistic is based on the symbolic addresses.

Refer to **<trace>.STATistic.DistriB** for a description of the parameters and options.

See also

- MIPS.STATistic

MIPS.STATistic.Func MIPS for functions numerically

Format: **MIPS.STATistic.Func** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays MIPS for functions numerically.

Refer to **<trace>.STATistic.Func** for a description of the parameters and options.

See also

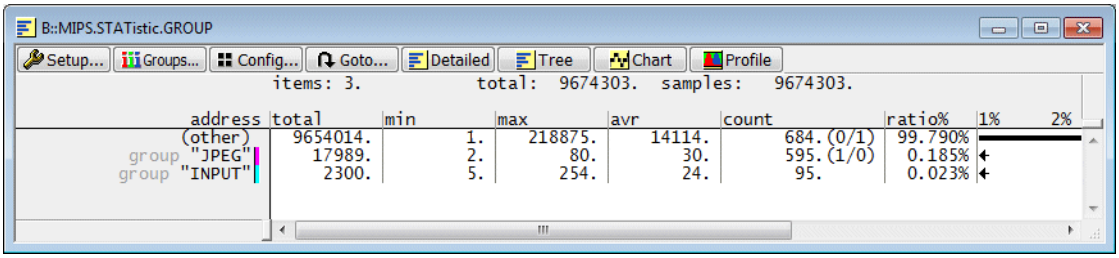
- MIPS.STATistic

MIPS.STATistic.GROUP MIPS statistic for groups

Format: **MIPS.STATistic.GROUP** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

MIPS statistic for **groups**.

Refer to **<trace>.STATistic.GROUP** for a description of the parameters and options.



See also

- MIPS.STATistic

MIPS.STATistic.LINKage — Per caller MIPS statistic of function

Format: **MIPS.STATistic.LINKage** [**%***<format>*] [*<items>* …] [*/<option>*]

Performs a function MIPS statistic for a single function itemized by its callers.

Refer to **<trace>.STATistic.LINKage** for a description of the parameters and options.

See also

- MIPS.STATistic

MIPS.STATistic.MODULE — MIPS for modules numerically

Format: **MIPS.STATistic.MODULE** [**%***<format>*] [*<list_items>* …] [*/<option>*]

Displays MIPS for symbol modules numerically. The list of loaded modules can be displayed with **sYmbol.List.Module**.

Refer to **<trace>.STATistic.MODULE** for a description of the parameters and options.

See also

- MIPS.STATistic

Format: **MIPS.STATistic.ParentTREE** *<address>*

Show call tree and MIPS of all callers of the specified function. The function is specified by its start *<address>*.

Refer to **<trace>.STATistic.ParentTREE** for a description of the parameters and options.

See also

- MIPS.STATistic

MIPS.STATistic.PROGRAM MIPS for programs numerically

Format: **MIPS.STATistic.PROGRAM** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays MIPS for loaded object file programs numerically. The loaded programs can be displayed with the command **sYmbol.Browse ***.

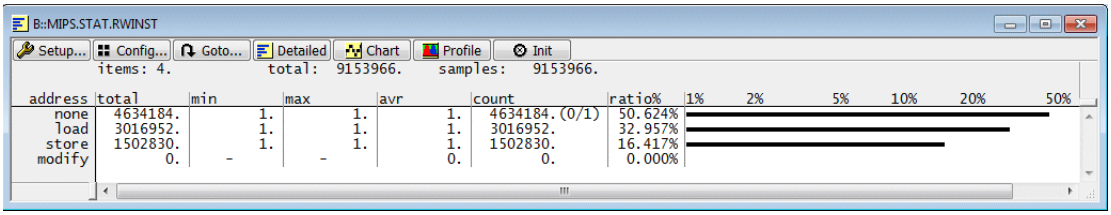
Refer to **<trace>.STATistic.PROGRAM** for a description of the parameters and options.

See also

- MIPS.STATistic

Format: **MIPS.STATistic.RWINST** [%*<format>*] [*<list_items>* …] [*/<option>*]

MIPS per read, write, modify instruction and all others (none) instructions in a numerical statistic.



See also

- MIPS.STATistic

MIPS.STATistic.sYmbol MIPS for all program symbols numerically

Format: **MIPS.STATistic.sYmbol** [%*<format>*] [*<list_items>* …] [*/<option>*]

MIPS for all program symbols numerically.

Refer to **<trace>.STATistic.sYmbol** for a description of the parameters and options.

See also

- MIPS.STATistic

MIPS.STATistic.TASK MIPS per task numerically

Format: **MIPS.STATistic.TASK** [%*<format>*] [*<list_items>* …] [*/<option>*]

MIPS per task numerically. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to **<trace>.STATistic.TASK** for a description of the parameters and options.

See also

- MIPS.STATistic

Format: **MIPS.STATistic.TASKINFO** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays a MIPS statistic for special messages written to the Context ID register for ETM trace. Please refer to **Trace.STATistic.TASKINFO** for more information.

See also

- MIPS.STATistic

MIPS.STATistic.TASKINTR MIPS per ISR2 numerically

Format: **MIPS.STATistic.TASKINTR** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

MIPS per ORTI based ISR2 numerically. This feature is only available, if an OSEK/ORTI system is used, and if the OS Awareness is configured with the **TASK.ORTI** command.

Refer to **<trace>.STATistic.TASKINTR** for a description of the parameters and options.

See also

- MIPS.STATistic

MIPS.STATistic.TASKKernel MIPS task analysis with kernel markers

Format: **MIPS.STATistic.TASKernel** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Refines the command **MIPS.STATistic.TASK** for RTOS systems that don't assign a task ID to the kernel. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to **<trace>.STATistic.TASKKernel** for a description of the parameters and options.

See also

- MIPS.STATistic

Format: **MIPS.STATistic.TASKSRV** [%*<format>*] [*<list_items>* …] [**/***<option>*]

MIPS per OS service routine numerically. This feature is only available, if an OSEK/ORTI system is used, and if the OS Awareness is configured with the **TASK.ORTI** command.

Refer to **<trace>.STATistic.TASKSRV** for a description of the parameters and options.

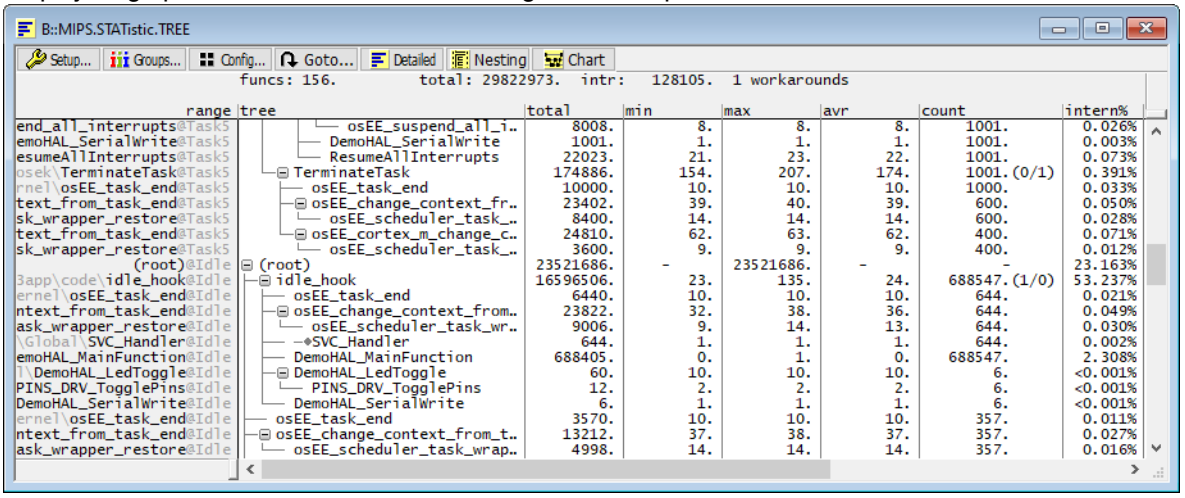
See also

- MIPS.STATistic

MIPS.STATistic.TREE Tree display of nesting functions with MIPS

Format: **MIPS.STATistic.TREE** [%*<format>*] [*<list_items>* …] [**/***<option>*]

Displays a graphical tree of the function nesting with MIPS per function results.



range	tree	total	min	max	avr	count	intern%
end_all_interrupts@Task5	osEE_suspend_all_i…	8008.	8.	8.	8.	1001.	0.026%
emoHAL_SerialWrite@Task5	DemoHAL_SerialWrite	1001.	1.	1.	1.	1001.	0.003%
esumeAllInterrupts@Task5	ResumeAllInterrupts	22023.	21.	23.	22.	1001.	0.073%
osek\TerminateTask@Task5	TerminateTask	174886.	154.	207.	174.	1001.(0/1)	0.391%
rnel\osEE_task_end@Task5	osEE_task_end	10000.	10.	10.	10.	1000.	0.033%
text_from_task_end@Task5	osEE_change_context_fr…	23402.	39.	40.	39.	600.	0.050%
sk_wrapper_restore@Task5	osEE_scheduler_task_…	8400.	14.	14.	14.	600.	0.028%
text_from_task_end@Task5	osEE_cortex_m_change_c…	24810.	62.	63.	62.	400.	0.071%
sk_wrapper_restore@Task5	osEE_scheduler_task_…	3600.	9.	9.	9.	400.	0.012%
(root)@Idle	(root)	23521686.	-	23521686.	-	-	23.163%
3app\code\idle_hook@Idle	idle_hook	16596506.	23.	135.	24.	688547.(1/0)	53.237%
ernel\osEE_task_end@Idle	osEE_task_end	6440.	10.	10.	10.	644.	0.021%
ntext_from_task_end@Idle	osEE_change_context_from…	23822.	32.	38.	36.	644.	0.049%
ask_wrapper_restore@Idle	osEE_scheduler_task_wr…	9006.	9.	14.	13.	644.	0.030%
\Global\SVC_Handler@Idle	SVC_Handler	644.	1.	1.	1.	644.	0.002%
emoHAL_MainFunction@Idle	DemoHAL_MainFunction	688405.	0.	1.	0.	688547.	2.308%
l\DemoHAL_LedToggle@Idle	DemoHAL_LedToggle	60.	10.	10.	10.	6.	<0.001%
PINS_DRV_TogglePins@Idle	PINS_DRV_TogglePins	12.	2.	2.	2.	6.	<0.001%
DemoHAL_SerialWrite@Idle	DemoHAL_SerialWrite	6.	1.	1.	1.	6.	<0.001%
ernel\osEE_task_end@Idle	osEE_task_end	3570.	10.	10.	10.	357.	0.011%
ntext_from_task_end@Idle	osEE_change_context_from_t…	13212.	37.	38.	37.	357.	0.027%
ask_wrapper_restore@Idle	osEE_scheduler_task_wrap…	4998.	14.	14.	14.	357.	0.016%

Refer to **<trace>.STATistic.TREE** for a description of the parameters and options.

See also

- MIPS.STATistic

MMU

Memory management unit

See also

- MMU.DUMP
- MMU.FORMAT
- MMU.INFO
- MMU.List
- MMU.MemAnalysis
- MMU.PageTable
- MMU.SCAN
- MMU.Set
- MMU.TDUMP
- MMU.TSCAN
- MMU.view
- TASK.CONFIG
- TRANSlation

- ▲ 'CPU specific MMU Commands' in 'CPU32/ColdFire Debugger and Trace'
- ▲ 'CPU specific MMU Commands' in 'Arm Debugger'
- ▲ 'CPU specific MMU Commands' in 'Armv8 and Armv9 Debugger'
- ▲ 'CPU specific MMU Commands' in 'Intel® x86/x64 Debugger'
- ▲ 'MMU Functions (Memory Management Unit)' in 'General Function Reference'
- ▲ 'Release Information' in 'Legacy Release History'

Overview MMU

Using the **MMU** command group, you can access and view the configuration of the memory management unit of the current core.

If supported by the CPU, you can view the logical-to-physical address translations denoted in the MMU TLBs (translation look-aside buffers) and page tables.

If you have loaded an OS Awareness to TRACE32 with the **TASK.CONFIG** command, you can also view the task page table associated with each running process of the OS.

What is the difference between the command groups...?

MMU	TRANSlation
Lets you access and view the real hardware MMU.	Configures and controls the TRACE32 internal debugger address translation. This feature is used to mimic the translations within the real hardware MMU so that the debugger can access code and data of any OS process at any time.

[Examples]

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>*
<cpu_specific_tables>:	**ITLB** | **DTLB** | **TLB** (e.g. for ARM, MIPS) **PTE** | **BAT** (e.g. for MPC8260, MPC750) **TLB0** | **TLB1** (e.g. for MPC54xx, MPC85xx)

Displays the contents of the MMU translation table or a CPU specific TLB table.

- If the command is called without parameters, the complete current page table will be displayed; i.e., in this case **MMU.DUMP** is equivalent to **MMU.DUMP PageTable**.
- 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.

<table>	For descriptions of **PageTable**, **KernelPageTable**, etc, see **MMU.List**.
<cpu_specific_tables>	Many command arguments are target processor specific. For details, refer to the **Processor Architecture Manuals** [▲] listed in the **See also** block below.

Description of Columns in the MMU.DUMP.PageTable Window

These columns are available for all architectures.

Column name	Description
logical	Logical page address range
physical	Physical page address range
size	Size of mapped page in bytes
tablewalk	Details of table walk for logical page address (one sub-column for each table level, showing the table base address, entry index, entry width in bytes and value of table entry).

All other columns in the **MMU.DUMP PageTable** window are architecture specific.

Examples:

```
;dump entries of current page table in specified range
;           <table>            <range>
MMU.DUMP   PageTable    0xC0000000--0xDFFFFFFF
```

```
;display PT with physical base address A:0x00402100
;dump entries with logical address >= 0xC0000000
;           <table>     <start_address>         <root>
MMU.DUMP   PageTable      0xC0000000         A:0x00402100
```

```
;open the TASK.List.tasks window to display task name, task magic number,
;and space ID
TASK.List.tasks

;The format of the MMU.DUMP.TaskPageTable argument governs how it is
;interpreted:

MMU.DUMP TaskPageTable "ash"        ;task name
MMU.DUMP TaskPageTable 0xC707AE20   ;task magic number in hex

MMU.DUMP TaskPageTable 673.         ;space ID in decimal

;the same space ID in hex, specified as an extended address.
;Only the space ID part of the address is relevant for this command.
MMU.DUMP TaskPageTable 0x2A1:0x0
```

See also

■ MMU ■ MMU.view ■ TRANSlation.TlbAutoScan

▲ 'CPU specific MMU commands' in 'MPC5xx/8xx Debugger and Trace'
▲ 'Release Information' in 'Legacy Release History'

Format:	**MMU.FORMAT** *<format>* [*<base_address>* [*<logical_kernel_address_range>* *<physical_kernel_address>*]] [*/<option>*]
<option>:	**MACHINE** *<machine_id>* (Arm, i386 and PowerPC64 only) **Intermediate** (Arm, i386 and PowerPC64 only) **SPACEID** [**TTB0** | **TTB1**] (Arm only)

Default *<format>*: STD.

Defines the information needed for the page table walks, which are performed by TRACE32 for debugger address translation, page table dumps, or page table scans.

<format>

<format> is to be replaced with a CPU architecture specific keyword which defines the structure of the MMU page tables used by the kernel. By default, TRACE32 assumes that the MMU format is **STD**, unless *you* specify the **MMU.FORMAT** *<format>* explicitly.

- In addition to **STD**, some CPU architectures have further *<format>* keywords. For an overview of these keywords, please refer to **"Appendix - <format> Options of MMU.FORMAT"**, page 136.
- For more information about the declaration of the debugger address translation, refer to the **"OS Awareness Manuals"**. These manuals also list the OS specific *<format>* keywords broken down by CPU architecture.

<base_address>

<base_address> defines the start address of the default page table which is usually the kernel page table. The kernel page table contains translations for mapped address ranges owned by the kernel.

The debugger address translation uses the default page table if no process specific page table (task page table) is available to translate an address.

<base_address> can be left empty by typing a comma or set to zero if there is no default page table available in the system.

NOTE: Additional **MMU.FORMAT** input parameters may be required for some architecture specific *<format>* keywords. This is described in **"Appendix - <format> Options of MMU.FORMAT"**, page 136.

<logical_kernel_address_range> and <physical_kernel_address> for the Default Translation

The arguments *<logical_kernel_address_range>* and *<physical_kernel_address>* define a linear logical-to-physical address translation for the kernel addresses, called *kernel translation* or *default translation*. This translation should cover all statically mapped logical address ranges of kernel code or kernel data.

For the *<physical_kernel_address>* you just need to specify the start address.

NOTE:	If no kernel translation is specified for a given memory access, TRACE32 tries to use static address translations defined by the command **TRANSlation.Create**. The kernel translation is shown in the **TRANSlation.List** window.

Example

To enable the debugger address translation with page table walk, please use **TRANSlation.ON** and **TRANSlation.TableWalk ON** after specifying **MMU.FORMAT**. This example shows a typical **MMU.FORMAT** declaration for Linux:

```
; enable space ID usage, needed for Linux
SYStem.Option.MMUSPACES ON

; these are the arguments used for the MMU.FORMAT example:
; 1. use MMU page table format LINUX for page table walks
; 2. use symbol swapper_pg_dir to define the base address of the
;    kernel page table
; 3. define the kernel translation:
;      translate the logical kernel address range 0xc0000000--0xffffffff
;      to the physical address range 0x0--0x1fffffff
MMU.FORMAT LINUX swapper_pg_dir 0xc0000000--0xffffffff 0x0

; define the common kernel address range
TRANSlation.COMMON 0xc0000000--0xffffffff

; enable the table walk and the debugger address translation
TRANSlation.TableWalk ON
TRANSlation.ON
```

The known page tables can be viewed with commands **TRANSlation.List.***<page_table>* or **TRANSlation.DUMP.***<page_table>* where *<page_table>* specifies the page table to be viewed - please see the architecture specific documentation of these two commands in the **"Processor Architecture Manuals"**.

MACHINE Available if **SYStem.Option.MACHINESPACES** is set to ON.

Intermediate Specifies translation information for the translation of intermediate physical (IPA) to physical addresses (PA).

If this option is set, *<base_address>* is the page table base address used for the translation of IPA to PA.
<logical_kernel_address_range> and *<physical_kernel_address>* specify a simple default translation from IPA to PA.

SPACEID Configures whether task page tables are held by MMU register TTBR0 or TTBR1.

TTB0: TTBR0 holds the task page table, TTBR1 holds the fixed kernel page table (DEFAULT).
TTB1: TTBR1 holds the task page table, TTBR0 holds the fixed kernel page table

With **SYStem.Option.MMUSPACES ON**, TRACE32 uses the space ID of an address to select the task page table associated with the space ID.

If you specify "MMU.FORMAT ... /SPACEID TTBR1", TRACE32 will treat register TTBR1 as space ID dependent task page table and TTBR0 as fixed kernel page table.
Also, with /SPACEID TTBR1, the default page table *<base_address>* will be used in place of TTBR0 instead of TTBR1.

As **MMU.FORMAT** is a zone specific and/or machine specific command, option /SPACEID can be configured per-zone/per-machine.

NOTE: The error message "**INVALID COMBINATION**" will be shown if a Linux-related MMU format such as **LINUX**, **LINUXBIG**, … is specified without a previous **SYStem.Option.MMUSPACES ON** command. Linux page table handling requires space IDs to be enabled in TRACE32.

See also

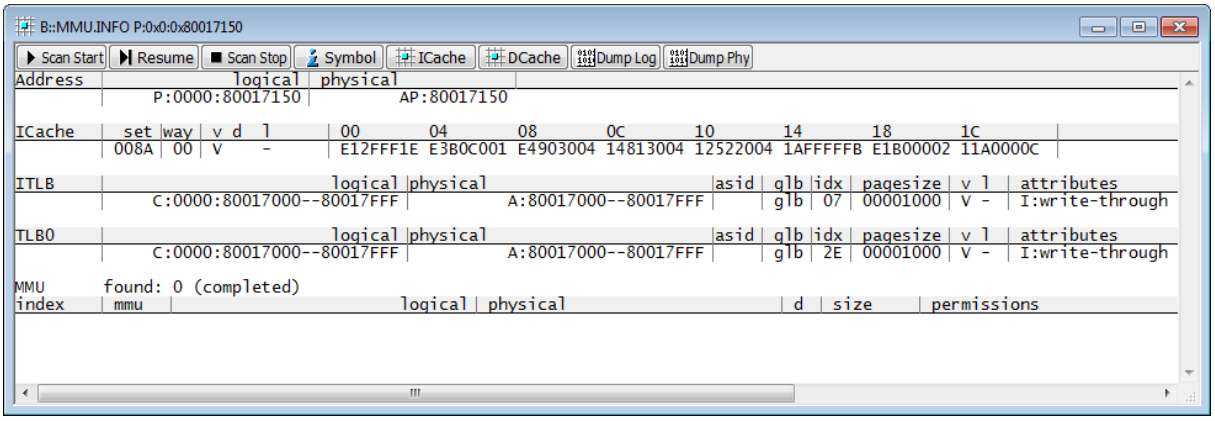
- MMU
- MMU.view
- TRANSlation
- TRANSlation.COMMON
- TRANSlation.Create
- TRANSlation.List
- TRANSlation.TableWalk
- TRANSlation.TlbAutoScan
- MMU.DEFAULTPT()
- MMU.FORMAT()

▲ 'Release Information' in 'Legacy Release History'

Format:	**MMU.INFO.**<*sub_cmd*> <*address*>
<*sub_cmd*>:	**create** | **scanSTART** | **scanRESUME** | **scanSTOP**

Displays all translation information related to a physical address. If the address is a logical address, TRACE32 first translates it into a physical address. The information contains:

- All cache lines that cache the physical address, including both instruction and data cache.
- All TLB entries that contain translation rules for the physical address.
- All mmu entries that contain translation rules for the physical address (or all pages mapped to the given physical address), including both the task and kernel MMU entries.



Description of Toolbar Buttons in the MMU.INFO.<sub_cmd> Window

create	Views all translation information related to an address.
scanSTART	Starts a scan in all MMU page tables for entries that contain translation rules for the physical address.
scanRESUME	Resumes the scan stopped with **scanSTOP**.
scanSTOP	Stops the scan.

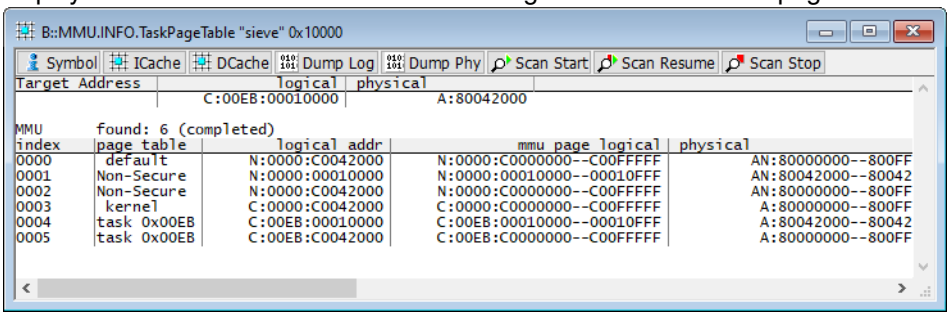
See also

■ MMU ■ MMU.view ■ sYmbol.INFO

▲ 'Release Information' in 'Legacy Release History'

Format:	**MMU.INFO.TaskPageTable** *<address> <task>*
<task>:	*<task_magic>* | *<task_id>* | *<task_name>*

displays all translation information related to a give address and task page table.



<task_magic>, etc.	See also **"What to know about the Task Parameters"** (general_ref_t.pdf).

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

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.

<cpu_specific> tables and options	Many command arguments are target processor specific. For details, see **Processor Architecture Manuals** [▲] listed in the **See also** block below.
<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 or a specific machine if a space ID and/or a machine ID is given.
<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.
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.

PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID and/or machine ID: list the translation table of the specified process and/or machine • else, this command lists the table the CPU currently uses for MMU translation.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Lists the MMU translation of 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**.

See also

■ MMU ■ MMU.view

▲ 'Release Information' in 'Legacy Release History'

MMU.MemAnalysis Analyze page tables

Format: **MMU.MemAnalysis** *<sub_cmd>*

<sub_cmd>:
START
SAVE *<export_file_name> <import_file_name>*
SET MaxBlocksize *<value>*
SET mergeLevel *<value>*
AddMask *<address_range>*
ClearMasks

The **MMU.MemAnalysis** feature helps in the process of saving an image of a target's physical memory content to binary files so that it can later be restored in a TRACE32 simulator.

Saving the complete physical memory can take a very long time, so **MMU.MemAnalysis** helps by analyzing MMU page table data, finally creating a custom PRACTICE *export* script containing **Data.SAVE.Binary** commands which save only the physical address ranges that are referred to by page table entries. Unused memory will not be saved. Further, a PRACTICE *import* script will be written too which can be used to quickly import the saved binary data files in a TRACE32 simulator.

START

Clear internal MemAnalysis database and start a new collection of page table entries for analysis. After MMU.MemAnalysis.START Enable option /ANALYZE in
commands **MMU.SCAN** and **MMU.SCAN.ALL** (the latter is only available for ARM and PowerPC debuggers).

SAVE
<export_script>
<import_script>

Analyze all saved page table entries, apply the filters, merge neighboring physical address ranges and write the resulting PRACTICE *<export_script>* and *<import_script>*.
1. the *<export_script>* script can be used to save the physical memory used by the target to binary files using **Data.SAVE.Binary** commands. Execute this script on your target debugger. This script also contains comments about the page tables containing references to the address ranges saved.
2. the *<import_script>* will, when executed on a TRACE32 simulator, load all binary files saved in step 1. to the simulator memory and recreate the physical memory content of the original target.

Both scripts can be executed with the DO command. For the export script is it possible specify a destination path where the binary files shall be saved to. This path must be enclosed in quotes, see the example below.

SET MaxBlocksize
<max>

Control the analysis of scanned page table entries: ignore page table entries with a size larger than *<max>.*

This setting is useful to remove individual page table entries which cover a large part or all of the total available physical memory. Kernels typically add such large “memory windows” to the kernel page table so that they can easily access arbitrary addresses, but such large entries render the analysis of all other small page table entries useless.

SET mergeLevel
<level>

Balance the number of **Data.SAVE.Binary** commands created in the export script versus the total amount of data saved.

<level> is a decimal number ranging between 0. and 52. (don't forget the period after the decimal number).

If the merge level is set to 0., the least possible amount of data is saved. However, if data is very fractioned among the physical memory, this may end up in a large number of **Data.SAVE.Binary** commands in the export script.

A larger merge level means that neighboring address ranges will be merged together, reducing the number of **Data.SAVE.Binary** commands will needed. The unused data in between them will be saved too, increasing the total time needed to save the data.

AddMask *<address_range>*
: Mask out one or multiple physical address ranges so that they are not saved in the export script.

  This is useful if page tables contain mappings of physical addresses which belong to memory mapped peripheral devices. Debugger access to such device addresses may cause the target to stall or crash. It is better to add a mask for the complete physical address range of all peripherals, so the export script will not save them with **Data.SAVE.Binary** commands.

ClearMasks
: Remove all masks set with **AddMask**.

The analysis begins with an MMU.MemAnalysis.START command which clears the internal analyis database and enables the option **/ANALYZE** for the **MMU.SCAN** and **MMU.SCAN.ALL** commands. The database is filled by executing any combination of **MMU.SCAN** *<page_table>* **/ANALYZE** commands until all page tables have been scanned. On ARM and PowerPC targets, **MMU.SCAN.ALL /ANALYZE** can be used to automatically scan all page tables known to TRACE32. This includes all task page tables of a running operation system. In case a hypervisor with one or multiple guest operation systems is running on the target, all page tables of all tasks on all machines will be scanned. Note that an OS Awareness and, for hypervisors, a Hypervisor Awareness must be loaded to TRACE32.

After the page tables have been scanned into the internal database, address ranges can be excluded from the analysis using **MMU.MemAnalysis.AddMask**. Some kernels create single large page table entries in the kernel page table for easy access to the full physical memory content. To prevent that such very large address “windows” render the analysis useless, they can be exempt from the analysis using **MMU.MemAnalysis.SET MaxBlocksize**.

During the MMU.MemAnalysis.SAVE process, all physical address ranges found in the internal database are sorted and merged, preserving the information which page tables have contributed to a physical address range. The *export* PRACTICE script will contain a list of **Data.SAVE.Binary** commands along with informative comments. When executing this script, the target’s physical memory will be saved to files. It is possible to specify a destination path name, enclosed in quotes, to which the binary files will be saved.

If the used physical address ranges are very fragmented, a large number of binary files may have to be saved. Using **MMU.MemAnalysis.SET mergeLevel <level>**, the total amount of data saved can be balanced versus the number of binary files saved.

NOTE:

- To restore a target state in a TRACE32 simulator, at least all used memory and the register content must be saved on the target and later restored in the simulator. See **STOre <file> Register** for more information about storing and restoring registers.
- If the kernel uses a single, locked TLB entry instead of a kernel page table for access to kernel data, MMU.MemAnalysis will not include the kernel data in the saved address space. In this case, you must add a **Data.SAVE.Binary** command for the kernel address manually to the export script.

Example:

```
; Initially, boot your target wait until the kernel has filled the page
tables.
; If necessary, do the TRACE32 MMU configuration with MMU.FORMAT,
TRANSlation.Create and TRANSlation.COMMON

; 1. start the MemAnalysis with MMU.Memanalysis.START
; Now, the /ANALYZE option in the MMU.SCAN.<page table> and MMU.SCANALL
commands are enabled
MMU.MemAnalysis START

; 2. fill the analysis database: scan all page tables you want to include
in the resulting (physical) memory analysis.
;    Either use
MMU.SCAN ALL /ANALYZE
; if supported on your target or use a combination of multiple MMU.SCAN
commands such as
; MMU.SCAN EL1PT /ANALYZE
; MMU.SCAN EL2PT /ANALYZE
; MMU.SCAN TaskPageTable 3:::0x123

; 3. if necessary, mask out one or multiple (peripheral) physical address
ranges so that they are not saved in the export script
MMU.MemAnalysis AddMask 0x6C000000--0x6CFFFFFF
MMU.MemAnalysis AddMask 0x6E000000--0x6FFFFFFF

; 4. if necessary, specify a maximum block size for page table entries.
MMU.MemAnalysis SET MaxBlocksize 0x40000000

; 5. control the number of Data.SAVE.BINary commands which are created in
the export script.
MMU.MemAnalysis SET mergeLevel 13.

; 6. export the list of physical address ranges from scanned page tables
to two PRACTICE scripts:
MMU.MemAnalysis SAVE C:\temp\export_memory.cmm C:\temp\import_image.cmm

; 7. if you are not satisfied with the resulting export scripts, modify
your settings from steps 3,4 or 5 and finally write new export and import
scripts with step 6.

; 8. finally run your export script on your target debugger.
DO C:\temp\export_memory.cmm
;  Optionally you can specify the destination path where the binary files
shall be saved to. This path must be enclosed in quotes.
DO C:\temp\export_memory.cmm "C:\ARM_Linux\sim_image"
```

See also

- MMU
- MMU.view

Format: **MMU.PageTable.dump** [*<address>* | *<range>*] (deprecated)
Use MMU.DUMP.PageTable instead.

MMU.PageTable.List [*<address>* | *<range>*] (deprecated)
Use MMU.List.PageTable instead.

MMU.PageTable.SCAN [*<address>* | *<range>*] (deprecated)
Use MMU.SCAN.PageTable instead.

See also

- MMU
- MMU.view

Format:	**MMU.SCAN** *<table>* **MMU.SCAN** *<table>* *<range>* | *<address>* **MMU.SCAN** *<table>* *<range>* | *<address>* *<root>* **MMU.***<table>***SCAN** (deprecated)
<table>:	**PageTable** [*<range>* | *<address>* *<address>* …] **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** [**Clear**] *<cpu_specific_tables>*
<cpu_specific_tables>:	**ITLB** | **DTLB** | **TLB** (e.g. for MPC860, MIPS, Hexagon) **PTE** | **BAT** (e.g. for MPC8260, MPC750) **TLB0** | **TLB1** (e.g. for MPC54xx, MPC85xx) **OEMAddressTable**

Scans the entries of the specified MMU translation table on the target or scans the translation look-aside buffer entries (such as ITLB) into the TRACE32-internal static translation list. The result is a snapshot of the scanned table and does not reflect the fact that the table may be dynamically modified by the target's OS.

During the scan process of the specified MMU translation table, its translation entries are parsed and added as static translation entries to the TRACE32-internal static translation list. The TRACE32-internal static translation list can be displayed with **TRANSlation.List**.

The **MMU.SCAN** command obviates the need to manually create the logical to physical address translations used by the currently running target OS with the **TRANSlation.Create** command.

As counterpart to **MMU.SCAN**, the **MMU.DUMP** command can be used to view the entries of a target MMU translation table in a window.

<table>	For descriptions of **PageTable**, **KernelPageTable**, etc, see **MMU.List**.
ALL [**Clear**]	Includes only page tables and not CPU-specific tables like TLB and others. **Clear:** This option allows to clear the static translations list before reading it from all page translation tables.
<cpu_specific_tables>	Many command arguments are target processor specific. For details, refer to the **Processor Architecture Manuals** [▲] listed in the **See also** block below.

NOTE: MMU translation tables (page tables) are dynamic structures and are frequently modified by the OS.
Instead of **MMU.SCAN**, use **TRANSlation.TableWalk ON** to enable the debugger table walk.
This method dynamically parses the page tables on demand for every debugger address translation. The debugger table walk is faster than repetitive **MMU.SCAN** calls and ensures that the debugger address translations correspond to the current OS address translations.

Examples:

```
MMU.SCAN TaskPageTable "hello"        ; scan the MMU table for the
                                      ; process "hello"

MMU.SCAN TaskPageTable 0x12:0x0       ; the process is specified by the
                                      ; space ID
```

See also

- MMU
- MMU.view
- TRANSlation.Create
- TRANSlation.List
- TRANSlation.SCANall
- TRANSlation.TlbAutoScan

▲ 'Arm Specific Implementations' in 'Arm Debugger'
▲ 'Arm Specific Implementations' in 'Armv8 and Armv9 Debugger'
▲ 'CPU specific MMU commands' in 'MPC5xx/8xx Debugger and Trace'
▲ 'Release Information' in 'Legacy Release History'

MMU.Set — Set MMU registers or tables

Format 1: **MMU.Set** [<*register_name*> [<*value*>]]

Format 2: **MMU.Set** <*table*>
MMU.<*table*>**.Set** (deprecated)

<*table*>: **TLB** (e.g. MIPS)
TLB0 | **TLB1** (e.g. for MPC54xx, MPC85xx)

Sets a physical MMU register or table. Note that this command is not available on all probes.

See also

- MMU
- MMU.view
- ❏ MMU()

MMU.TDUMP

Dump task page table

Format: **MMU.TDUMP**
(is an alias for **MMU.DUMP TaskPageTable**)

See also

- MMU
- MMU.view

MMU.TSCAN

Scan task page table

Format: **MMU.TSCAN**
(is an alias for **MMU.SCAN TaskPageTable**)

See also

- MMU
- MMU.view

MMU.view

View MMU registers

Format: **MMU.view**

Displays all MMU registers (Not available for all probes).

See also

- MMU
- MMU.List
- MMU.Set
- ❏ MMU.FORMAT()
- ❏ MMU.LINEAREX()
- MMU.DUMP
- MMU.MemAnalysis
- MMU.TDUMP
- ❏ MMU.INTERMEDIATE()
- ❏ MMU.LOGICAL()
- MMU.FORMAT
- MMU.PageTable
- MMU.TSCAN
- ❏ MMU.INTERMEDIATEEX()
- ❏ MMU.PHYSICAL()
- MMU.INFO
- MMU.SCAN
- ❏ MMU.DEFAULTPT()
- ❏ MMU.LINEAR()
- ❏ MMU.PHYSICALEX()

▲ 'Release Information' in 'Legacy Release History'

MMX — MMX registers (MultiMedia eXtension)

The **MMX** command group is used to display and modify the MMX (MultiMedia eXtension) registers.

See also

- MMX.Init
- MMX.Set
- MMX.view
- MMX()
- ▲ 'Command Groups for Special Registers' in 'Intel® x86/x64 Debugger'
- ▲ 'MMX Function (MultiMedia eXtension)' in 'General Function Reference'
- ▲ 'Release Information' in 'Legacy Release History'

MMX.Init — Initialize MMX registers

Format: **MMX.Init**
MMX.RESet (deprecated)

Sets all MMX registers to zero.

See also

- MMX
- MMX.view

MMX.Set — Modify MMX registers

Format: **MMX.Set** *<register>* *<value>* [*/<option>*]

Modifies the MMX registers.

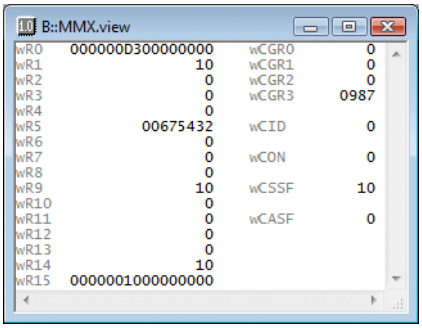
<option> — For a description of the options, see **Register.view**.

See also

- MMX
- MMX.view

Format: **MMX.view** [*/<option>*]

Opens an MMX register window.



<option> For a description of the options, see **Register.view**.

See also

■ MMX ■ MMX.Init ■ MMX.Set

▲ 'Release Information' in 'Legacy Release History'

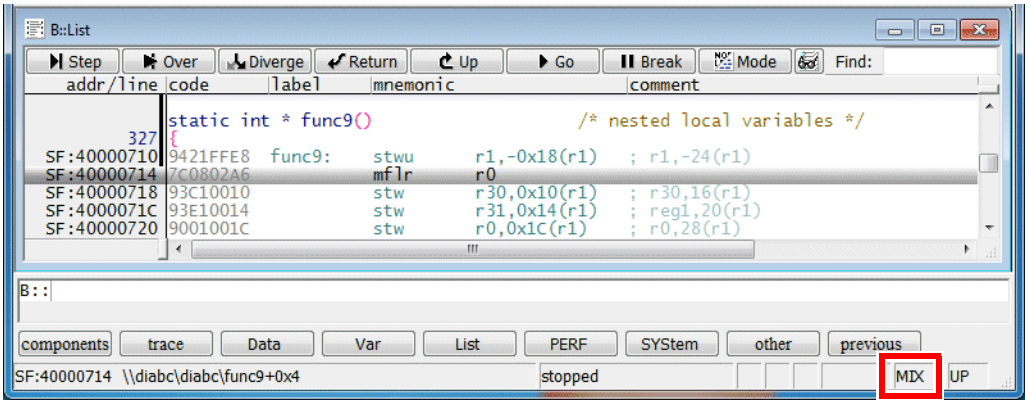
Mode — Set up the debug mode

Format:	**Mode**[.*<mode>*]
<mode>:	**switch** | **Asm** | **Hll** | **Mix**

The debug mode configures:

- How the source listing is displayed in the **List.*** windows, e.g. commands **List.Mix**, **List.Hll**.
- How the debugging commands, mainly **Go** and **Step**, behave.

The current debug mode is displayed in the TRACE32 state line, here: debug mode MIX.



Asm	In assembler mode, the source listing displays the disassembled memory contents without the source code information. Debugging is done on assembler level.
Hll	In HLL (High-Level Language) mode, the source listing displays only the source code. Debugging is done whenever possible at source code level.
Mix	In mixed mode, the source listing displays both, the disassembled memory contents and the source code. Debugging is done on assembler level.
switch	Toggles between **Mix** and **Hll** mode.

See also

- MACHINE.select
- List.Asm
- List.Hll
- List.Mix
- DEBUGMODE()

Appendix - <format> Options of MMU.FORMAT

<format> Options for Andes:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
STD	Standard format defined by the CPU

<format> Options for ARC:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
STD	Standard format defined by the CPU

<format> Options for ARM:

<format>	Description
EXTENSION.ALL	Table walk performed exclusively by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions. **MMU.FORMAT EXTENSION** can be used as an alias.
EXTENSION.MMUSPACES	Similar to **EXTENSION.ALL**, but requires **SYStem.Option.MMUSPACES ON**. With this format, *only* the table walk for *non-current* processes is performed by the user's TRACE32 extension. All other translation requests are handled by the built-in table walk function.
LINUX	Standard format used by Linux
LINUXSWAP	Linux <= 2.6.37 with configured swap space
LINUXSWAP3	Linux >= 2.6.38 with configured swap space
QNX.PLAIN	QNX format using the ARM FCSE translation. Use this format only if the kernel address range starts at a lower addresses than 0xFC000000. Other than format QNX.fcse, page table entries in the range 0x02000000 <= VA < 0xFC000000 are not hidden, but MMU.List.PageTable shows valid translations between 0x02000000 and the begin of the kernel address space which are actually not used by the OS. */
QNX.fcse	Standard QNX format using the ARM FCSE translation, assuming a kernel address range of 0xFC000000--0xFFFFFFFF. Page table entries for 0x02000000 <= VA < 0xFC000000 are hidden because these are neither process nor kernel specific addresses. */
STD	Standard format defined by the CPU
SYMBIAN	Format used by Symbian
TINY	MMU format using a tiny page size of only 1024 bytes
WINCE5	Format used by Windows CE5
WINCE6	Format used by Windows CE6 / EC7 / EC2013

<format> Options for BEYOND:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
STD	Standard format defined by the CPU

<format> Options for Hexagon QDSP6:

<format>	Description
BLAST	QURT Page Table format
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.
L4	L4 Page Table format
QURTV2	QuRT Page Table version 2
TLB	Reads TLB entries to translate logical addresses.
VTLB	QuRT VTLB. This is a virtual TLB in memory. If **SYStem.Option.MMUSPACES** is set to **ON**, the TRACE32 space ID of an address represents the ASID. During debugger address translation, TRACE32 matches the address and the ASID of VTLB entries. VTLB entries with ASID 0 or the global bit set will match any space ID. With **SYStem.Option.MMUSPACES** set to **OFF** (default), ASIDs will be ignored completely during the translation.

<format> Options for MicroBlaze:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
STD	Standard format defined by the CPU

<format>	Description
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.
LINUX32	Linux 32-bit, page size 4kB
LINUX32P16	Linux 32-bit, page size 16kB
LINUX32P16R2	Linux 32-bit, page size 16kB, used on MIPS32 R2 or R6 (internally identical to format LINUX32P16R41)
LINUX32P16R2	Deprecated: internally identical to format LINUX32P16R41
LINUX32R4K	Linux 32-bit, page size 4kB, like LINUX32 but different page flags
LINUX32RIXI	Linux 32-bit with RI/XI bits
LINUX64	Linux 64-bit with 64-bit PTEs, page size 4kB. Separate page table for high address range can be specified with optional extra parameter *<base_address_highrange>* (For details, see **MMU.FORMAT** in debugger_mips.pdf).
LINUX64HTLB	Linux 64-bit with 64-bit PTEs, page size 4kB for huge TLB. Uses separate sub table for addresses > 0xFFFFFFFFC0000000.
LINUX64HTLBP16	Linux 64-bit like LINUX64HTLB but pag esize 16kB.
LINUX64P16	Linux 64-bit with 64-bit PTEs, page size 16kB. Depth 3 levels.
LINUX64P64	Linux 64-bit with 64-bit PTEs, page size 64kB. Depth 3 levels.
LINUX64P64LT	Linux 64-bit with 64-bit PTEs, page size 64kB. Depth 2 levels with large level 1 table (used for BROADCOM(R) XLP SDK 3.7.10 and alike)
LINUX64RIXI	Linux 64-bit with 64-bit PTEs with RI/XI bits, page size 4kB. Separate page table for high address range can be specified with optional extra parameter *<base_address_highrange>* (For details, see **MMU.FORMAT** in debugger_mips.pdf).
LINUXBIG	Linux 32-bit with 64-bit PTEs on MIPS32
LINUXBIG64	Linux 32-bit with 64-bit PTEs on MIPS64
STD	Standard format defined by the CPU
WINCE6	Format used by Windows CE6

<format> Options for Motorola 68000:

<format>	Description
LINUX	Standard format used by Linux
STD	Standard format defined by the CPU

<format> Options for NIOS:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
STD	Standard format defined by the CPU

<format>	Description
DEOS	DEOS OS (32 bit) specific MMU format
DEOS64	DEOS OS (64 bit) specific MMU format
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
LINUX64_E6	Use LINUX64_E6 for e6500 core devices
LINUXE5	Linux with 64-bit PTEs, e500 core
LINUXEXT	Linux with 64-bit PTEs, no e500 core
LYNXOS	LynxOS format, virtual table pointers
LYNXOSPHYS	LynxOS format, physical table pointers
OSE	OSE format for load modules
PIKEOS.E500	PIKEOS specific format for PowerPC e500 core (formerly named PIKEOSE5).Works for PikeOS 4.1 and older. For e500 cores with PikeOS 4.2 and newer use E500MC format.*/
PIKEOS.E500MC	PIKEOS specific format for PowerPC e500mc core (PPC64 only).Can also be used with PikeOS 4.2 and newer on PPC32 e500 cores.*/
PIKEOS.E500MC4G	PIKEOS specific format for PowerPC e500mc core addressing 4GB of memory.Has no common address range.*/
PIKEOS.E5500	PIKEOS specific format for PowerPC e5500 core
PIKEOS.OEA	PIKEOS specific format for PowerPC core (formerly named PIKEOS) */
QNX	QNX standard format
QNXBIG	QNX format with 64-bit table entries
STD	Standard format defined by the CPU
VX653	MMU format for VXWORKS 653
VXWORKS.E500	VxWorks specific format for PowerPC e500 core
VXWORKS.E500MC	VxWorks specific format for PowerPC e500mc core with 36 bit physical addresses (PPC64 only)
VXWORKS.E500_64	VxWorks specific format for PowerPC e500 core (PPC64 only)
VXWORKS.E6500	VxWorks specific format for PowerPC e6500 core

<format> Options for RISC-V:

<format>	Description
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.
STD	Automatic detection of the page table format from the SATP register.
SV32	32-bit page table format (for SV32 targets only)
SV39	39-bit page table format (for SV64 targets only)
SV48	48-bit page table format (for SV64 targets only)

<format> Options for SH4:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
LINUXEXTP64	Linux with extended TLBs (3 page table levels, 64-bit PTEs)
QNX	QNX standard format
STD	Standard format defined by the CPU
WINCE6	Format used by Windows CE6

<format> Options for x86:

<format>	Description
EPT	Extended page table format (type autodetected)
EPT4L	Extended page table format (4-level page table)
EPT5L	Extended page table format (5-level page table)
EXTENSION	Table walk performed by a TRACE32 extension that a) was developed by the customer and b) defines table walk callback functions.
LINUX64	PAE64 derivative with different level 1 translation table entries for addresses >0xFFFF800000000000
P32	32-bit format with 2 page table levels
PAE	Format with 3 page table levels
PAE64	64-bit format with 4 page table levels
PAE64L5	64-bit format with 5 page table levels
STD	Automatic detection of the page table format used by the CPU

<format> Options for XTENSA:

<format>	Description
LINUX	Standard format used by Linux
LINUX26	Linux format with physical table pointers
STD	Standard format defined by the CPU