General Commands Reference Guide T

Release 09.2024

MANUAL

General Commands Reference Guide T

Version 05-Oct-2024

History

22-May-2024	A more detailed description of the <trace>.STATistic.TASKState command.
26-Sep-2023	New commands <trace>.PROfile.channel and <trace>.PROfile.CTU.
13-Jul-2023	Moved the option /ARTIAP from <trace>.FLOWPROCESS to the command <trace>.STATistic.PROCESS.
13-Jul-2023	New command <trace>.EXPORT.MDF. New option **/TimeZero** for the command <trace>.EXPORT.ARTI.
21-Jun-2023	Added the TRIG connector characteristics on PowerDebug X50 in chapter 'Overview TrBus'.
06-Jan-2023	New method <trace>.METHOD.CIProbe.
30-Aug-2022	Added the TRIG connector characteristics on PowerDebug E40 in chapter 'Overview TrBus'.
11-Aug-2022	New modes for TERM.Mode command to support UTF-8 encoded characters.
05-Aug-2022	New option **/Clear** for TRANSlation.SCANall command.
19-Jul-2022	New option **/BEAT** for <trace>.List command.
12-Jul-2022	New command TRACEPORT.LanePolarity.
16-Feb-2022	New command TASK.RELOAD.
05-Jan-2022	New command <trace>.EXPORT.ARTIAP.
05-Jan-2022	New option **/ARTIAP** for the commands <trace>.Chart.TASK, <trace>.Chart.TASKState, <trace>.FLOWPROCESS, <trace>.STATistic.TASK and <trace>.STATistic.TASKState.
05-Jan-2022	New **ARTIAP** items for <trace>.List.
Sep-2021	Description of the command <trace>.STATistic.TASKStateDURation.
Sep-2021	Description of the command <trace>.STATistic.RUNNABLEDURation.

TargetSystem

TRACE32 PowerView instances

Using the command group **TargetSystem**, you can start new TRACE32 PowerView instances from within a running instance and keep an overview of these instances.

The instances started with **TargetSystem.NewInstance** are automatically connected to the same PowerDebug hardware module or to the same MCI Server as the instance that initiated the start process. (In case of the MCI Server, the setting in the config file is: `PBI=MCISERVER`).

NOTE:	The **TargetSystem.NewInstance** command is not available for: • The TRACE32 Instruction Set Simulator (`PBI=SIM` in the config file) • The debuggers connected to the target via the GDI interface (`PBI=GDI`) • The debuggers connected to the target via the MCD interface (`PBI=MCD`)

The **TargetSystem.state** window provides an overview of the status of the cores assigned to the various TRACE32 instances. The window also helps you keep an overview of the synchronization mechanism between the TRACE32 instances, which is set up with the **SYnch** command group.

In addition, the **TargetSystem.state** window displays the InterCom names and UDP port numbers used by the instances for communication with each other via the **InterCom** system.

See also

■ TargetSystem.NewInstance ■ TargetSystem.state ■ SYnch ■ InterCom

[Examples]

Format:	**TargetSystem.NewInstance** *<intercom_name>* [**/***<option>*]
<option>:	**ARCHitecture** *<arch>* **API.PORT** *<port_number>* **ChipIndex** *<index>* | **ChipIndexMin** *<index_min>* **GDB.PORT** *<port>* | **GDB.PROTocol** [**TCP** | **UDP**] **InterCom.Port** *<port>* **LICense.PoolPort** [**None** | **Merge** | *<port>*] **ONCE** **SCReen.Size** [**Normal** | **ICONic** | **FULL** | **INVisible**] **TIMEOUT** [**None** | **Infinite** | *<time>*] **USEmask** *<value>*
<arch>:	**8051** | **COLDFIRE** | **ANDES** | **AP3** | **ARC** | **ARM** | **ARM64** | …
<index>:	**1.** … **254.**
<index_min>:	**1.** … **254.**

Allows a TRACE32 PowerView instance to start new TRACE32 PowerView instances (max. 15 new instances) for debugging AMP systems. In AMP (asynchronous multiprocessing) systems, each TRACE32 PowerView instance is responsible for an SMP subsystem or single core. For more information, see **CORE.ASSIGN**.

All instances started with **TargetSystem.NewInstance** are automatically connected to the same PowerDebug hardware module or the same MCI Server (`PBI=MCISERVER` in the config.t32 file) as the instance that initiated the start process.

The instance that starts another instance clones the current config file (by default config.t32) and extends the cloned file for the new instance.

NOTE:	The **TargetSystem.NewInstance** command is not available for: • The TRACE32 Instruction Set Simulator (`PBI=SIM` in the config file) • The debuggers connected to the target via the GDI interface (`PBI=GDI`) • The debuggers connected to the target via the MCD interface (`PBI=MCD`)

<table>
<tr><td><intercom_name></td><td colspan="2">Assigns a user-defined InterCom name to the new TRACE32 instance.</td></tr>
<tr><td>ARCHitecture
<arch></td><td colspan="2">Selects the architecture of the new TRACE32 instance. If the ARCHitecture option is omitted, then a TRACE32 instance of the same architecture will be started.

The softkeys below the TRACE32 command line include only architectures and families that are used for AMP debugging.</td></tr>
<tr><td>API.PORT
<port_number></td><td colspan="2">Parameter Type: Decimal value.
Passes a UDP remote API <port_number> to the new TRACE32 instance.</td></tr>
<tr><td>ChipIndex<index></td><td colspan="2">Sets the value of <code>CORE=</code> in the config file of the new instance.
See also:
• “Section PBI” in TRACE32 Installation Guide, page 42 (installation.pdf)
• SYStem.USECORE()</td></tr>
<tr><td>ChipIndexMin
<index_min></td><td colspan="2">Automatically chosen index will not be below the minimum specified with <index_min>.</td></tr>
<tr><td>GDB.PORT <port></td><td colspan="2">Enables the GDB server listening at the passed port for the new TRACE32 instance to start.</td></tr>
<tr><td>GDB.PROTocol</td><td colspan="2">Setups the used IP protocol for the GDB service. Default: TCP.</td></tr>
<tr><td>InterCom.Port
<value></td><td colspan="2">Specifies the new InterCom port that shall be used for the new instance. This option presumes that the current instance already have an assigned InterCom port to avoid later conflicts.</td></tr>
<tr><td rowspan="4">LICense.PoolPort</td><td colspan="2">Manages license pool ports for TargetSystem.NewInstance command for MCISERVER scenarios.
This option defines how new instance work with pool ports.
The default is Merge when the POOLPORT has been specified in the current instance otherwise None.</td></tr>
<tr><td>None</td><td>Does not use the POOLPORT keyword in the LICENSE section.</td></tr>
<tr><td>Merge</td><td>Introduces the POOLPORT and create new pools depending the started architectures.</td></tr>
<tr><td><port></td><td>Configures the POOLPORT to certain value.</td></tr>
<tr><td>ONCE</td><td colspan="2">Avoids starting an instance with the same name multiple times.</td></tr>
</table>

<table>
<tr><td rowspan="5">SCReen.Size</td><td colspan="2">Configures window modes.</td></tr>
<tr><td>Normal</td><td>The new PowerView instance is started as normal window.</td></tr>
<tr><td>ICONic</td><td>The new PowerView instance is minimized.</td></tr>
<tr><td>FULL</td><td>The new PowerView instance is started in full screen mode.</td></tr>
<tr><td>INVisible</td><td>The new PowerView instance is invisible.</td></tr>
<tr><td rowspan="4">TIMEOUT</td><td colspan="2">Used to configure the timeout to wait until the new instance has finished the initialization phase.</td></tr>
<tr><td>None</td><td>Immediately returns from the command and does not wait until the new TRACE32 instance is spawned.</td></tr>
<tr><td>Infinite</td><td>Waits until the new TRACE32 instance is spawned for an infinite time or the STOP button is clicked.</td></tr>
<tr><td><time></td><td>Waits a certain time until the new TRACE32 instance is spawned. Default: 10 seconds.</td></tr>
<tr><td>USEmask <value></td><td colspan="2">Used to overwrite USE= property of PBI section.
For rare use case the use mask to address POD bus devices can be modified for the new GUI instance. The use mask can be passed as string or as value with least significant bit corresponding to first POD bus device in the chain.

Examples:
<code>/USEMASK 001101</code>; generates USE=001101, first character corresponds to the first device in the POD bus chain.
<code>/USEMASK 0y001101;</code> generates USE=1011, least significant bit corresponds to the first device in the POD bus chain.
<code>/USEMASK 0x0D;</code> generates USE=1011, least significant bit corresponds to the first device in the POD bus chain.</td></tr>
</table>

Examples

Example 1: This script shows how to start a second TRACE32 instance named `mySecondInstance` from within the current TRACE32 instance.

```
TargetSystem.NewInstance mySecondInstance /ARCHitecture ARM64
InterCom.execute mySecondInstance PRINT "started by the first instance"
```

Example 2: Let’s assume you have started a number of instances and now want to quit a particular instance. This script shows how to quit a TRACE32 instance named `mySecondInstance` in a set of TRACE32 instances.

```
InterCom.execute mySecondInstance QUIT
```

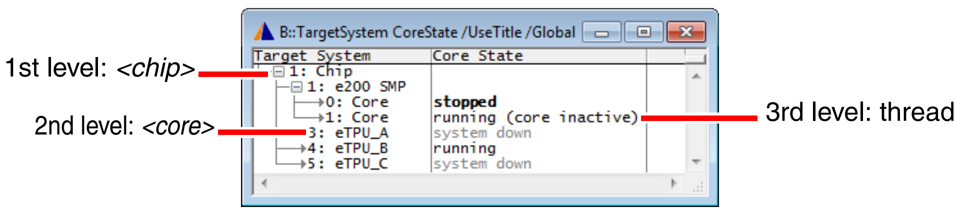
See also

■ TargetSystem ■ InterCom.ENable

▲ ’Release Information’ in ’Legacy Release History’

Format:	**TargetSystem.state** [*<column>* …] [**/***<option>* …]
<column>:	**DEFault** | **ALL** **TargetSystem** | **CoreType** | **CoreState** | **Title** | **InterComPort** | **InterComName** | **INSTance** | **UseCore** **SYnch.All** | **SYnch.Go** | **SYnch.Step** | **SYnch.Break** | **SYnch.SystemMode** **LicensePoolPort**
<option>:	**Global** | **UseTitle** | **UseICName**

Opens the **TargetSystem.state** window, providing an overview of the multicore system configuration and state across multiple TRACE32 instances sharing one PowerDebug hardware module or MCI Server. The indices on the first and second level are configured using **SYStem.CONFIG.CORE** *<chip> <core>*. The indices on the third level indicate the thread index of the SMP system that can be defined by **CORE.ASSIGN** or **CORE.NUMber**.



The **TargetSystem** window is not available for front-end debuggers.

To illustrate the **TargetSystem.state** command, the following use cases are provided:

- **Use case 1:** Diagnostic tool for the target system structure
- **Use case 2:** TRACE32 instance selector
- **Use case 3:** Manage the **SYnch** settings for all TRACE32 instances

DEFault	Adds **TargetSystem**, **CoreType** and **CoreState** column. If no column is passed **DEFault** is used automatically.
ALL	Displays all available columns in the **TargetSystem.state** window.
TargetSystem	Adds the **TargetSystem** column to show a hierarchical view on the system. If the column is left out, it will be added automatically. The parameter is used to tell the dialog that the **DEFault** option is not active and only the **TargetSystem** column shall be shown.
CoreType	Adds a column to show the target architecture of a core and core family name if available.
CoreState	Shows the state of the core. The state can be system down (gray color), power down (red color), reset (red color), stopped (bold) or running. The running state can be extended by an attribute that indicates a run mode e.g. "no core clock".
Title	Adds a column with the corresponding window title. The title can be set by the configuration file before start-up or by the **TITLE** command.
InterComPort	Adds a column with the InterCom UDP port numbers of TRACE32 instances. The InterCom port numbers are used by the **InterCom** commands and the **SYnch** commands. You can assign a new port number by double-clicking a port number in the **ic port** column. For an illustrated example, see **InterCom.PORT**.
InterComName	Adds a column with the InterCom names of TRACE32 instances. Names are created with the commands **InterCom.NAME** or **InterCom.ENable**. The names can then be used as arguments in **InterCom** and **SYnch** commands. You can rename an instance by double-clicking a name in the **ic name** column. For an illustrated example, see **InterCom.NAME**.
INSTance	Adds a column, showing the value of `INSTANCE=` from the config file. If `INSTANCE=` is missing in the config file, then 1 is displayed by default. That is, in this case the display value is equivalent to the explicit setting `INSTANCE=1` in the config file.
UseCore	Adds a column, showing the value of `CORE=` from the config file. If `CORE=` is missing in the config file, then 1 is displayed by default. That is, in this case the display value is equivalent to the explicit setting `CORE=1` in the config file. See also **SYStem.USECORE()**.

SYnch.All	Adds the columns **SYnch.Go**, **SYnch.Step, SYnch.Break** and **SYnch.SystemMode**.
SYnch.Go	Adds the column to indicate and edit the **SYnch.MasterGo** and **SYnch.SlaveGo** setting. The header of the column is named **SG**.
SYnch.Step	Adds the column to indicate and edit the **SYnch.MasterStep** and **SYnch.SlaveStep** setting. The header of the column is named **SS**.
SYnch.Break	Adds the column to indicate and edit the **SYnch.MasterBreak** and **SYnch.SlaveBreak** setting. The header of the column is named **SB**.
SYnch.System-Mode	Adds the column to indicate and edit the **SYnch.MasterSystemMode** and **SYnch.SlaveSystemMode** setting. The header of the column is named **SM**.
LicensePoolPort	Displays license pool port column in **TargetSystem** window.

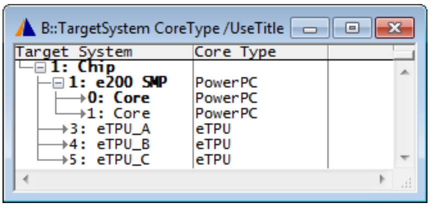
<options> - Options for the TargetSystem.state Window

Global	Don't highlight specific information for the TRACE32 instance from where the dialog was opened. The dialog can be moved outside of the main window and used to act as an independent window to bring a certain instance to foreground by a double click to of an entry of the **TargetSystem** tree column.
UseTitle	Use the TRACE32 window title as name for an SMP Subsystem or Core. The title can be set by the configuration file before start-up or by the PRACTICE command **TITLE**.
UseICName	Use the TRACE32 InterCom name as window title for an SMP subsystem or core. The InterCom name can be set with the **InterCom.NAME** command.

Use case 1: Diagnostic tool for the target system structure

The command opens the window showing the overall system. Nodes that belong to this TRACE32 instance are displayed in bold. A double-click to a thread selects this thread to be active.

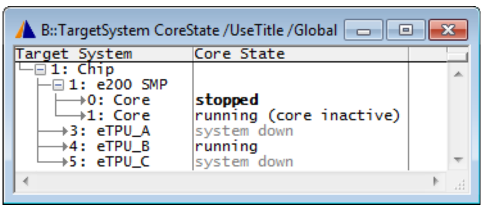
```
TargetSystem.state CoreType /UseTitle
```



Use case 2: TRACE32 instance selector

The command opens the window showing the overall system and the state of the particular cores. The window can be moved outside of the TRACE32 instance where the command was executed. A double-click at an SMP system node or core will bring the assigned instance to foreground.

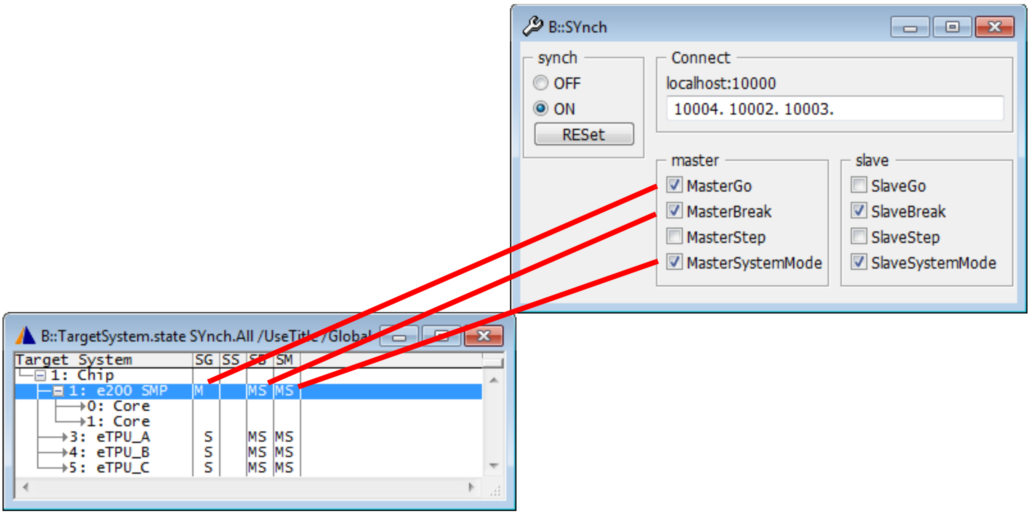
```
TargetSystem.state CoreState /UseTitle /Global
```



Use case 3: Manage the SYnch settings for all TRACE32 instances

The command opens the window showing the overall system and the **SYnch** settings.

```
TargetSystem.state SYnch.All /UseTitle /Global
```



A single click at an entry in one of the columns will change the setting in the **SYnch** dialog and set the connection ports.

default		Neither master nor slave option is set.
1st click	**M**	master option set.
2nd click	**S**	slave option is set.
3rd click	**MS**	master and slave option is set.

See also

■ TargetSystem

TASK

OS Awareness for TRACE32

[Task Magic Numbers, IDs, Names] [Machine Magic Numbers, IDs, Names] [Glossary]

See also

- ■ TASK.ACCESS
- ■ TASK.ATTACH
- ■ TASK.Break
- ■ TASK.CACHEFLUSH
- ■ TASK.CONFIG
- ■ TASK.COPYDOWN
- ■ TASK.COPYUP
- ■ TASK.Create
- ■ TASK.CreateExtraID
- ■ TASK.CreateID
- ■ TASK.DELete
- ■ TASK.DeleteID
- ■ TASK.DETACH
- ■ TASK.Go
- ■ TASK.INSTALL
- ■ TASK.KILL
- ■ TASK.List
- ■ TASK.ListID
- ■ TASK.NAME
- ■ TASK.ORTI
- ■ TASK.RELOAD
- ■ TASK.RESet
- ■ TASK.RUN
- ■ TASK.select
- ■ TASK.SETDIR
- ■ TASK.STacK
- ■ EXTension
- ❏ TASK.ACCESS()
- ❏ TASK.BACK()
- ❏ TASK.CONFIG()
- ❏ TASK.CONFIGFILE()
- ❏ TASK.COUNT()
- ❏ TASK.FIRST()
- ❏ TASK.FORE()
- ❏ TASK.ID()
- ❏ TASK.MACHINEID()
- ❏ TASK.MAGIC()
- ❏ TASK.MAGICADDRESS()
- ❏ TASK.MAGICRANGE()
- ❏ TASK.MAGICSIZE()
- ❏ TASK.NAME()
- ❏ TASK.NEXT()
- ❏ TASK.ORTIFILE()
- ❏ TASK.SPACEID()

▲ 'TASK Functions' in 'General Function Reference'

Overview TASK

This chapter describes the OS Awareness features, generic to all processors and kernels. Kernel specific features are described in additional manuals, see **OS Awareness Manuals**.

The OS Awareness may support the following main features:

- Display of kernel resources (e.g. tasks, queues, semaphores, messages).
- Task stack coverage.
- Task related breakpoints.
- Task context display.
- Operating system's MMU support.
- Dynamic task performance measurement
- Task runtime statistics and flowchart display out of the trace buffer. Display of task switches in the trace listing.
- Task state statistics and time chart out of the trace buffer, i.e. show how long each task is in a certain state (running, ready, etc.).
- Task-related function runtime statistics, flowchart display and function nesting display out of the trace buffer.
- Fast access to the features through dedicated menus.

Not all features are implemented for all processors and kernels. Please see the kernel specific manual for a detailed description of the supported features.

OS Awareness Configurations

The OS Awareness is configured by the **TASK.CONFIG** command. The command loads a configuration file that tells the debugger all kernel-related information. It can be adopted to any (RT)OS kernel. Lauterbach provides ready-to-start configuration files for a wide range of operating systems. If you want to adapt it to your own proprietary kernel, ask Lauterbach for assistance.

What to know about the Task Parameters

In TRACE32, operating system tasks (short: tasks) can be identified based on one of these values:

- Task magic number
- Task ID
- Task name

For OS-aware debugging and tracing, these three values are displayed in the **TASK.List.tasks** window and can be returned with the functions **TASK.MAGIC()**, **TASK.ID()**, and **TASK.NAME()**. In addition, the three values can be passed as parameters to task-related TRACE32 commands and options.

NOTE:	In case of the **TASK.CONFIG** command, you will encounter the parameter *<magic_address>*. • *<task_magic>* and *<magic_address>* are **not** the same. • For information about *<magic_address>*, see **TASK.CONFIG** command.

Task Magic Number

The task magic number is an arbitrary hex value, used by TRACE32 to uniquely identify a task of an operating system. The meaning of the value depends on the OS Awareness; often it refers to the task control block of the target OS or to the task ID.

<task_magic>	**Parameter Type**: Hex value. **Example**: `TASK.select 0xEFF7B040`

Task ID

This value refers to the numeric task ID as given by the operating system. If the OS does not provide a task ID, this option may not be available.

<task_id>	**Parameter Type**: Decimal value. **Example**: `TASK.select 1546.`

Task Name

This string refers to the task name as given by the operating system. If the OS does not provide a task name, this option may not be available.

If the task runs in a system involving virtualization, then the task name can be preceded with the machine name.

<task_name>	**Parameter Type**: String. **Example 1**: `TASK.select "adbd:1546"` **Example 2**: `TASK.select "FreeRTOS:::SieveDemo"` `FreeRTOS` is the name of the machine. The three colons `:::` serve as the separator between the machine name and the task name `SieveDemo`.

What to know about the Machine Parameters

In hypervisor-based environments, TRACE32 identifies machines based on one of these values:

- Machine magic number
- Machine ID
- Machine name

For hypervisor debugging and tracing, these three values are displayed in the **TASK.List.MACHINES** window. In addition, the three values can be passed as parameters to machine-related TRACE32 commands and options.

Machine Magic Number

A machine magic number is an arbitrary hex value, used by TRACE32 to uniquely identify a machine (host machine or guest machine). The meaning of the value depends on the Hypervisor Awareness; often it refers to the guest control block of the hypervisor or to the machine ID.

<machine_magic>	**Parameter Type**: Hex value. **Range**: machine magic number > **`0xFF`** Machine magic numbers are displayed, for example, in the **magic** column of the **TASK.List.MACHINES** window as hex values.

Machine ID

A *machine ID* is a numeric identifier which extends a logical address and intermediate address in TRACE32 or can be used together with the option **MACHINE** in some TRACE32 commands. The purpose of a machine ID is to identify guest machines within a system that is using a hypervisor to run multiple virtual machines.

<machine_id>	**Parameter Type**: Decimal or hex value. **Range**: **`0x0`** <= machine ID < **`0x1F`** Machine IDs are displayed, for example, in the **mid** column of the **TASK.List.MACHINES** window as decimal values (**`1.`**, **`2.`**, etc.)

In TRACE32, the machine ID clearly specifies which virtual machine (a guest machine or the host machine) an address belongs to:

- The machine ID 0 (zero) is always associated with the host machine running the hypervisor.
- All the other machine IDs >= 1 are associated with the guest machines.

Format of addresses with machine IDs:

In the TRACE32 address format, the machine ID is always in the leading position, directly after the access class specifier. The machine ID is followed a triple colon (**`:::`**) to separate the machine ID from the remaining parts of an address. The format of a TRACE32 address containing a machine ID looks like this:

- Without space ID:

  `<access_class>:<machine_id>:::<address_offset>`

- With space ID:

  `<access_class>:<machine_id>:::<space_id>::<address_offset>`

Examples:

- Without space ID:
  - `G:0x1:::0x80000000`
  - `0x2:::0xA0000000`
- With space ID:

- `G:0x3:::0x020A::0x80000000`
- `G:0x0:::0x0::0x4000C000`
- `0x2:::0x170::0x1F000000`

Notes:

- Machine IDs can only be used if a TRACE32 Hypervisor Awareness is loaded with the command **EXTension.LOAD**.
- Use command **SYStem.Option.MACHINESPACES ON** to enable machine IDs in TRACE32.

Machine Name

A machine name is a meaningful string that allows users to identify a host or guest machine in a hypervisor-based environment. The machine name is given by the Hypervisor Awareness. If the Hypervisor Awareness does not provide a machine name, you can assign a name to a machine by using the **NAME** option of the **EXTension.LOAD** command. Without the **NAME** option, the base name of the extension definition file will be used.

In a hypervisor-based environment, the machine name precedes the task name.

<machine_name>	**Parameter Type**: String. **Example**: `TASK.select "FreeRTOS:::SieveDemo"` `FreeRTOS` is the name of the machine. The three colons `:::` serve as the separator between the machine name and the task name `SieveDemo`.

Glossary

For important OS Awareness and Hypervisor Awareness terms, such as task, thread, process, machine, kernel, MMU space, and virtual machine, refer to the **"TRACE32 Concepts"** (trace32_concepts.pdf).

TASK.ACCESS

Control memory access

Format: **TASK.ACCESS** [*<class>*]

Defines the memory access class used by **TASK** related windows.

TASK related windows may access the target memory (e.g. when reading task control blocks). If the access class is set to E:, the debugger uses emulation memory access to read the memory (e.g. emulation memory, shadow memory or pseudo-dual-port access). If set to C:, the debugger uses CPU access. If the appropriate access is not possible, the window is temporarily frozen.

TASK.ACCESS without parameter enables the default mode, which uses E:, if the application is running, and C: if the application is stopped.

Please see refer to your **Processor Architecture Manuals** for a description of E: and C:.

See also

■ TASK

TASK.ATTACH

Attach to a running process

Format: **TASK.ATTACH** *<id>*

Start the execution of a single task or thread.

Only applicable if GDB (Linux) is used as debug agent or for the Native Process Debugger.

See also

■ TASK

▲ 'GDB Front-End TASK Commands' in 'TRACE32 as GDB Front-End'
▲ 'Native Process Debugger Specific TASK Commands' in 'Native Process Debugger'

TASK.Break

Stop the execution of a single task or thread

Format: **TASK.Break** *<id>*

Stop the execution of a single task or thread.

Only applicable if GDB (Linux) is used as debug agent or for the Native Process Debugger.

See also

- TASK
- 'GDB Front-End TASK Commands' in 'TRACE32 as GDB Front-End'
- 'Native Process Debugger Specific TASK Commands' in 'Native Process Debugger'

TASK.CACHEFLUSH — Reread task list

Format:	**TASK.CACHEFLUSH**

Usually not needed. Use only if advised to do so.

The debugger reads out the task list of the target at each single step or Go/Break sequence, and stores the list internally (see **TASK.List.tasks**). If the task list or task characteristics change *while* the target is halted, a manual update of the task list *may* be necessary. This command forces an immediate re-evaluation of the task list.

See also

- TASK

TASK.CONFIG — Configure OS Awareness

Format:	**TASK.CONFIG** *<os_awareness_file> <magic_address> <args>* [**/***<option>*]
<option>:	**ACCESS** *<class>*

Configures the OS Awareness using a given configuration file. Please refer to the OS-specific manual. See **OS Awareness Manuals**.

Arguments:

<os_awareness_file>	File name of the configuration file.
<magic_address>	Address of the memory location holding the task magic number of the currently running task. See **"What to know about the Task Parameters"**, page 27.
<args>	All other arguments are interpreted by the configuration file. Details of predefined files are described in the kernel-specific part of an **OS Awareness Manual**.

Options:

ACCESS	Defines the memory access class used by **TASK**-related windows. See **TASK.ACCESS**.

See also

- TASK
- EXTension.LOAD
- MMU

▲ 'Release Information' in 'Legacy Release History'

TASK.COPYDOWN

Copy file from host into target

Format: **TASK.COPYDOWN** *<source_file_host> <destination_file_target>*

Copies a file from the host into the target. Only supported for Linux and QNX run mode debugging.

See also

- TASK

▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'

Format:	**TASK.COPYUP** *<source_file_target> <destination_file_host>*

Copies a file from the target into the host. Only supported for Linux and QNX run mode debugging.

See also

■ TASK

▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'

TASK.Create — Create task

The **TASK.Create** command group allows to create new tasks.

See also

■ TASK

TASK.Create.MACHINE — Define a manual machine

Format:	**TASK.Create.MACHINE** [*<mach_magic>*] [*<id>*] [*<name>*] [*<vttb>*] [*<trace_id>*] [/*<option>*]
<option>:	**MMUspaces ON** | **OFF** | **EXTension** **CORE** *<core1>* [*<core2>*...]

Defines a persistent machine. Machines are usually created and removed from the machine list by a Hypervisor Awareness. This commands creates machines that are independent of the Hypervisor Awareness.

Only available if **SYStem.Option.MACHINESPACES** is ON.

Parameter:	Format	Description
<mach_magic>	hex	Specifies a value that uniquely identifies a machine.
<id>	dec	Specifies a machine ID as used by fully qualified virtual addresses.
<name>	string	Specifies a machine name.
<vttb>	hex	Specifies the translation table base address of this machine.
<trace_id>	hex	Specifies a value that identifies a machine in the trace.

All parameters are optional. If omitted (specify ','), the debugger will try to get the value from the Hypervisor Awareness (if available).

Option:	Description	
MMUspaces	**ON**	This machine has MMU spaces.
	OFF	This machine does not have MMU spaces.
	EXTension (default)	An OS Awareness for this machine (if available) reports, if this machine has MMU spaces.
CORE	Assigns a machine to specific logical cores.	

Examples:

```
;Declare a machine with machine ID 1 and name “guest1”:
TASK.Create.MACHINE , 1. “guest1”
```

```
;Set the trace ID of machine with magic “0x1234” to “0x2”:
TASK.Create.MACHINE 0x1234, , , , 0x2
```

```
;Machine with ID 2 is bound to logical cores 2 and 3:
TASK.Create.MACHINE , 2. /CORE 2. 3.
```

See also

- TASK.List.MACHINES

TASK.Create.RUNNABLE — Define an AUTOSAR runnable

Format: **TASK.Create.RUNNABLE** [<*function*>] [<*id*>] [<*name*>] [<*start*>] [<*stop*>] [<*traceidstart*>] [<*traceidstop*>] [/<*option*>]

Defines an AUTOSAR runnable. Usually used in conjuntion with an ORTI awareness (see **TASK.ORTI**).

Parameter:	Format	Description
<function>	string	Specifies a function symbol that represents this runnable.
<id>	dec	Specifies a runnable id.
<name>	string	Specifies a runnable name.
<start>	address	Specifies the start address of the runnable.
<stop>	address	Specifies the end address of the runnable.
<traceidstart>	hex	Specifies a value that identifies the start of a runnable in the trace.
<traceidstop>	hex	Specifies a value that identifies the end of a runnable in the trace.

All parameters are optional. If omitted (specify ','), the debugger will try to evaluate the other values by the given values.

Example:

```
;Declare a runnable:
TASK.Create.RUNNABLE Rte_Runnable_ComM_GetCurrentComMode_Start 3.\
                                                     "GetCurrentComMode"
```

See also

- TASK.List.RUNNABLES
- <trace>.EXPORT.ARTI
- <trace>.Chart.RUNNABLE
- <trace>.STATistic.RUNNABLE

▲ 'Overview of TRACE32 Command Structure' in 'Application Note Profiling on AUTOSAR CP with ARTI'

TASK.Create.SPACE — Define a manual MMU space

Format:	**TASK.Create.SPACE** [*<space_magic>*] [*<id>*] [*<name>*] [*<ttb>*] [/*<option>*]
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

Defines a persistent MMU space. MMU Spaces are usually created and removed from the space list by an OS Awareness. This commands creates spaces that are independent of the OS Awareness.

Only available if **SYStem.Option.MMUSPACES** is ON.

Parameter:	Format	Description
<space_magic>	hex	Specifies a value that uniquely identifies a space within a machine.
<id>	dec	Specifies a space ID as used by fully qualified virtual addresses.
<name>	string	Specifies a space name.
<ttb>	hex	Specifies the translation table base address of this space.

All parameters are optional. If omitted (specify ','), the debugger will try to get the value from the OS Awareness (if available).

Option:	Description
MACHINE	Creates the task to be part of the given machine. (only available if **SYStem.Option.MACHINESPACES** is ON)

Examples:

```
;Declare an MMU space with space ID 1 and name "proc1":
TASK.Create.SPACE , 1. "proc1"
```

```
;Set the TTB of the MMU space with magic "0x1234" on machine 1 to
;"0x1000":
TASK.Create.SPACE 0x1234, , , 0x1000 /MACHINE 1.
```

See also

- TASK.List.SPACES

Format:	**TASK.Create.task** [*<task_magic>*] [*<id>*] [*<name>*] [*<trace_id>*] [/*<option>*]
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>* **SPACE** *<space_magic>* | *<space_id>* | *<space_name>*

Defines a persistent task. Tasks are usually created and removed from the task list by an OS Awareness. This commands creates tasks that are independent of the OS Awareness.

Parameter:	**Format**	**Description**
<task_magic>	hex	Specifies a value that uniquely identifies a task within a machine.
<id>	dec	Specifies an arbitrary task ID.
<name>	string	Specifies a task name.
<trace_id>	hex	Specifies a value that identifies a task in the trace.

All parameters are optional. If omitted (specify ','), the debugger will try to get the value from the OS Awareness (if available).

Option:	**Description**
MACHINE	Creates the task to be part of the given machine. (only available if **SYStem.Option.MACHINESPACES** is ON)
SPACE	Create the task to be part of the given space. (only availabe if **SYStem.Option.MMUSPACES** is ON)

Examples:

```
;Declare a task with magic "0x200" and name "thread1" as part of MMU
;space "proc1":
TASK.Create.task 0x200 , "thread1" /SPACE "proc1"
```

```
;Set the trace ID of task with magic "0x200" of machine 1 to "0x4":
TASK.Create.task 0x200 , , 0x4 /MACHINE 1.
```

See also

■ TASK.List.tasks

TASK.CreateExtraID — Create a virtual task

Format:	**TASK.CreateExtraID** *<task_name> <task_id> <space_id> <trace_id>*

Creates a virtual task ID for trace analysis. Trace analysis will use the given task ID for task identification rather than the task magic number. Only for some dedicated applications.

See also

- TASK

TASK.CreateID — Create virtual task

Format:	**TASK.CreateID** *<task_name> <task_id> <space_id> <trace_id>*

Creates a virtual task name for trace analysis. Trace analysis will use the given task name for task identification, rather than the task magic. Only for some dedicated applications.

See also

- TASK

TASK.DELete — Delete file from target

Format:	**TASK.DELete** *<target_file>*

Deletes a file from the target file system. Only applicable if GDB (Linux) is used as debug agent.

See also

- TASK

TASK.DeleteID

Delete virtual task

Format:	**TASK.DeleteID** *<task_id>*

Delete a virtual task created with **TASK.CreateID** or **TASK.CreateExtraID**.

See also

■ TASK

TASK.DETACH

Detach from task

Format:	**TASK.DETACH** *<id>*

Requests the debug agent to detach from the process *<id>*.

Only applicable if GDB (Linux) is used as debug agent.

Example:

```
TASK.DETACH 41.
```

See also

■ TASK
▲ 'Native Process Debugger Specific TASK Commands' in 'Native Process Debugger'

TASK.Go

Start the execution of a single task or thread

Format:	**TASK.Go** *<id>*

Start the execution of a single task or thread.

Only applicable if GDB (Linux) is used as debug agent or for the Native Process Debugger.

See also

■ TASK
▲ 'GDB Front-End TASK Commands' in 'TRACE32 as GDB Front-End'
▲ 'Native Process Debugger Specific TASK Commands' in 'Native Process Debugger'

TASK.INSTALL

Deprecated

Format:	**TASK.INSTALL** (deprecated)

See also

■ TASK

TASK.KILL

End task

Format:	**TASK.KILL** *<id>*

Request the debug agent to end the process *<id>*.

Only applicable if GDB (Linux) or TRK (Symbian) is used as debug agent.

Example:

```
TASK.KILL 41.
```

See also

■ TASK

▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'
▲ 'Native Process Debugger Specific TASK Commands' in 'Native Process Debugger'

The windows of the **TASK.List** command group provide information about processes, space IDs, MMU spaces, machines, and tasks known to the debugger in an OS and hypervisor environment. The debugger needs a so-called “awareness” of the OS or hypervisor to be able to read out these items from the target.

See also

- TASK.List.MACHINES
- TASK.List.RUNNABLES
- TASK.List.SPACES
- TASK.List.tasks
- TASK.List.TREE
- TASK

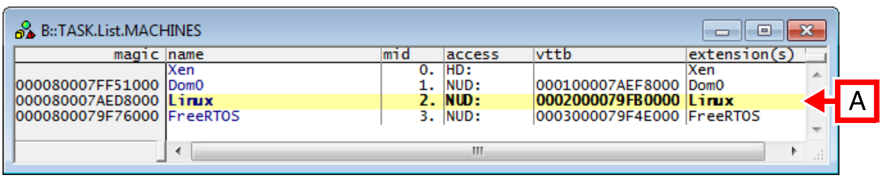
TASK.List.MACHINES List machines

Format:	**TASK.List.MACHINES**

Lists information about all machines known to the debugger. Machines refer to virtual machines in a hypervisor environment. The hypervisor itself is listed as machine with ID 0.

Machines are only available if **SYStem.Option.MACHINESPACES** is set to **ON**.

For several purposes, the debugger needs to know which machines are active in the system. The debugger uses the hypervisor specific awareness to read out all machine characteristics that it needs for its operation. **TASK.List.MACHINES** shows the machine characteristics that the debugger uses.



A The machine that is currently running on the selected core is marked.

Description of Columns in the TASK.List.MACHINE Window

magic	Machine magic number. Unique number for the machine. Usually the address of the control block structure.
name	Name of the object, if available.
mid	Machine ID if a hypervisor system is set up.
access	Access class that an awareness uses for this machine.
vttb	“Virtual translation table base” address of this machine. The VTTB address points to the MMU table of the guest physical (= intermediate) address to host physical address translation.
extension(s)	Extensions loaded for this machine (**EXTension.LOAD**).

See also

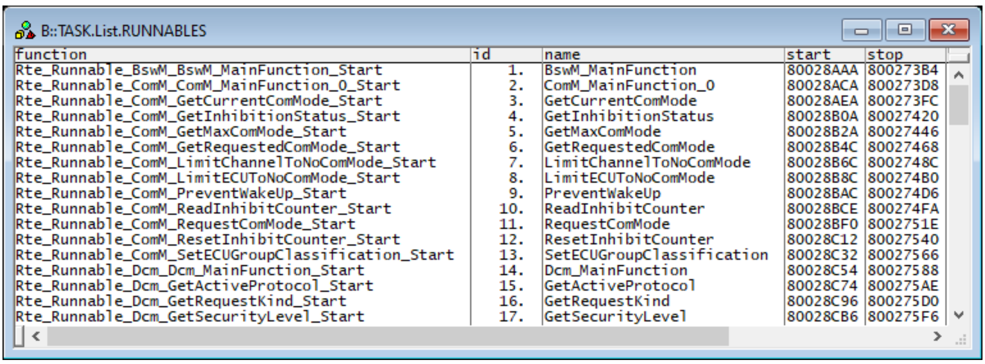
- ■ TASK.List
- ■ TASK.List.tasks
- ■ TASK.Create.MACHINE
- ▲ 'Release Information' in 'Legacy Release History'

TASK.List.RUNNABLES List AUTOSAR runnables

Format:	**TASK.List.RUNNABLES**

Lists information about AUTOSAR runnables.

Runnables are declared to the debugger by the command **TASK.Create.RUNNABLE**. The information is used to create performance calculations shown with **Trace.Chart.RUNNABLE** and **Trace.STATistic.RUNNABLE**. **Trace.EXPORT.ARTI** also relies on this information to export trace events based on runnables.


B::TASK.List.RUNNABLES

function	id	name	start	stop
Rte_Runnable_BswM_BswM_MainFunction_Start	1.	BswM_MainFunction	80028AAA	800273B4
Rte_Runnable_ComM_ComM_MainFunction_0_Start	2.	ComM_MainFunction_0	80028ACA	800273D8
Rte_Runnable_ComM_GetCurrentComMode_Start	3.	GetCurrentComMode	80028AEA	800273FC
Rte_Runnable_ComM_GetInhibitionStatus_Start	4.	GetInhibitionStatus	80028B0A	80027420
Rte_Runnable_ComM_GetMaxComMode_Start	5.	GetMaxComMode	80028B2A	80027446
Rte_Runnable_ComM_GetRequestedComMode_Start	6.	GetRequestedComMode	80028B4C	80027468
Rte_Runnable_ComM_LimitChannelToNoComMode_Start	7.	LimitChannelToNoComMode	80028B6C	8002748C
Rte_Runnable_ComM_LimitECUToNoComMode_Start	8.	LimitECUToNoComMode	80028B8C	800274B0
Rte_Runnable_ComM_PreventWakeUp_Start	9.	PreventWakeUp	80028BAC	800274D6
Rte_Runnable_ComM_ReadInhibitCounter_Start	10.	ReadInhibitCounter	80028BCE	800274FA
Rte_Runnable_ComM_RequestComMode_Start	11.	RequestComMode	80028BF0	8002751E
Rte_Runnable_ComM_ResetInhibitCounter_Start	12.	ResetInhibitCounter	80028C12	80027540
Rte_Runnable_ComM_SetECUGroupClassification_Start	13.	SetECUGroupClassification	80028C32	80027566
Rte_Runnable_Dcm_Dcm_MainFunction_Start	14.	Dcm_MainFunction	80028C54	80027588
Rte_Runnable_Dcm_GetActiveProtocol_Start	15.	GetActiveProtocol	80028C74	800275AE
Rte_Runnable_Dcm_GetRequestKind_Start	16.	GetRequestKind	80028C96	800275D0
Rte_Runnable_Dcm_GetSecurityLevel_Start	17.	GetSecurityLevel	80028CB6	800275F6

See also

- ■ TASK.List
- ■ TASK.List.tasks
- ■ TASK.Create.RUNNABLE
- ■ <trace>.Chart.RUNNABLE
- ■ <trace>.EXPORT.ARTI
- ■ <trace>.STATistic.RUNNABLE
- ▲ 'Overview of TRACE32 Command Structure' in 'Application Note Profiling on AUTOSAR CP with ARTI'

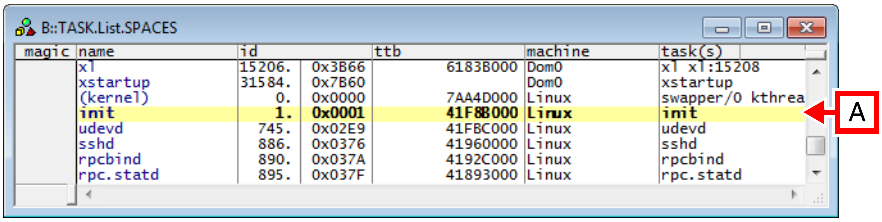
TASK.List.SPACES List MMU spaces

Format:	**TASK.List.SPACES**

Lists all MMU spaces known to the debugger. MMU spaces usually refer to processes in an OS/RTOS environment. MMU spaces are only available if **SYStem.Option.MMUSPACES** is set to **ON**.

For several purposes, the debugger needs to know which MMU spaces are active in the system. The debugger uses the kernel specific awareness to read out all space characteristics that it needs for its operation. **TASK.List.SPACES** shows the space characteristics that the debugger uses.

Each kernel specific awareness has a different display command to show the active processes with the characteristics that are essential to the specific kernel. Please see the appropriate **OS Awareness Manual** (rtos_*<os>*.pdf) for this command.



A The MMU space that is currently active on the selected core is marked.

Description of Columns in the TASK.List.SPACES Window:

magic	Space magic number. Unique number for the space. Usually the address of the control block structure.
name	Name of the object, if available.
id	ID of the object, if available.
machine	Machine name or machine ID if a hypervisor system is set up.
ttb	TTB address of this space
task(s)	Tasks running in this space

See also

- TASK.List
- TASK.List.tasks
- TASK.Create.SPACE

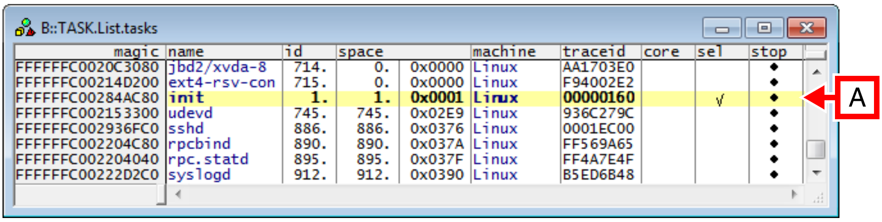
TASK.List.tasks — List all running tasks

Format:	**TASK.List.task**

Lists all tasks known to the debugger. Additional information about machines and MMU spaces is only displayed if **SYStem.Option.MMUSPACES** and **SYStem.Option.MACHINESPACES** are set to **ON**.

For several purposes, the debugger needs to know which tasks are active in the system. The debugger uses the kernel specific awareness to read out all task characteristics that it needs for its operation. **TASK.List.tasks** shows the task characteristics that the debugger uses.

Each kernel specific awareness has a different display command to show the active tasks with the characteristics that are essential to the specific kernel. Please see the appropriate **OS Awareness Manual** (rtos_*<os>*.pdf) for this command.



A The task that is currently running on the selected core is marked.

Description of Columns in the TASK.List.tasks Window:

magic	Task magic number. Unique number for the task. Usually the address of the control block structure.
name	Name of the object, if available.
id	ID of the object, if available.
space	Space name or ID if the OS uses MMU spaces.
traceid	ID that identifies an object in the trace list.
core	Identifies in SMP systems at which core this task runs.
sel	Task selected for debugging (only in Run Mode Debugging).
stop	Task selected to stop on break (only in Run Mode Debugging).
machine	Machine name or machine ID if a hypervisor system is set up.

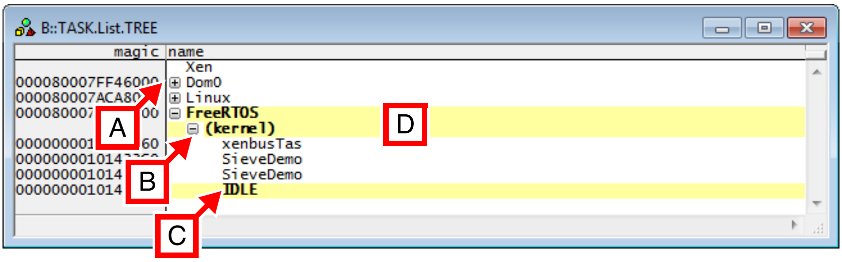
See also

■ TASK.List ■ TASK.List.MACHINES ■ TASK.List.RUNNABLES ■ TASK.List.SPACES
■ TASK.List.TREE ■ TASK.Create.task ■ CORE.List
▲ 'Overview of TRACE32 Command Structure' in 'Application Note Profiling on AUTOSAR CP with ARTI'

TASK.List.TREE Display tasks in a tree structure

Format:	**TASK.List.TREE** [/*<option>*]
<option>:	**Machine** *<machine_magic>* | *<machine_id>* | *<machine_name>*

Displays machines, MMU spaces, and tasks in the form of a tree structure.



A Level 1 of the tree: Machines.

B Level 2: MMU spaces.

C Level 3: Tasks.

D Yellow lines: The machine, the MMU space, and the task that are currently running on the selected core are marked.

Description of Columns in the TASK.List.TREE Window

magic	Magic number. Unique number for each object (machine/MMU space/task). Usually the address of the control block structure.
name	Name of the object, if available.

See also

■ TASK.List ■ TASK.List.tasks

▲ 'Release Information' in 'Legacy Release History'

TASK.ListID List virtual tasks

Format:	**TASK.ListID**

Opens the **TASK.ListID** window, displaying virtual tasks created with**TASK.CreateID** or **TASK.CreateExtraID**.

See also

■ TASK

TASK.NAME — Translation of task magic number to task name

Several windows of the OS Awareness show task-related information, e.g. **TASK.STacK** or **Trace.Chart.TASK**. Internally, the OS Awareness always uses the task magic numbers to identify a task. When displaying the task-related information, the debugger can translate this task magic number into a more readable task name, using a task name translation table. If the debugger finds an entry with the appropriate task magic number, it shows the task name instead of the task magic number (or task ID).

The translation table can be populated manually or automatically. If the TASK configuration file supports it, the debugger automatically populates the table with the current available task magic numbers and their names. Additionally, or if no configuration file exists, or if the configuration doesn't support task names, table entries may be added manually. If a manual entry and an automatic entry have the same task magic number, the manual entry overwrites the automatic one.

See also

- TASK.NAME.DELete
- TASK.NAME.RESet
- TASK.NAME.Set
- TASK.NAME.view
- TASK

TASK.NAME.DELete — Delete a task name table entry

Format:	**TASK.NAME.DELete** *<task_magic>*

Deletes the entry, specified by *<task_magic>*, from the task name translation table. If the entry is an automatic entry, the next usage of task names may add the automatic entry again.

See also

- TASK.NAME
- TASK.NAME.view

TASK.NAME.RESet — Reset task name table

Format:	**TASK.NAME.RESet**

Erases the whole task name translation table. If the TASK configuration file supports task name evaluation, the next usage of task names will populate the table again with automatic entries.

See also

- TASK.NAME
- TASK.NAME.view

TASK.NAME.Set

Set a task name table entry

Format:	**TASK.NAME.Set** *<task_magic> <task_name>*

Adds a manual entry to the task name translation table.

<task_magic>, *<task_name>*	The string specified by *<task_name>* is assigned to the task specified by *<task_magic>*. If the table contains already an automatic entry for the specified task magic number, it will be overwritten by the new entry!

Example:

```
TASK.NAME.Set 0x58D68 "My_Task 1"
```

See also

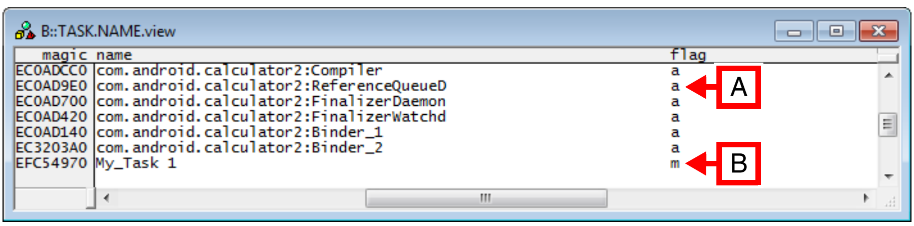
- TASK.NAME
- TASK.NAME.view

TASK.NAME.view

Show task name translation table

Format:	**TASK.NAME.view**

Shows the contents of the task name translation table.



A Flag “a”: The entry was set automatically by the TASK configuration file.

B Flag “m”: The entry was set manually by the **TASK.NAME.Set** command.

See also

- TASK.NAME
- TASK.NAME.DELete
- TASK.NAME.RESet
- TASK.NAME.Set

See also

- TASK.ORTI.CPU
- TASK.ORTI.load
- TASK.ORTI.NOSTACK
- TASK.ORTI.SPLITSTACK
- TASK

▲ 'Release Information' in 'Legacy Release History'
▲ 'Configuration' in 'OS Awareness Manual OSEK/ORTI'

TASK.ORTI.CPU — Set OSEK SMP CPU number

Format:	**TASK.ORTI.CPU** *<cpu_id>*

If TRACE32 is set up in AMP mode (one PowerView instance for each core), it assigns a CPU ID to each individual core, starting with zero. An AUTOSAR/OSEK operating system in SMP mode may assign a different CPU ID to the cores, depending how the OS uses the chip.

This command instructs the debugger to use the given CPU ID when extracting core dependent information from the ORTI file.

See also

- TASK.ORTI

TASK.ORTI.load — Configure OS Awareness for OSEK/ORTI

Format:	**TASK.ORTI.load** *<file>*

Configures the OS Awareness for AUTOSAR/OSEK operating systems using ORTI. For a detailed description, please refer to the chapter **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf).

See also

- TASK.ORTI

Format:	**TASK.ORTI.NOSTACK** *<task_name>*

When using the OS Awareness for ORTI (see **TASK.ORTI.load**), this command excludes a task from all stack evaluations, e.g. when performing a trace function analysis. Usually used for the idle routine if it isn't running as a separate task.

See also

- TASK.ORTI

Format:	**TASK.ORTI.SPLITSTACK** *<task_name>*

Some AUTOSAR/OSEK OSs use the same magic (NO_TASK) for the idle ORTI tasks on all cores. However, for the function analysis, the idle tasks need to be split to the individual cores because the cores are executing the idle tasks concurrently.

The command **TASK.ORTI.SPLITSTACK** splits the stacks of the idle ORTI tasks to the individual cores.

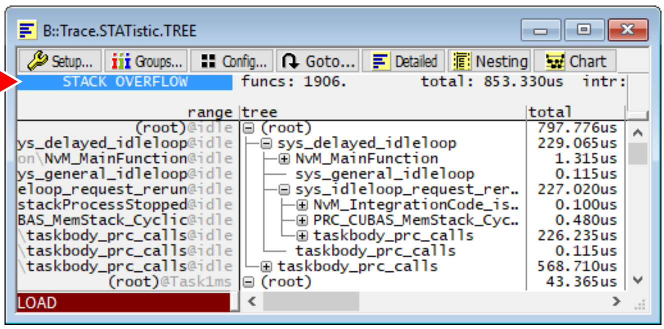
<task_name>	Specify the name of the idle ORTI task.

Example:

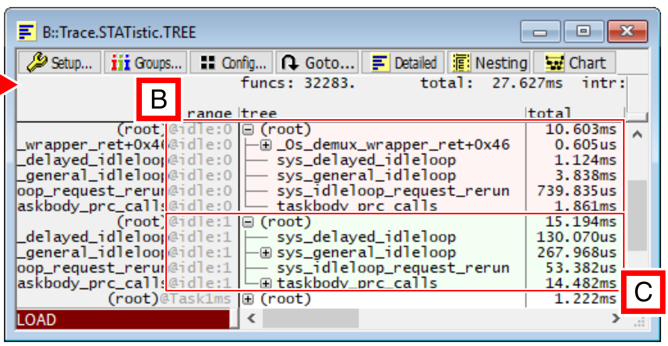
```
TASK.ORTI.SPLITSTACK "idle"
```

Output: Function analysis in the **<trace>.STATistic.TREE** window

Before A



After A



A Note that there is no stack overflow **after** executing **TASK.ORTI.SPLITSTACK**.

B The core numbers, here `0` and `1`, are appended to the name of the idle task `idle:0` and `idle:1`

C Core coloring scheme, e.g. green for core 1. See also **CORE.SHOWACTIVE**.

See also

- TASK.ORTI

TASK.RELOAD

Reread task list

Format:	**TASK.RELOAD**

This command initiates a reloading of the task list and enables the OS Awareness.

The OS Awareness may be disabled if an access to the current task fails, or if the system state changed, to prevent the debugger from accessing faulty memory. TASK.RELOAD explicitly re-enables the OS Awareness and initiates the update of the internal task list.

See also

- TASK

TASK.RESet

Reset OS Awareness

Format:	**TASK.RESet**

Resets the OS Awareness.

The configuration is cleared, all additional commands and features are removed.

See also

- TASK

Format:	**TASK.RUN** *<process>*

Loads *<process>* and prepares it for debugging.

Only applicable if GDB (Linux) or TRK (Symbian) is used as debug agent.

Example:

```
TASK.RUN /bin/hello
```

See also

■ TASK

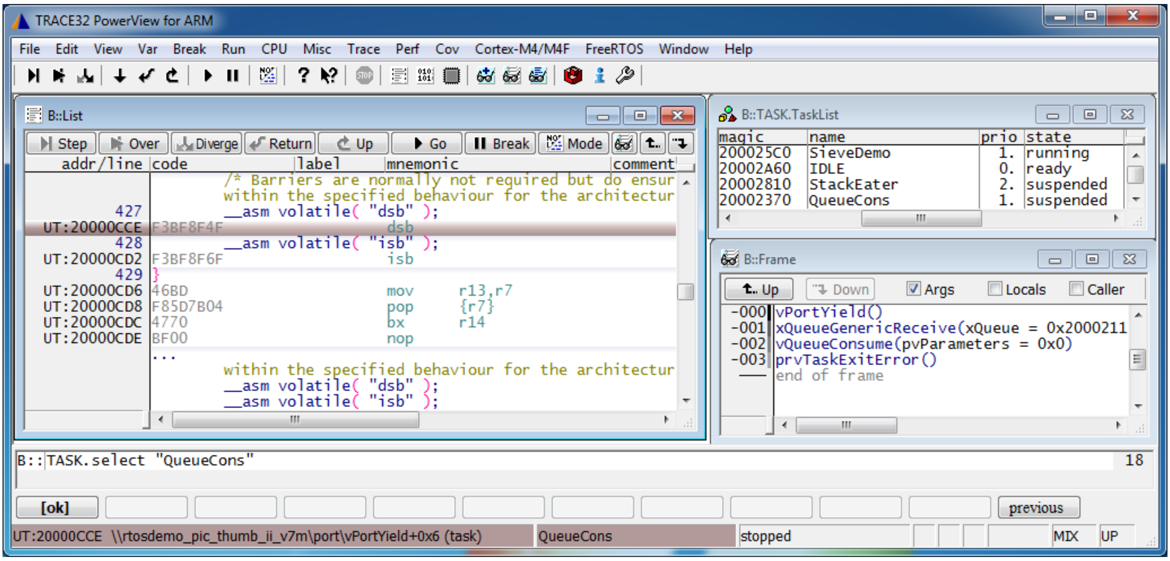
▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'
▲ 'Native Process Debugger Specific TASK Commands' in 'Native Process Debugger'

Format:	**TASK.select** *<task_magic>* | *<task_id>***.** | **"***<task_name>***"**

Stop mode debugging: In the case of an SMP system the currently selected core is changed to the core running the specified task. As a result the debugger view is changed to this core and all TRACE32 commands without **/CORE** *<number>* option apply to it.

If the specified task is not running, TRACE32 reads the register set of the specified task from the OS data structures. This is needed to display the context of the specified task in the TRACE32 PowerView GUI.

The TRACE32 state line changes to a reddish look-and-feel (see screenshot below) to indicate that the context of a not-running task is displayed. TRACE32 display commands such as **List.auto**, **Register.view**, **Frame.view** or **Var.Local** apply to this task. Whereas all other commands switch back to the currently running task before they are executed.



If the task is running on different virtual machine, TRACE32 reads the context of the VCPU that is processing the task on this machine.

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

Run mode debugging: Selects the specified task for debugging (e.g. GDB (Linux) or TRK (Symbian)).

```
TASK.select 41.
```

See also

■ TASK ■ CORE.select

▲ 'Release Information' in 'Legacy Release History'
▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'

TASK.SETDIR — Set the awareness directory

OS awarenesses: Linux only

Format:	**TASK.SETDIR** *<path>*

The Linux awareness and menu call scripts from the awareness directory. This directory is set per default to ~~/demo/*<arch>*/kernel/linux/*<linux_version>*. When loading the awareness outside this directory, TRACE32 prints a warning. With this command you can change the awareness directory. Scripts will be called then from the new directory.

See also

■ TASK

TASK.STacK

Stack usage coverage

The **TASK.STacK** command group allows to watch the stack usage in single tasking and multi-tasking systems. In single tasking systems, or in non supported operating systems, the user has to specify the stack area manually. The task magic number can be any number to identify a stack area.

In configured RTOS operation, the magic number must be the respective task magic number.

The debugger tries to get the current stack pointer. If the OS Awareness is configured, and the configuration file supports stack coverage, the current stack pointer is read out of the task control block of the application. When the application is stopped, the stack pointer is read from register and displayed at the current running task. Without any RTOS configuration the stack pointer will be displayed at the stack that fits to the pointer (pointer inside the stack). If no stack fits, or if the running task could not be found, the stack pointer of the register is displayed in an extra line. (See also **TASK.STacK.view**)

To evaluate the maximum stack space, the debugger uses a pattern search. Note, that the stack has to be initialized with a know pattern by the target application. The debugger searches from stack top to stack bottom for the first byte, that is not equal to the specified pattern. (See also **TASK.STacK.PATtern**)

For more information on stack coverage in operating systems, refer to the **OS Awareness Manuals**.

See also

- TASK.STacK.ADD
- TASK.STacK.DIRection
- TASK.STacK.Init
- TASK.STacK.PATtern
- TASK.STacK.PATternGAP
- TASK.STacK.ReMove
- TASK.STacK.RESet
- TASK.STacK.view
- TASK

TASK.STacK.ADD

Add stack space coverage

Format:	**TASK.STacK.ADD** [*<task_magic>* [*<stackrange>*]] [**/***<option>*]
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

With the 1st argument: Adds a stack area to the **TASK.STacK.view** window.

Without the 1st argument: Opens the **TASK.STacK.ADD** window. Double-click the entry of a stack area you want to add to the **TASK.STacK.view** window.

When no OS Awareness is loaded:

<task_magic>, *<stackrange>*	The task magic number is any number used to identify a stack area. In this case the stack range must be specified as a second parameter. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).

When an OS Awareness is loaded:

<task_magic>	The magic number **must** be the task magic number. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).
<stackrange>	If the extension definition file supplies automatic stack range detection (only possible in some OS's), then the stack range parameter can be omitted. Otherwise specify the stack area manually. If available, you can omit the magic and select a task from a task list.

In hypervisor-based environments:

MACHINE	Add only stack areas that belong to the selected machine. See also **"What to know about the Machine Parameters"** (general_ref_t.pdf).

Examples

Example 1: When no OS Awareness is loaded

```
TASK.STacK.ADD 2 0x1000--0x1fff
```

Example 2: When an OS Awareness is loaded

```
TASK.STacK.ADD 0x101433C0
```

Example 3: In a hypervisor-based environment

```
TASK.STack.ADD 0x101433C0 /MACHINE 3
```

See also

- TASK.STacK
- TASK.STacK.view

TASK.STacK.DIRection — Define stack growth direction

Format:	**TASK.STacK.DIRection** [**UP** | **DOWN**]

Defines whether the stack grows downwards or upwards.

DOWN	The stack starts with the high address and grows to a lower address.
UP	The stack starts with the low address and grows to a higher address.

See also

- TASK.STacK
- TASK.STacK.view

TASK.STacK.Init — Initialize unused stack space

Format:	**TASK.STacK.Init** [*<task_magic>*]

Overwrites the currently unused stack space with the pattern defined by **TASK.STacK.PATtern**. The memory starting from the stack pointer onto the stack boundary address (equaled the low address, if the stack grows downwards) will be initialized with the pattern.

CAUTION:	If the stack is used in an unusual way, e.g. some stack space is used even if the stack pointer does not point behind the used area, relevant target data may be overwritten, and your application may crash.

See also

- TASK.STacK
- TASK.STacK.view

Format:	**TASK.STacK.PATtern** [[**%**<*format*>] <*pattern*>] [**/**<*option*>]
<*option*>:	**TASK** <*task_magic*> | <*task_id*> | <*task_name*>

Defines stack pattern for stack coverage calculation.

<*pattern*>	Stack coverage calculation is done by comparing the stack data with defined pattern. The pattern must be the value, which represents unused stack space. This will only work, if the stack space is initialized with this value. Use **TASK.STacK.Init** to re-initialize currently unused stack space with the pattern. <*pattern*> can also be a string enclosed in quotes.
<*format*>	Use a <*format*> to define formats other than bytes e.g. **%Long**.
TASK	Sets the stack pattern only for the given task.

See also

- TASK.STacK
- TASK.STacK.view

TASK.STacK.PATternGAP — Define check pattern gap

Format:	**TASK.STacK.PATternGAP** [*<value>*]

If the stack check pattern defined with **TASK.STacK.PATtern** is not contiguous, this command defines the gap between two consecutive patterns.

<value>	Number of bytes between two consecutive stack check patterns.

Example: If the stack is pre-filled with a 4-byte pattern 0xdeadbeef on each 64byte boundary, specify:

```
TASK.STacK.PATtern %Long 0xDEADBEEF
TASK.STacK.PATternGAP 0x40-4
```

See also

- TASK.STacK
- TASK.STacK.view

TASK.STacK.ReMove — Remove stack space coverage

Format:	**TASK.STacK.ReMove** [*<task_magic>*] [**/***<option>*]
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

With the 1st argument: Removes a stack area from the **TASK.STacK.view** window.

Without the 1st argument: Opens the **TASK.STacK ReMove** window. Double-click the entry of a stack area you want to remove from the **TASK.STacK.view** window.

<task_magic>	Specify the task magic number of the task whose stack area you want to remove. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).
MACHINE	Removes only stack areas that belong to the selected machine. See also **"What to know about the Machine Parameters"** (general_ref_t.pdf).

Example:

```
TASK.STacK.ReMove 0x10147420
```

See also

■ TASK.STacK ■ TASK.STacK.view

TASK.STacK.RESet Reset stack coverage

Format:	**TASK.STacK.RESet** [*<task_magic>*]

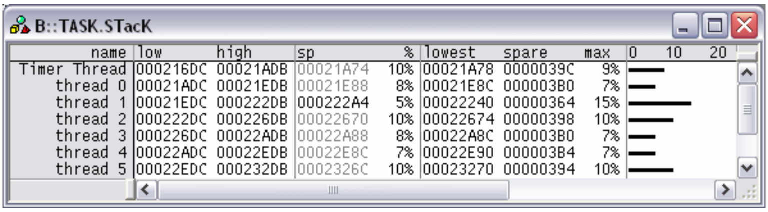
Resets the stack coverage system and all manually defined stack areas.
Resets the defined pattern to zero.

See also

■ TASK.STacK ■ TASK.STacK.view

Format:	**TASK.STacK.view** [*<task_magic>* [*<stackrange>*]] [**/***<option>*]
<option>:	**HumanReadable** **MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*

Opens a window with stack space coverage.



<task_magic> In single-tasking systems, or in non-supported multitasking systems, you have to specify the first stack manually. Use any task magic number as an ID, and specify the stack range to cover.

<stackrange> If the RTOS configuration file supports detection of the stack range, you can use the magic of a specific task and omit the stack range. The range will be automatically calculated from the information of the operating system. In the case of a fully supported operating system, you can start the window without any parameter. The debugger then automatically adds all current active tasks with its stacks to the window.

HumanReadable Shows the size of the stack and the spare stack memory in human readable form (byte, kilobytes, megabytes).

MACHINE Shows only the stacks of the selected machine.

See also **"What to know about the Machine Parameters"** (general_ref_t.pdf).

Description of Columns in the TASK.STacK.view Window

Column	Description
name	Name or ID for the stack space. In configured RTOS environment it specifies the name or ID of the task.
low and **high**	The lowest and highest address of the stack range.
sp (gray)	Gray: The stack pointer, calculated from a task control block (if available).

Column	Description
sp (black)	Black: The current value of the stack pointer register when the application is halted. In non-configured systems, the black value is displayed at the stack, where the current sp fits inside the stack borders. In configured RTOS systems the sp is shown at the current running task. If no according stack could be found, sp appears in an extra line at the end.
sp (red)	Red: Either if the current sp does not fit into the stack range of the current task, or if the sp fits into a stack range that is not the current task.
%	Percentage of the currently used stack space.
lowest	The lowest used stack address. If using the flag system, it shows the address, at which the first write flag in the stack area appears. If using pattern check, it shows the first address, at which the date is not equal to the pattern.
spare	Amount of bytes not used in the stack area.
max	The maximum stack space used in percent (calculated from 'lowest'). The following bar shows this percentage graphically.

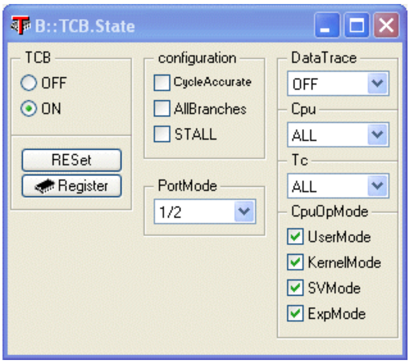
See also

- TASK.STacK
- TASK.STacK.ADD
- TASK.STacK.DIRection
- TASK.STacK.Init
- TASK.STacK.PATtern
- TASK.STacK.PATternGAP
- TASK.STacK.ReMove
- TASK.STacK.RESet

TCB Trace control block

The TCB (**T**race **C**ontrol **B**lock) is the HW control interface to the **MIPS** hardware trace block. For details please refer to the MIPS Trace specifications.

For configuration, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **TCB.state** window:



In the following, TCB specific controlling and associated commands are described.

See also

- TCB.AllBranches
- TCB.CycleAccurate
- TCB.EX
- TCB.IM
- TCB.KE
- TCB.OFF
- TCB.PCTrace
- TCB.PortWidth
- TCB.RESet
- TCB.SRC
- TCB.state
- TCB.SyncPeriod
- TCB.ThreadSizeBits
- TCB.UM
- TCB.CPU
- TCB.DataTrace
- TCB.FCR
- TCB.InstructionCompletionSizeBits
- TCB.LSM
- TCB.ON
- TCB.PortMode
- TCB.Register
- TCB.SourceSizeBits
- TCB.STALL
- TCB.SV
- TCB.TC
- TCB.Type
- TCB.Version

TCB.AllBranches

Broadcast all branches

Format:	**TCB.AllBranches** [**ON** | **OFF**]

OFF (default)	The TCB broadcasts only the address information when the processor branches to a location that cannot be directly inferred from the source code.
ON	The TCB broadcasts the address information for all branches or jumps.

See also

- TCB
- TCB.state

TCB.CPU

Broadcast information for specified CPU only

Format:	**TCB.CPU ALL** | *<cpu_x>*
<cpu_x>:	**CPU0** | **CPU1**

The TCB broadcasts only information for the specified CPU.

ALL (default)	The TCB broadcasts information for executed instructions of all active CPU's.
<cpu_x>	The TCB broadcasts only information for executed instructions of *<cpu_x>*.

See also

- TCB
- TCB.state

TCB.CycleAccurate

Cycle accurate tracing

Format:	**TCB.CycleAccurate** [**ON** | **OFF**]

Cycle accurate tracing can be used to observe the exact number of cycles that a particular code sequence takes to execute. If cycle accurate tracing is used, trace information is generated for each clock cycle. In this case the *<core_clock>* can be used to calculate the timestamps for the trace information.

ON	The TCB broadcasts the information which instructions were executed, but additionally stall information. No timestamps are generated by TRACE32.
OFF (default)	The TCB broadcasts only the information which instructions were executed. Timestamps are generated by TRACE32.

Example:

```
TCB.CycleAccurate ON

Trace.CLOCK 500.MHz                     ; specify the <core_clock> as
                                        ; base for the trace timestamps

Trace.List                              ; display the trace information
```

See also

■ TCB ■ TCB.state

Format:	**TCB.DataTrace** *<def>*
<def>:	**ON** | **OFF** | **Address** | **ReadAddress** | **WriteAddress** | **Data** | **ReadData** | **WriteData** | **Read** | **Write**

The TCB broadcasts only specified address and data information.

ON	The TCB broadcasts all address and data information.
OFF (default)	The TCB broadcasts no address and data but only PC information.
Address	The TCB broadcasts all address information.
ReadAddress	The TCB broadcasts only address information in case of a read.
WriteAddress	The TCB broadcasts only address information in case of a write.
Data	The TCB broadcasts all data information.
ReadData	The TCB broadcasts only data information in case of a read.
WriteData	The TCB broadcasts only data information in case of a write.
Read	The TCB broadcasts address and data information in case of a read.
Write	The TCB broadcasts address and data information in case of a write.

See also

■ TCB ■ TCB.state

TCB.EX Broadcast exception level information

Format:	**TCB.EX** [**ON** | **OFF**]

If enabled the TCB broadcasts information for instructions executed on exception level.

ON (default)	The TCB broadcast information for executed instructions in exception operating mode.
OFF	The TCB does not broadcast information for executed instructions in exception operating mode.

See also

- TCB
- TCB.state

TCB.FCR Broadcast function call-return information

Format:	**TCB.FCR** [**ON** | **OFF**]

Enables broadcasting of function call-return information. This information is not treated within TRACE32 PowerView but has to be taken into account for trace decoding especially in case of a belated trace analysis.

See also

- TCB
- TCB.state

TCB.IM Broadcast instruction cache miss information

Format:	**TCB.IM** [**ON** | **OFF**]

Enables broadcasting of instruction cache miss information. This information is not treated within TRACE32 PowerView but has to be taken into account for trace decoding especially in case of a belated trace analysis.

See also

- TCB
- TCB.state

TCB.InstructionCompletionSizeBits — Specify size of completion message

Format:	**TCB.InstructionCompletionSizeBits** *<number>*

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

This command allows to specify how many bits are used in the trace stream dot instruction completion message.

See also

- TCB
- TCB.state

TCB.KE — Broadcast kernel mode information

Format:	**TCB.KE** [**ON** | **OFF**]

If enabled the TCB broadcasts information for instructions executed in kernel mode.

OFF	The TCB does not broadcast information for executed instructions in kernel operating mode.
ON (default)	The TCB broadcast information for executed instructions in kernel operating mode.

See also

- TCB
- TCB.state

TCB.LSM — Broadcast load store data cache information

Format:	**TCB.LSM** [**ON**	**OFF**]

Enables broadcasting of load store data cache miss information. This information is not treated within TRACE32 PowerView but has to be taken into account for trace decoding especially in case of a belated trace analysis.

See also

- TCB
- TCB.state

TCB.OFF — Switch TCB off

Format:	**TCB.OFF**

Disables TCB functionality.

See also

- TCB
- TCB.state

TCB.ON — Switch TCB on

Format:	**TCB.ON**

Enables TCB functionality.

See also

- TCB
- TCB.state

Format:	**TCB.PCTrace** [**ON** | **OFF**]

If enabled, the TCB broadcasts program counter trace information.

OFF	The TCB does not broadcast program counter trace information.
ON (default)	The TCB broadcast program counter trace information.

See also

- TCB
- TCB.state

TCB.PortMode

Specify trace clock ratio

Format:	**TCB.PortMode** *<trace_clock>***/***<cpu_clock>*
<trace_clock> **/***<cpu_clock>*:	**8/1** | **4/1** | **2/1** | **1/1** | **1/2** | **1/4** | **1/6** | **1/8**

Specifies the ratio between trace- and CPU clock in case of off-chip trace.

Example:

```
TCB.PortMode 1/2           ; <trace_clock> is one half of <core_clock>.
```

See also

- TCB
- TCB.state

TCB.PortWidth

Specify trace port width

Format:	**TCB.PortWidth** *<width>*
<width>:	**4** | **8** | **16** | **64**

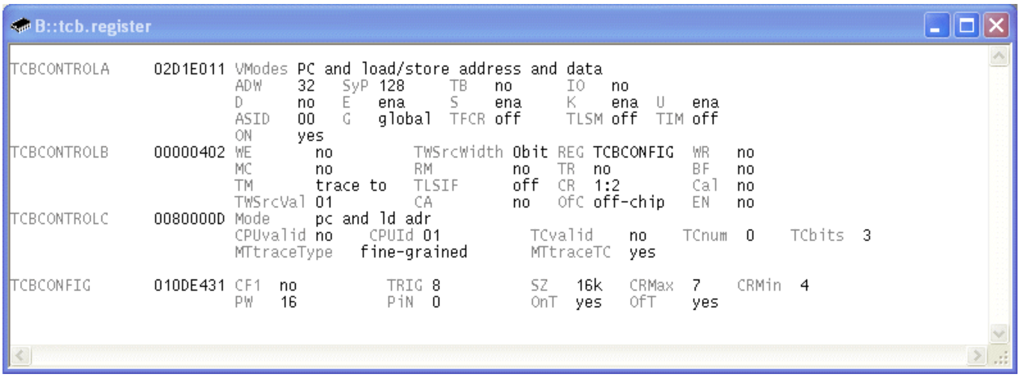
Specify the trace port width in number of bits. This value is determined automatically by selecting trace method or reading trace configuration register from target. Therefore this command should only be used for diagnosis purpose or if necessary belated trace analysis.

See also

- TCB
- TCB.state

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

Default: OFF.



<option>	For a description of the options, see **PER.view**.
Deport	Updates the control registers while the program is running (only possible if **SYStem.MemAccess Enable** is selected).

Example:

```
TCB.Register permipstcb.per          ; display the TCB control registers

                                     ; use the format description in
                                     ; permipstcb.per

TCB.Register, /SpotLight             ; display the TCB control registers

                                     ; mark changes on the registers
```

See also

- TCB
- TCB.state

TCB.RESet — Reset TCB setup to default

Format:	**TCB.RESet**

Resets the TCB settings to default.

See also

- TCB
- TCB.state

TCB.SourceSizeBits — Specify number of bit for core information in trace

Format:	**TCB.SourceSizeBits** *<number>*

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

This command allows to specify how many bits are used in the trace stream to identify the source core.

See also

- TCB
- TCB.state

TCB.SRC — Control selective trace

Format:	**TCB.SRC[***<n>***] ON** | **OFF**

Controls if the TCB broadcasts information for the specified SRC.

See also

- TCB
- TCB.state

Format:	**TCB.STALL** [**ON** | **OFF**]

If enabled, TCB broadcasts slow but complete trace information.

OFF	The TCB broadcasts trace information in real-time with the risk of broken trace flow.
ON (default)	The TCB stall CPU if necessary and broadcast always complete information.

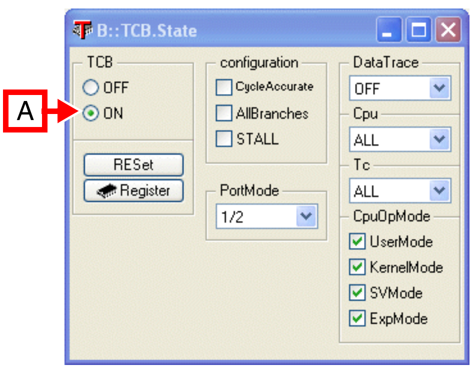
See also

- TCB
- TCB.state

TCB.state — Display TCB setup

Format:	**TCB.state**

Displays the TCB configuration window.



A For descriptions of the commands in the **TCB.state** window, please refer to the **TCB.*** commands in this chapter. **Example**: For information about **ON**, see **TCB.ON**.

See also

- TCB
- TCB.CPU
- TCB.DataTrace
- TCB.FCR
- TCB.InstructionCompletionSizeBits
- TCB.LSM
- TCB.ON
- TCB.AllBranches
- TCB.CycleAccurate
- TCB.EX
- TCB.IM
- TCB.KE
- TCB.OFF
- TCB.PCTrace

- TCB.PortMode
- TCB.PortWidth
- TCB.Register
- TCB.RESet
- TCB.SourceSizeBits
- TCB.SRC
- TCB.STALL
- TCB.SV
- TCB.SyncPeriod
- TCB.TC
- TCB.ThreadSizeBits
- TCB.Type
- TCB.UM
- TCB.Version

TCB.SV — Broadcast supervisor mode information

Format:	**TCB.SV** [**ON** | **OFF**]

If enabled the TCB broadcasts information for instructions executed in supervisor mode.

ON (default)	The TCB broadcast information for executed instructions in supervisor operating mode.
OFF	The TCB does not broadcast information for executed instructions in supervisor operating mode.

See also

- TCB
- TCB.state

TCB.SyncPeriod — Specify TCB sync period

Format:	**TCB.SyncPeriod** *<period>*
<period>:	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7**

Specify the period in cycles the TCB broadcasts a synchronization message.

<period>	The TCB sync period in 2 ^ (*<period>* + 5) cycles.

See also

- TCB
- TCB.state

TCB.TC — Broadcast information for specified HW thread

Format:	**TCB.TC ALL** | *<tc_x>*
<tc_x>:	**TC0** | **TC1** | **TC2** | **TC3** | **TC4** | **TC5** | **TC6** | **TC7** | **TC8**

The TCB broadcasts only information for the specified HW thread.

ALL (default)	The TCB broadcasts information for executed instructions of all active TCs.
<tc_x>	The TCB broadcasts only information for executed instructions of *<tc_x>*.

See also

- TCB
- TCB.state

TCB.ThreadSizeBits — Specify number of bit for thread information in trace

Format:	**TCB.ThreadSizeBits** *<number>*

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

This command allows to specify how many bits are used in the trace stream to identify the source thread context.

See also

- TCB
- TCB.state

TCB.Type

Specify TCB type

Format:	**TCB.Type** | *<tcb_type>*
<tcb_type>:	**PD** | **PD74K** | **IFLOW** | **FALCON** | **ZEPHYR**

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

PD	MIPS standard program and data trace control block.
PD74K	Specific MIPS74K program data trace control block.
IFLOW	MIPS standard instruction flow trace control block.
FALCON	Lantiq specific instruction flow trace control block.
ZEPHYR	Broadcom specific program and data trace control block.

See also

- TCB
- TCB.state

TCB.UM

Broadcast user mode information

Format:	**TCB.UM** [**ON** | **OFF**]

If enabled the TCB broadcasts information for instructions executed in user mode.

ON (default)	The TCB broadcast information for executed instructions in user operating mode.
OFF	The TCB does not broadcast information for executed instructions in user operating mode.

See also

- TCB
- TCB.state

TCB.Version

Specify trace cell version

Format:	**TCB.Version** *<number>*

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated trace analysis. This command allows to specify manually the version number of the TCB trace cell. The version number must fit to the TCB the trace data have been recorded with. It could be found in the header of the TCB window if TRACE32 is connected to the referring target.

See also

- TCB
- TCB.state

TERM Terminal emulation

See also

- TERM.CLEAR
- TERM.CLOSE
- TERM.CMDLINE
- TERM.GATE
- TERM.HARDCOPY
- TERM.HEAPINFO
- TERM.LocalEcho
- TERM.METHOD
- TERM.METHOD2
- TERM.Mode
- TERM.Out
- TERM.OutBREAK
- TERM.PIPE
- TERM.PipeREAD
- TERM.PipeWRITE
- TERM.PULSE
- TERM.READ
- TERM.RESet
- TERM.SCROLL
- TERM.SIZE
- TERM.STDIN
- TERM.TCP
- TERM.TELNET
- TERM.TRIGGER
- TERM.view
- TERM.WRITE
- TERM.LINE()
- TERM.NEWHANDLE()
- TERM.READBUSY()
- TERM.RETURNCODE()
- TERM.TRIGGERED()

▲ 'TERM Functions (Terminal Window)' in 'General Function Reference'
▲ 'Release Information' in 'Legacy Release History'

Overview TERM

Multitasking operating systems or monitor programs running on the target system often need a terminal interface for operation. This interface can be implemented either using peripherals (e.g. serial port) or as a memory based interface. The memory based interface can work in several operation modes. It can communicate either on character basis or with blocks of up to 255 bytes length. The memory access can either be made while the target is running (when the system supports such run-time memory accesses) or only when the target is stopped.

When the EPROM simulator (ESI) is used, the ESI can be used as communication port as well. Some processor architectures also provide a special communication interface which is accessible through the BDM/JTAG port (DCC modes).

The standard terminal window provides only the basic functions **Backspace**, **Return** and **LineFeed**. A VT100 emulation mode is also available. A character can only be entered when the cursor is positioned in an active window. The terminal window may also be used for "virtual hosting". This allows to access some basic operation system functions and the file system of the host from the target program. This functionality is only available in the **TERM.GATE** command.

Interface Routines

In this section:

- EPROM Simulator
- Single Character Modes
- Buffered Modes
- Serial Line Debugger
- Special Hardware, JTAG

This is an example in C to access the terminal window. The address of the ports depends on the width and location of the EPROMs. The example assumes 8-bit wide EPROMs. For 16-bit EPROMs the addresses must be doubled and the types changed from char to short.

```
extern volatile unsigned char input_port at 0x1000;
extern volatile unsigned char status_port at 0x1400;
extern volatile unsigned char output_port[256] at 0x1800;

void char_out(c)
unsigned char c;
{
     unsigned char dummy;
     if ( c == 0 )                /* refuse to send 0 (break) character */
          return;
     while ( status_port&2);      /* wait until port is free */
     dummy = outport_port[c];     /* send character */
}

int char_in();
{
     unsigned char c;             /* wait until character is ready */
     while (!(status_port&1));    /* read character */
     c = input_port;              /* manual break executed ? */
     if ( c == 0 )
          break_emulation();
     return c;
}
```

Interface Routines (Single Character Modes)

This interface occupies two memory cells in which characters can be transferred. A zero means that no character is available and the interface is ready. When the target is not able to provide a dual-ported memory access it is possible to stop the target after it has placed a character in the communication area and the terminal command will restart the target automatically after it has processed the character.

This is an example in C to access the terminal window. By changing the char_in and char_out routines within the library, all more complex functions like printf() or scanf() are redirected to the terminal window.

NOTE:	Some emulation heads have special dual-port access modes, that require special cycles to be executed (e.g. IDLE mode on H8 probes).

```
extern volatile char input_port,
output_port

      void char_out(c)
 char c;
 {
      while (output_port != 0 ) ;      /* wait until port is free */
      output_port = c;                 /* send character */
 }
 int char_in();
 {
      char c;
      while ( input_port == 0 ) ;      /* wait until character is ready */
      c = input_port;                  /* read character */
      input_port = 0;                  /* clear input port */
      return c;
 }
```

Interface Routines (Buffered Modes)

An example for using the buffered mode can be found in ~~/demo/etc/terminal/t32term/t32term_memory.c. This example contains also examples for using the virtual hosting feature of the **TERM.GATE** command.

Interface Routines (Serial Line Debugger)

The serial line can be used as usual. Only the data values 0 have a special meaning. Receiving such a value means an emulation break. Sending such a value is not allowed for the user program.

Interface Routines (Special Hardware, JTAG)

Check the target appendix for your processor for details and availability.

Functions

Refer to **“TERM Functions (Terminal Window)”** in General Function Reference, page 365 (general_func.pdf).

Fast Data Write

The fast data write system allows to transfer data from the target to a file on the host. The data transfer rate can be up to 250 KBytes/s. The max. reaction time is 50 µs when the transfer is not interruptible or 150 µs when the transfer is interruptible. Data can be transferred either 8, 16 or 32 bit wide. The principle is similar to the terminal emulation. The interface occupies two memory cells, one byte to control the transfer and a second byte or word to hold the data. A zero in the control cell means that the debugger is ready to accept data. Writing a '01' by the CPU causes the data to be transferred to the host. Writing '02' saves the current data buffer to the host. The time required by this disk save dependents on the host and communication speed. The data buffer is saved automatically after the buffer is full. The value '03' can be used as a NOP command to wait for the start of the transfer. Writing 'ff' terminates the data transfer. The Fast Data Write system has been replaced by the FDX system.

Interface Routines

This is an example in C to access the fast data transfer.

```
extern volatile char control_port;
extern volatile short data_port;

void word_out()
short c;
{
     while (control_port != 0) ;       /* wait until port is free */
     data_port = c ;                    /* place 16 bit in buffer */
     control_port = 1;                  /* send data to buffer/host */
}

int begin_transfer(c);
short c;
{
     while (control_port != 0) ;       /* wait until transfer is ready */
}

int end_transfer(c);
short c;
{
     while (control_port != 0) ;       /* wait until port is free */
     control_port = 0xff;               /* stop transfer program */
}
```

TERM.CLEAR

Clear terminal window

Format:	**TERM.CLEAR** [*<channel>*] **TERM.CLEAR** [*<address>*] (deprecated)
<channel>:	**#***<number>*

Clears the terminal window and places the cursor to the home position.

See also

- TERM
- TERM.view

TERM.CLOSE

Close files

Format:	**TERM.CLOSE** [*<channel>*] **TERM.CLOSE** [*<address>*] (deprecated)
<channel>:	**#***<number>*

Closes the output file created with **TERM.WRITE**.

See also

- TERM
- TERM.view

TERM.CMDLINE

Specify a command line

Format:	**TERM.CMDLINE** *<cmdline>*

The command can specify a command line for the SYS_GET_CMDLINE (0x15) system call if ARM compatible semihosting is used.

See also

- TERM
- TERM.view

TERM.GATE

Terminal with virtual hosting

Format:	**TERM.GATE** [*<channel>*] **TERM.GATE** [*<addresses>*] (deprecated)
<channel>:	**#***<number>*
<addresses>:	[*<address_out>*] [*<address_in>*]

TERM.GATE allows to an application program running on a target processor to communicate with the host computer of the debugger. This way the application can use the I/O facilities of the host computer like keyboard input, screen output, and file I/O. This is especially useful if the target platform does not provide these I/O facilities or in order to output additional debug information in printf() style. The implementation on target and settings in TRACE32 vary between targets, this is also not available for all platforms. Typically, you need a third party library like newlib on your target (which is usually part of the compiler toolchain) and correct **TERM.METHOD** settings in debugger. For more details, please refer to your **Processor Architecture Manual**.

See also

■ TERM ■ TERM.view

▲ 'Release Information' in 'Legacy Release History'

TERM.HARDCOPY

Print terminal window contents

Format:	**TERM.HARDCOPY** [*<channel>*]
<channel>:	**#***<number>*

Opens the **Print** dialog of the operating system. From the **Print** dialog, you can select a printer to make a hardcopy of the terminal window contents or print the terminal window contents to file.

See also

■ TERM ■ TERM.view

TERM.HEAPINFO Define memory heap parameters

Format: **TERM.HEAPINFO** [*<heap_base>*] [*<heap_limit>*] [*<stack_base>*] [*<stack_limit>*]

Defines the memory heap and stack locations returned by the ARM compatible semihosting calls. Only relevant when ARM compatible semihosting is used.

Please note that the heap grows toward higher memory addresses (heap_base < heap_limit) and the stack grows towards lower memory addresses (stack_base > stack_limit). *<heap_base>* = 0 advises the application to locate the heap at the top of the memory region.

See also

- TERM
- TERM.view

TERM.LocalEcho Enables/disables local echo for new terminal windows

Format: **TERM.LocalEcho** [*<channel>*] [**ON** | **OFF**]

<channel>: **#***<number>*

Defines, if terminal windows, which are opened after the **TERM.LocalEcho** command with the **TERM.view** or **TERM.GATE** command, will have a local echo or not.
Terminal windows with enabled local echo also show the transmitted characters in addition to the received characters.

See also

- TERM
- TERM.view

[Examples]

Format:	**TERM.METHOD** [*<channel>*] *<method>* **TERM.Protocol** (deprecated)
<channel>:	**#***<number>*
<method>:	**SingleE** [*<output>*] [*<input>*] [**/***<option>*] **BufferE** [*<output>*] [*<input>*] [**/***<option>*] **SingleC** *<pc>* [*<output>*] [*<input>*] [**/***<option>*] **BufferC** *<pc>* [*<output>*] [*<input>*] [**/***<option>*] **SingleS** [*<output>*] [*<input>*] [**/***<option>*] **BufferS** [*<output>*] [*<input>*] [**/***<option>*] **COM** [*<name>*] [*<baudrate>*] [*<bits>*] [*<parity>*] [*<stopbits>*] [*<handshake>*] / [**RTSDISabled** | **DTRDISabled**] **TCP** *<host>* [*<port>*] **PIPE** **DCC** [**/***<option>*] **DCC3** [**/***<option>*] **DCC4A** [**/***<option>*] **DCC4B** [**/***<option>*] **SIM** **CCIO** **BRK1_14** [*<address>*] [**/***<option>*] **ARMSWI** [*<address>*] [**/***<option>*] **RISCVSWI** [**/***<option>*] **CHORUS** **ESI** **SERIAL**
<input>: *<output>*:	*<address>*
<name>:	Windows: **COM1** | **COM2** | … | **COM9** alternatively (if COMx fails) and for ports >9: **\\.\COM1** | **\\.\COM2** | … | **\\.\COM10** | **\\.\COM11** | … Linux: path to device, e.g. **/dev/ttyS0** | **/dev/ttyS1** | **/dev/ttyUSB0** | …
<bits>:	**5** | **6** | **7** | **8**

<parity>:	**NONE** | **EVEN** | **ODD** | **MARK** | **SPACE**
<stopbits>:	**1STOP** | **2STOP**
<handshake>:	**NONE** | **RTSCTS** | **DTRDSR** | **XONXOFF**

Defines how data is exchanged between the target application and the debugger. On some targets additional processor specific modes may be available.

NOTE:	This command does not change the settings of already opened terminal windows. Therefore, if you want to change parameters of an existing one, close it and reopen it again.

<methods>	Description
SingleE	Single characters using real time access (e.g. Dualport)
BufferE	Buffered transfer using real time access
SingleS	Single characters using regular access at spot points.
BufferS	Buffered transfer using regular access at spot points.
BRK1_14 Only available for Xtensa	This is a CPU specific option for XTENSA. For more information, see **"CPU specific TERM.METHOD Command"** in Xtensa Debugger and Trace, page 66 (debugger_xtensa.pdf).
SingleC	Single characters, accessed when CPU is stopped. The additional parameter the PC location of the breakpoint that stops the CPU for communication.
BufferC	Buffered transfer, accessed when CPU is stopped.
ESI	Use the ESI for communication. This protocol can also be used when a BDM/JTAG debugger is used together with an ESI (EPROM simulator).
SERIAL	Use the serial (or ethernet) interface of the debug monitor to exchange data.
DCC	Use the DCC port of the JTAG interface (only on some architectures)
DCC3	Same as DCC, but transfer up to 3 characters at once.
DCC4A	Same as DCC, but transfer up to 4 ascii characters at once.
DCC4B	Same as DCC, but transfer always 4 characters at once.
ARMSWI	ARM compatible SWI bases semihosting via SWI breakpoint.
RISCVSWI	RISC-V compatible semihosting via semihosting trap instruction sequence (slli, ebreak, srai).

<methods>	Description
SIM	Terminal via simulator API.
COM	Serial interface of the host.
TCP	Routes terminal input/output to TCP port. See example below.

Parameters	Description
<output> *<input>*	Addresses of the output (target->debugger) and input (debugger->target) buffers for memory based terminals.
<host>	Host name or IP address of TCP terminal (TELNET)
<port>	TCP terminal port number (default: 23)
RTSDISabled	If RTS is not used for handshaking (<handshake>!=RTSCTS), by default RTS is permanently enabled. Use this option to permanently disable RTS.
DTRDISabled	If DTR is not used for handshaking (<handshake>!=DTRDSR), by default DTR is permanently enabled. Use this option to permanently disable DTR.

Examples:

```
TERM.METHOD BufferE Var.ADDRESS("messagebufferout") \
Var.ADDRESS("messagebufferin")
```

```
TERM.METHOD #1 BufferE Var.ADDRESS("messagebufferout") \
Var.ADDRESS("messagebufferin")
```

```
; Route terminal input/output from /dev/ttyUSB0 on LAB-PC with baudrate
; 115200 to TCP port 8765
$RemoteMachine> socat TCP-LISTEN:8765,fork,reuseaddr FILE:/dev/ttyUSB0\
,b115200,raw,echo=0
TERM.METHOD TCP LAB-PC 8765.
TERM
```

See also

■ TERM ■ TERM.view

▲ 'Release Information' in 'Legacy Release History'

Format:	**TERM.METHOD2** [*<channel>*] *<method>*
<channel>:	**#***<number>*
<method>:	**OFF** **ITM** *<itm ch>*

Defines an additional method for the target to send data to the terminal.

<method>	Description
OFF	Default setting. No additional method is configured.
ITM	Use data written to an ITM stimuli channel by the target. An ITM is present on many Arm Cortex-M chips. This requires that the ITM trace is captured via the CAnalyzer in STREAM or PIPE mode. Please refer to the demo PRACTICE script and application found at **~~/demo/arm/hardware/kinetis/kinetis_k/k60/itm_term_printf/**.

Parameters	Description
<itm ch>	ITM channel where terminal data is written to by the application. On most Cortex-M systems, channel **0** captures the data written to address **0xE0000000**, channel **1** captures the data written to address **0xE0000004**, and so on.

Example: This example assumes that an external trace (either via SWV or parallel trace) is already set up.

```
; Set up a primary method for the terminal. This example can be used
; even if DCC is not available.
TERM.RESet    #2
TERM.METHOD   #2 DCC
TERM.Mode     #2 STRING
TERM.SIZE     #2 80. 25. 200.

; Set up the ITM method and show the terminal window.
TERM.METHOD2 #2 ITM 0.
TERM.view     #2

; Set up and arm the trace. Due to a limitation of the Cortex-M
; infrastructure, the target must be running to generate proper
; synchronization packets on the trace port.
CAnalyzer.Mode PIPE ; STREAM would also work
CAnalyzer.AutoArm ON
Go.direct

; Write some text to the stimulus channel. This is only for
; demonstration purposes and should normally be done by the target
; application.
Data.Set E:0xE0000E00 %LE %Long 0xFFFFFFFF ; Enable stimuli channels
Data.Set E:0xE0000000 %LE %Long 0x6C6C6548 ; "Hell"
Data.Set E:0xE0000000 %LE %Long 0x57202C6F ; "o, W"
Data.Set E:0xE0000000 %LE %Long 0x646C726F ; "orld"
Data.Set E:0xE0000000 %LE %Word 0x2121     ; "!!"
Data.Set E:0xE0000000 %LE %Byte 0x0A       ; "\n"
```

See also

- TERM
- TERM.view

Format:	**TERM.Mode** [*<channel>*] [*<mode>*]
<channel>:	**#***<number>*
<mode>:	**ASCII** | **UTF8** | **STRING** | **STRING-UTF8** | **RAW** | **HEX** | **VT100** | **VT-UTF8**
<option>:	**CORE** *<corenumber>*

Defines the terminal type used for new terminal windows.

ASCII	Terminal behaves like a typewriter. CR and LF are evaluated.
UTF8	Support UTF-8 encoded characters. Terminal behaves like a typewriter. CR and LF are evaluated.
STRING	Terminal interprets data as single line strings. Needed for some Printf libraries. CR is ignored. LF is evaluated.
STRING-UTF8	Support UTF-8 encoded characters. Terminal interprets data as single line strings. Needed for some Printf libraries. CR is ignored. LF is evaluated.
RAW	Terminal shows the incoming data like an HEX/ASCII dump. E.g. Spaces, Tabs, CRs, LFs are displayed as special characters only. CR is ignored. LF is evaluated.
HEX	Terminal shows the incoming bytes as HEX values. CR and LF are ignored.
VT100	Terminal interprets the VT100 protocol. Color Codes are evaluated e.g. Linux bash like console. CR and LF are evaluated.
VT-UTF8	Support UTF-8 encoded characters. Terminal interprets the VT100 protocol. Color Codes are evaluated e.g. Linux bash like console. CR and LF are evaluated.

See also

- TERM
- TERM.view

▲ 'Release Information' in 'Legacy Release History'

TERM.Out

Send data to virtual terminal

Format:	**TERM.Out** [*<channel>*] *<string>* …
	TERM.Out [*<address_in>*] *<string>* … (deprecated)
<channel>:	**#***<number>*

Sends characters to a terminal. Can be used to control the terminal through a PRACTICE script (*.cmm) or to input non-printable characters from the command line.

Example:

```
;configure u-boot through serial terminal
TERM.METHOD #1 COM COM1 115200. 8 NONE 1STOP NONE
TERM.view   #1
TERM.Out    #1 10. ;send a single line feed
TERM.Out #1 "setenv bootcmd bootm 0xfe000000 0xfe800000 0xffe00000" 10.
TERM.Out #1 "setenv bootargs root=/dev/ram console=ttyS0,115200" 10.
TERM.Out #1 "saveenv" 10.
```

See also

■ TERM ■ TERM.view

TERM.OutBREAK

Send serial break

Format:	**TERM.OutBREAK** [*<channel>*]
<channel>:	**#***<number>*

Sends serial break to terminal.

See also

■ TERM ■ TERM.view

TERM.PIPE — Connect terminal to named pipe

Format:	**TERM.PIPE** [*<channel>*] *<pipename>* **TERM.PIPE** [*<address_out>*] [*<address_in>*] *<pipename>* (deprecated)
<channel>:	**#***<number>*

Connects the terminal to a bidirectional named pipe.

See also

- TERM
- TERM.view

TERM.PipeREAD — Connect terminal input to named pipe

Format:	**TERM.PipeREAD** [*<channel>*] *<file>* **TERM.PipeREAD** [*<address_in>*] *<file>* (deprecated)
<channel>:	**#***<number>*

Connects the terminal to a pipe which sends data to the host.

See also

- TERM
- TERM.view

TERM.PipeWRITE — Connect terminal output to named pipe

Format:	**TERM.PipeWRITE** [*<channel>*] *<file>* **TERM.PipeWRITE** [*<output>*] *<file>* (deprecated)
<channel>:	**#***<number>*

Connects the terminal to a pipe which receives data from the host.

See also

- TERM
- TERM.view

Format:	**TERM.PULSE** [*<channel>*] [**ON** | **OFF**]
<channel>:	**#***<number>*

Issues a pulse on the PODBUS trigger after each transfer. This pulse may be used to trigger an interrupt on the target system to trigger interrupt based communication.

See also

- TERM
- TERM.view

TERM.READ Get terminal input from file

Format:	**TERM.READ** [*<channel>*] *<file>* **TERM.READ** [*<address_in>*] *<file>* (deprecated)
<channel>:	**#***<number>*

The contents of the file are send to the terminal, defined by the optional address. The terminal must already exist to use this command. The **TERM.CLOSE** command closes the input file after or during transfer.

Example:

```
TERM.READ #1 key_input.in
```

See also

- TERM
- TERM.view

TERM.RESet Reset terminal parameters

Format:	**TERM.RESet** [*<channel>*]
<channel>:	**#***<number>*

Closes the I/O redirection files and set all parameters to default values.

See also

- TERM
- TERM.view

TERM.SCROLL Enable automatic scrolling for terminal window

Format:	**TERM.SCROLL** [*<channel>*] [**ON** | **OFF**]
<channel>:	**#***<number>*

Default: OFF.

Enables or disables automatic scrolling. With automatic scrolling enabled the visible window will follow the terminal cursor.

To enable the display of the scroll bar within the **TERM.view** window, it is necessary to configure **TERM.SIZE** accordingly.

See also

■ TERM ■ TERM.view

TERM.SIZE Define size of terminal window

Format:	**TERM.SIZE** [*<channel>*] [*<columns>*] [*<lines>*] [*<backlog_size>*]
<channel>:	**#***<number>*

Defines the size of the virtual terminal in lines and columns.

<backlog_size>	This value defines the lines of the backlog buffer. The backlog is updated whenever a line scrolls out of the “real” part of the **TERM.view** window.

See also

■ TERM ■ TERM.view
▲ ’Release Information’ in ’Legacy Release History’

TERM.STDIN Get terminal input from file

Format:	**TERM.STDIN** [*<channel>*] *<file>*
<channel>:	**#***<number>*

The contents of the file are send to the terminal, defined by the optional address. The terminal must already exist to use this command. The **TERM.CLOSE** command closes the input file after or during transfer. An EOF is returned, for some semihosting interfaces, when the file is transferred.

See also

■ TERM ■ TERM.view

TERM.TCP

Route terminal input/output to TCP port

Format:	**TERM.TCP** [*<channel>*] *<port>*
<channel>:	**#***<number>*

Routes terminal input/output to TCP port.

See also

- TERM
- TERM.view

TERM.TELNET

Open TELNET terminal window

Format:	**TERM.TELNET** [*<channel>*]
<channel>:	**#***<number>*

Opens the terminal emulation window for TELNET.

Example:

```
TERM.METHOD TCP 10.2.23.140 ;using default port 23
TERM.MODE VT100
TERM.TELNET
```

See also

- TERM
- TERM.view
- ▲ 'Release Information' in 'Legacy Release History'

[Example]

Format:	**TERM.TRIGGER** [*<channel>*] *<message_string>* **TERM.TRIGGER** [*<address_out>*] *<string>* (deprecated)
<channel>:	**#***<number>*

Sets a trigger for the occurrence of a specific string in the terminal window. The function **TERM.TRIGGERED()** returns if the trigger has occurred or not.

<channel>	Handle to refer to a terminal. A new handle can be created with **TERM.METHOD**.
<address_out>	Only required for memory-based data exchange (**SingleE**, **BufferE**, **SingleS**, **BufferS**).
<message_string>	Case sensitive. The message string or substring you want the **TERM.TRIGGER()** function to find in the **TERM.view** or **TERM.GATE** window.

Example: A typical use case might be to automatize the boot process. The following script stops the boot process after the string “Hit any key to stop autoboot” appears in the terminal window.

Example terminal output:

```
U-Boot <year>.<month>
CPU:   example CPU
Board: example Board
Boot:  SD-Card
DRAM:  2 GiB
MMC:   SDHC: 0
In:    serial
Out:   serial
Err:   serial
Normal Boot
Hit any key to stop autoboot: 3
```

Script that waits for the message "Hit any key" and boots the target:

```
;create terminal configuration and assign it to the handle #1
TERM.METHOD #1 COM COM3 115200. 8 NONE 1STOP NONE

;create the terminal and open the TERM.view window
TERM.view   #1

; STATE.RUN() -> STOPPED
Break

; wait for trigger with timeout, press ENTER
TERM.TRIGGER #1 "Hit any key"
; start CPU
Go
SCREEN.WAIT TERM.TRIGGERED(#1) 10.s
IF !TIMEOUT()
(
  TERM.OUT #1 0xA
  WAIT 0.1s
  TERM.OUT #1 "setenv bootargs ...."
)
ELSE
(
  ; error handler
)
```

See also

- ■ TERM
- ■ TERM.view
- ❏ TERM.TRIGGERED()
- ❏ TIMEOUT()

Format:	**TERM.view** [*<channel>*] **TERM.view** [*<address_out>*] [*<address_in>*] (deprecated)
<channel>:	**#***<number>*

Opens the terminal emulation window. The protocol of the terminal is defined through **TERM.METHOD**. For protocols based on memory based data exchange (**SingleE**, **BufferE**, **SingleS**, **BufferS**), the communication buffer addresses can either be specified with **TERM.METHOD** or directly with **TERM.view**.

Example:

```
; see terminal source code in
; ~~/demo/etc/terminal/t32term/t32term_memory.c
TERM.METHOD #1  BufferE  E:0x00000100 E:0x00000200
TERM.MODE   #1  VT100
TERM.view   #1

; Hint: the pre-commands WinExt and WinResist create a window that is
; (a) "external" to the TRACE32 PowerView main window and that is
; (b) "resistant" to the WinCLEAR command.
WinExt.WinResist.TERM.view #1
```

See also

- TERM
- TERM.CLEAR
- TERM.CLOSE
- TERM.CMDLINE
- TERM.GATE
- TERM.HARDCOPY
- TERM.HEAPINFO
- TERM.LocalEcho
- TERM.METHOD
- TERM.METHOD2
- TERM.Mode
- TERM.Out
- TERM.OutBREAK
- TERM.PIPE
- TERM.PipeREAD
- TERM.PipeWRITE
- TERM.PULSE
- TERM.READ
- TERM.RESet
- TERM.SCROLL
- TERM.SIZE
- TERM.STDIN
- TERM.TCP
- TERM.TELNET
- TERM.TRIGGER
- TERM.WRITE
- TERM.LINE()

▲ 'Release Information' in 'Legacy Release History'

Format:	**TERM.WRITE** [*<channel>*] *<file>* **TERM.WRITE** [*<address_out>*] *<file>* (deprecated)
<channel>:	**#***<number>*

The output sent from the target to the terminal emulation window is written to the specified file. The terminal emulation window must be opened before using this command. The **TERM.CLOSE** command closes the output file after or during transfer.

Example:

```
TERM.WRITE #1 term_out.lst
```

See also

■ TERM ■ TERM.view

▲ 'Release Information' in 'Legacy Release History'

TPIU

Trace Port Interface Unit (TPIU)

See also

- TPIU.CLEAR
- TPIU.IGNOREZEROS
- TPIU.NOFLUSH
- TPIU.PortClock
- TPIU.PortMode
- TPIU.PortSize
- TPIU.RefClock
- TPIU.Register
- TPIU.RESet
- TPIU.state
- TPIU.SWVPrescaler
- TPIU.SWVZEROS
- TPIU.SyncPeriod

▲ 'TPIU Functions' in 'General Function Reference'

Overview TPIU

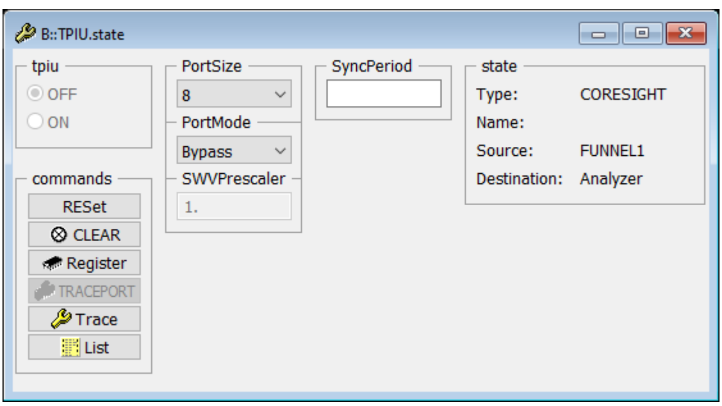
The **TPIU** command group enables you to configure and control the Trace Port Interface Unit (TPIU) of an ARM processor system or a non-ARM processor system using the ARM CoreSight trace. The TPIU is a trace sink which sends the trace data off-chip for capturing by a trace tool.

The TPIU typically outputs trace data via a parallel trace interface consisting of up to 32 trace data signals, a trace clock and optionally a trace control signal (indicating idle).

Some chip designs use these signals internally as an input to a High Speed Serial Trace Port (HSSTP) which converts the parallel data into a serial Xilinx-Aurora-based protocol for sending the serial bit stream off-chip on differential lanes.

A variant of the TPIU is the Serial Wire Output (SWO) which outputs trace data of the Serial Wire Viewer (SWV) via a single signal line. This output has a much lower bandwidth, is typically used for system trace, and is typically found on Cortex-M based designs. This variant does normally not use a dedicated trace connector. Instead it re-uses the TDO pin of a debug connector.

For TPIU setup, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **TPIU.state** window.



TPIU.CLEAR — Re-write the TPIU registers

Format:	**TPIU.CLEAR**

Re-writes the TPIU registers on the target with the settings displayed on the **TPIU.state** window.

See also

- TPIU
- TPIU.state

TPIU.IGNOREZEROS — Workaround for a special chip

Format:	**TPIU.IGNOREZEROS** [**ON** | **OFF**]

See also

- TPIU
- TPIU.state

TPIU.NOFLUSH — Workaround for a chip bug affecting TPIU flush

Format:	**TPIU.NOFLUSH** [**ON** | **OFF**]

Default: OFF.

Activates a workaround for a chip bug which caused serious issues when the trace tool caused a TPIU flush at the end of the trace recording.

See also

- TPIU
- TPIU.state

Format:	**TPIU.PortClock** *<frequency>* **ETM.PortClock** *<baud_rate>* (deprecated) **ITM.PortClock** *<frequency>* (deprecated)

Default: 1500Mbps

Informs the debugger about the HSSTP trace frequency to improve the accuracy of the timestamp calculation.

Example:

```
TPIU.PortClock 3125Mbps
TPIU.PortClock 3125M        ; M is the short form of Mbps
```

See also

■ TPIU ■ TPIU.state

Format:	**TPIU.PortMode** *<mode>* **ITM.PortMode** *<option>* (deprecated)
<mode>:	**Bypass** | **Wrapped** | **Continuous** | **NRZ**

Selects the operation mode of the TPIU.

Modes for Parallel Trace and HSSTP	
The TPIU can optionally output a trace control signal (TRACECTL) which indicates idle cycles of the trace port not worth to record. The TPIU formatter can be used to add the idle information to the trace packets. The formatter needs to be used in case of multiple trace sources to add the ID of the trace source.	
Bypass	TRACECTL pin is available, formatter is not used.
Wrapped	TRACECTL pin is available, formatter is used.
Continuous	TRACECTL pin is not available, formatter is used.

Modes for Serial Wire Output	
TRACE32 supports the UART/NRZ (NRZ = Non-Return-to-Zero) coding of the Serial Wire Output but not yet the Manchester coding. The bitrate of this asynchronous interface is derived by dividing the CPU frequency.	
NRZ	NRZ coding at CPU clock divided by *<divisor>* set up by: **TPIU.SWVPrescaler** *<divisor>* (default: 1)
NRZ/2 (deprecated) See example below.	NRZ coding at half of the CPU clock speed.
NRZ/3 (deprecated) See example below.	NRZ coding at a third of the CPU clock speed.
NRZ/4 (deprecated) See example below.	NRZ coding at a quarter of the CPU clock speed.

Example:

```
;(deprecated)
TPIU.PortMode NRZ/4

;please use these two commands instead of NRZ/<divisor>
TPIU.PortMode NRZ
TPIU.SWVPrescaler 4.
```

See also

- TPIU
- TPIU.state

Format:	**TPIU.PortSize** *<size>*
<size>:	**1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **12** | **16** | **18** | **20** | **24** | **32** | **8A** | **12A** | **16A** | **16E** | **1Lane** | **2Lane** | **3Lane** | **4Lane** | **5Lane** | **6Lane** | **SWV**

Specifies the interface type and port size of the TPIU.

Size in case of Parallel Trace:	
1, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **12**, **16**, **18**, **20**, **24**, **32**	Number of trace data signals. TRACE32 supports the listed sizes. A TPIU can support all sizes from 1 to 32 or only a few out of 1 to 32.
8A, **12A**, **16A**, **16E**	Variants of “8”, “12”, “16” in case of SoC from Texas Instruments. The selected size is the same, but additionally the Debug Resource Manager (DRM) gets configured which maps trace signals to output pins: • 8A: TRACEDATA[0:7] -> EMU[4:11] • 12A: TRACEDATA[0:11] -> EMU[4:15] • 16A: TRACEDATA[0:15] -> EMU[4:19] • 16E: TRACEDATA[0:1] -> EMU[0:1], TRACEDATA[2:15] -> EMU[4:17]

Size in case of HSSTP:	
1Lane, **2Lane**, **3Lane**, **4Lane**, **5Lane**, **6Lane**	Number of used differential lanes.

Size in case of Serial Wire Viewer (SWV) / Serial Wire Output (SWO):	
SWV	Selects SWV/SWO which uses only one signal.

See also

■ TPIU ■ TPIU.state ■ <trace>.PortSize

Format:	**TPIU.RefClock** [**/**<*option*>]
<*option*>:	**OFF** | **OSC** | **1/1** | **1/2** | **1/20** | **1/25** | **1/30** | **1/40** | **1/50**

Defines the reference clock frequency the serial preprocessor outputs to the target. Defaults depending on architecture:

- PowerPC: bit clock frequency
- TriCore and RH850: 100MHz
- ARM: bit clock frequency

OFF	TRACE32 does not send any reference clock to the target.
OSC	An asynchronous oscillator will be enabled. Its frequency is architecture dependent.
1/<*x*>	A synchronous clock source will be enabled. Its dividers generate a reference clock as a fraction of the bit clock (lane speed), e.g. 100MHz at 5Gbps with divider 1/50. Once a divider is selected, the reference clock will automatically change with the lane speed.

See also

■ TPIU ■ TPIU.state

TPIU.Register Display TPIU registers

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

Opens the **TPIU.Register** window, displaying the TPIU registers and the registers of other trace related modules.

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

See also

- TPIU
- TPIU.state

TPIU.RESet Reset TPIU settings

Format:	**TPIU.RESet**

Resets the settings in the **TPIU.state** window to their default values and re-configures the TPIU registers on the target.

See also

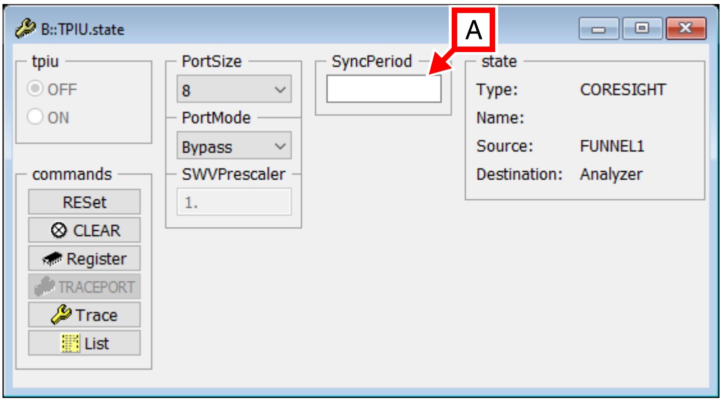
- TPIU
- TPIU.state

TPIU.state

Display TPIU configuration window

Format:	**TPIU.state**

Displays the **TPIU.state** configuration window.



A For descriptions of the commands in the **TPIU.state** window, please refer to the **TPIU.*** commands in this chapter. **Example**: For information about the **SyncPeriod** box, see **TPIU.SyncPeriod**.

Exceptions:

- The setting **TPIU.ON** and **TPIU.OFF** is read-only. The setting depends on the selected trace mode (Analyzer, Onchip, ...).
- The **Trace** button opens the main trace control window (**Trace.state**)
- The **List** button the main trace list window (**Trace.List**).

See also

- TPIU
- TPIU.CLEAR
- TPIU.IGNOREZEROS
- TPIU.NOFLUSH
- TPIU.PortClock
- TPIU.PortMode
- TPIU.PortSize
- TPIU.RefClock
- TPIU.Register
- TPIU.RESet
- TPIU.SWVPrescaler
- TPIU.SWVZEROS
- TPIU.SyncPeriod

TPIU.SWVPrescaler

Set up SWV prescaler

Format:	**TPIU.SWVPrescaler** *<divisor>*

Default: 1.

In case of **TPIU.PortMode** **NRZ**, the bitrate of the Serial Wire Viewer / Serial Wire Output is derived by dividing the CPU frequency. The command **TPIU.SWVPrescaler** sets up the divisor, which can range from 0x1 to 0x1000 (1. to 4096.).

Examples:

```
TPIU.PortMode NRZ
TPIU.SWVPrescaler 7.      ; NRZ coding at a 7th of the CPU clock
```

```
TPIU.PortMode NRZ
TPIU.SWVPrescaler 10.     ; NRZ coding at a 10th of the CPU clock
```

See also

- TPIU
- TPIU.state

TPIU.SWVZEROS — Workaround for a chip bug

Format:	**TPIU.SWVZEROS** [**ON** | **OFF**]

Default: OFF.

Activates a workaround for a chip bug affecting SWV/SWO data of a certain device.

See also

- TPIU
- TPIU.state

Format:	**TPIU.SyncPeriod** [*<packets>*]

Sets the number of regular TPIU packets which will be output to the trace stream between two synchronization packets.

What are synchronization packets? Synchronization packets are periodic starting points in the trace stream, which allow the recorded flow trace data to be decoded. The result can then be visualized in the *<trace>*.* windows of TRACE32, e.g. the **Trace.List** or the **Trace.PROfileChart.sYmbol** window. A visualization of the flow trace data is usually *not possible without* synchronization packets in the trace stream.

<packets>	If omitted, then the default number of regular packets between synchronization packets is chosen by the debugger or the chip.

In this example, the number of regular packets is 1024.

$\underbrace{\text{RP ... RP}}_{1024}$ **SP** $\underbrace{\text{RP ... RP}}_{1024}$ **SP** $\underbrace{\text{RP ... RP}}_{1024}$ **SP** RP ...

RP = regular packet
SP = synchronization packet

See also

- TPIU
- TPIU.state

TPU.BASE Base address

See command **TPU.BASE** in 'TPU Debugger' (tpu.pdf, page 5).

TPU.Break Break TPU

See command **TPU.Break** in 'TPU Debugger' (tpu.pdf, page 11).

TPU.Dump Memory display

See command **TPU.Dump** in 'TPU Debugger' (tpu.pdf, page 9).

TPU.Go Start TPU

See command **TPU.Go** in 'TPU Debugger' (tpu.pdf, page 12).

TPU.List View microcode

See command **TPU.List** in 'TPU Debugger' (tpu.pdf, page 11).

TPU.ListEntry Table display

See command **TPU.ListEntry** in 'TPU Debugger' (tpu.pdf, page 10).

TPU.Register.ALL Register operation mode

See command **TPU.Register.ALL** in 'TPU Debugger' (tpu.pdf, page 6).

TPU.Register.NEWSTEP New debugging mode

See command **TPU.Register.NEWSTEP** in 'TPU Debugger' (tpu.pdf, page 7).

TPU.Register.Set Register modification

See command **TPU.Register.Set** in 'TPU Debugger' (tpu.pdf, page 9).

TPU.Register.view Register display

See command **TPU.Register.view** in 'TPU Debugger' (tpu.pdf, page 8).

TPU.RESet Disable TPU debugger

See command **TPU.RESet** in 'TPU Debugger' (tpu.pdf, page 13).

TPU.SCAN Scannig TPU

See command **TPU.SCAN** in 'TPU Debugger' (tpu.pdf, page 5).

TPU.SELect Select TPU for debugging

See command **TPU.SELect** in 'TPU Debugger' (tpu.pdf, page 12).

TPU.Step Single step TPU

See command **TPU.Step** in 'TPU Debugger' (tpu.pdf, page 13).

TPU.view View TPU channels

See command **TPU.view** in 'TPU Debugger' (tpu.pdf, page 6).

Trace — Trace configuration and display

Format:	**Trace** | *<trace>*
<trace>:	*<trace_method>* | *<trace_source><trace_method>*

Trace	For information, see section **Overview Trace** in this command group description.
<trace>	For information, see subsection **About the Command Placeholder <trace>** in this command group description
<trace_method>	For information, see subsection **Replacing <trace> with a Trace Method - Examples** in this command group description.
<trace_source><trace_method>	For information, see subsection **Replacing <trace> with Trace Source and Trace Method - Examples** in this command group description.

NOTE: There is **NO** period between *<trace_source><trace_method>.*

This syntax convention is reserved for:
- Processing trace data from only one particular trace source, e.g. ITM.
- Processing trace data from more than one trace source, e.g. ITM and HTM.
- Processing trace data from very special trace sources.

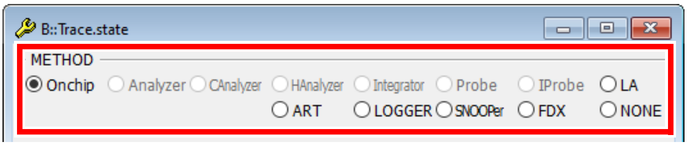
See also

- <trace>.CustomTrace
- <trace>.CustomTraceLoad
- <trace>.ListVar
- <trace>.MERGEFILE
- <trace>.PipeWRITE
- <trace>.SPY
- <trace>.TRIGGER
- <trace>.TSELect
- Analyzer.TOut
- Analyzer.TraceCLOCK
- Integrator.CSELect
- Integrator.TPreDelay
- Integrator.TSYNC
- Probe.TDelay

▲ 'Trace Functions' in 'General Function Reference'

Overview Trace

The command **Trace** is a general command for trace configuration and trace display. It is available for all kind of trace methods provided by TRACE32. The currently used trace method is displayed under **METHOD** in the **Trace.state** window.



For descriptions of the trace methods, see **Trace.METHOD**.

In this section:

- About the Command Placeholder <trace>
- What to know about the TRACE32 default settings for Trace.METHOD
- Types of Replacements for <trace>
- Replacing <trace> with a Trace Method - Examples
- Replacing <trace> with a Trace Evaluation - Example
- Replacing <trace> with RTS for Real-time Profiling - Example
- Replacing <trace> with Trace Source and Trace Method - Examples
- How to access the trace sources in TRACE32
- List of <trace> Command Groups consisting of <trace_source><trace_method>
- Related Trace Command Groups

About the Command Placeholder <trace>

In the TRACE32 manuals, *<trace>* is used as a placeholder for all types of trace commands. As the name *placeholder* implies, it cannot be used directly in the TRACE32 command line. As soon as you type *<trace>***.List** at the command line, you receive the error message “unknown command”. Consequently, you need to replace *<trace>* with the correct trace command before the command line accepts your input.

What to know about the TRACE32 default settings for Trace.METHOD

The easiest way to replace *<trace>* with a correct command is to type **Trace** at the command line. The meaning of **Trace**, e.g. in **Trace.List**, is then controlled by a sequence of TRACE32 default settings.

1. The TRACE32 *hardware module* connected to your target board determines the *trace method*. And this trace method will be used for recording the trace data. In the header of the **Trace.state** window, you can view the selected trace method.

   TRACE32 determines the default trace method as follows:

   - If the hardware module connected to your target board is a PowerTrace, then the **Analyzer** trace method becomes the default setting for the 1st TRACE32 PowerView GUI. For the other GUIs of an AMP configuration, the default setting is **Trace.METHOD NONE**.

   - If a hardware module other than a PowerTrace is connected to your target board, TRACE32 adjusts the trace method accordingly. For the other GUIs of an AMP configuration, the default setting is **Trace.METHOD NONE**.

   - If the chip has an onchip trace sink, then the **Onchip** trace method becomes the default setting for the 1st TRACE32 PowerView GUI.
     However, if the onchip trace recording is not yet operational, then the trace method is set to **NONE**. For the other GUIs of an AMP configuration, the default setting is **Trace.METHOD NONE**.

   - If the chip does *not* have an onchip trace sink, then the **ART** trace method becomes the default setting.

   - If TRACE32 runs in software-only mode as an instruction set simulator, then it is again the **Analyzer** trace method that becomes the default setting.

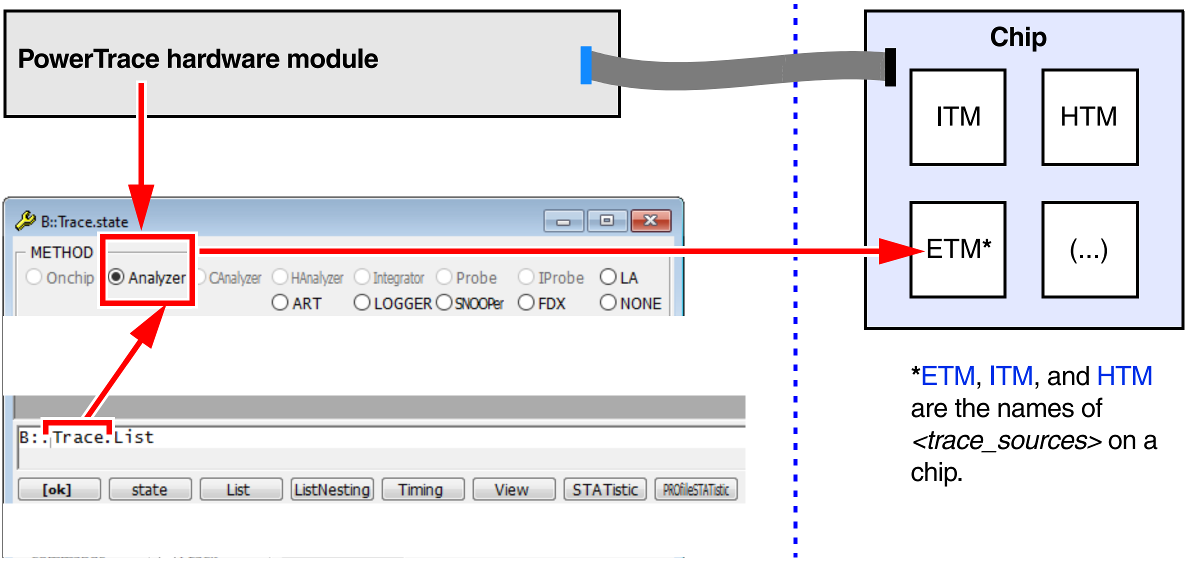
2. The **Analyzer** trace method is designed to look for a specific *trace source* that generates the program flow trace on the chip. For ARM chips, this trace source is called Embedded Trace Macrocell (ETM). For other chips, the trace source can be NEXUS or a proprietary trace block.

3. All **Trace** commands refer to the selected trace method.

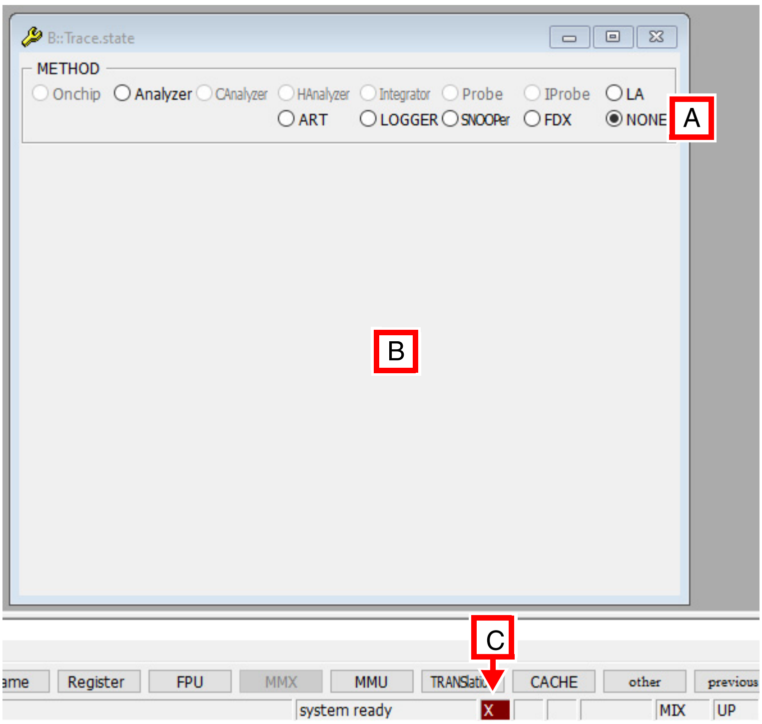
In the following first figure, the arrows illustrate the default settings used by the 1st TRACE32 PowerView GUI.

The second figure shows the effects of the default setting **Trace.METHOD NONE** on all other TRACE32 PowerView GUIs of an AMP configuration.

1st TRACE32 PowerView GUI:



All other TRACE32 PowerView GUIs: How does a TRACE32 PowerView GUI indicate that the **Trace.METHOD** is set to **NONE**?



A In the **Trace.state** window, **NONE** is selected as trace method.

B All other GUI controls in the **Trace.state** window are temporarily hidden. Their underlying **Trace.*** commands cannot be successfully executed at the TRACE32 command line either. The only command exceptions are **Trace.METHOD** and **Trace.state**.

C The state line displays a white X against a red background.

Types of Replacements for <trace>

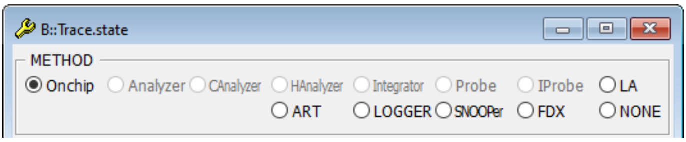
You can rely on the trace method that TRACE32 selects by default, but you can also select a trace method other than the default. As soon as you have selected the trace method *you want* in the **Trace.state** window, you can replace the placeholder *<trace>* with:

- **Trace** as explained in the previous section (Click here)
- The name of the trace method you have selected in the **Trace.state** window (Click here)
- Trace evaluation commands (Click here)
- **RTS**, the command for real-time profiling (Click here)
- Names of trace sources immediately followed by the name of the trace methods (Click here)

Replacing <trace> with a Trace Method - Examples

You can replace *<trace>* with the name of the selected trace method. The trace method commands are displayed in the **Trace.state** window:

- Onchip, Analyzer, CAnalyzer, , Integrator, Probe, IProbe, LA, ART, LOGGER, SNOOPer, FDX,



Example 1 for the trace method SNOOPer:

```
Trace.state                 ;select the trace method SNOOPer for recording
Trace.METHOD SNOOPer        ;trace data.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

Trace.List                  ;display the trace data recorded with SNOOPer
                            ;as a trace listing.
SNOOPer.List                ;this is the equivalent and explicit command.
```

Example 2 for the trace method LOGGER:

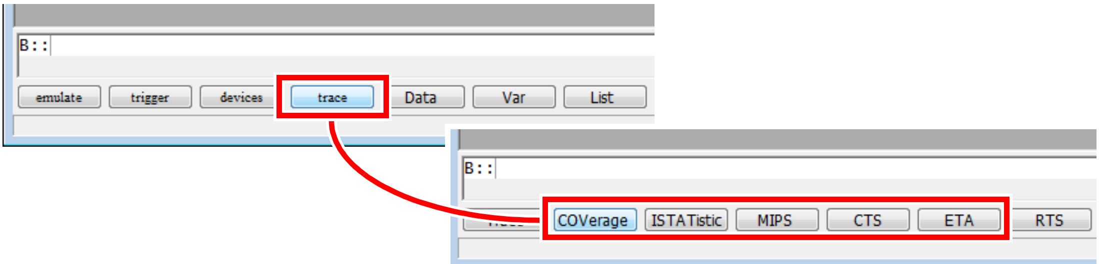
```
Trace.state                 ;select the trace method LOGGER for recording
Trace.METHOD LOGGER         ;trace data.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

Trace.List                  ;display the trace data recorded with LOGGER
                            ;as a trace listing.
LOGGER.List                 ;this is the equivalent and explicit command.
```

Replacing <trace> with a Trace Evaluation - Example

For trace evaluations, you can replace *<trace>* with a trace evaluation command; the name of the trace method is omitted.

The trace evaluation commands are accessible via the TRACE32 softkey bar:

- COVerage, ISTATistic, MIPS, CTS, ETA, BMC

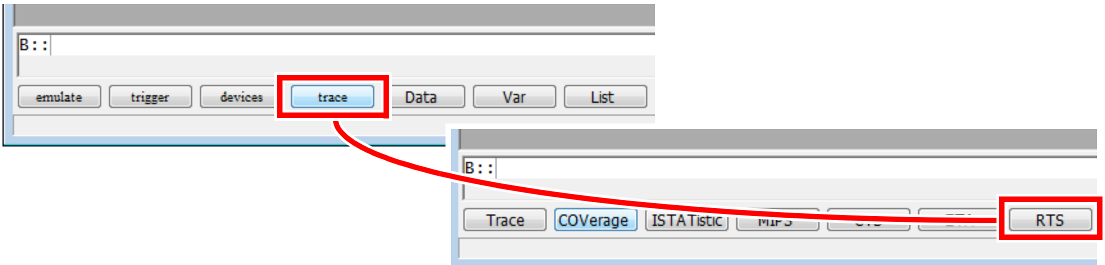


Example:

```
Trace.state                ;select the trace method Analyzer for recording
Trace.METHOD Analyzer      ;trace data.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

COVerage.List              ;<trace> is just replaced with the trace
                           ;evaluation command, since the trace method
                           ;Analyzer is defined above anyway.
```

For real-time profiling, you can replace the placeholder *<trace>* with **RTS**.

The **RTS** command is accessible via the TRACE32 softkey bar:



Example:

```
Trace.state                ;select the trace method Analyzer for
Trace.METHOD Analyzer      ;recording trace data.
;<configuration>

RTS.state
RTS.ON
;<configuration>

Go                         ;processes the trace data being recorded from
                           ;the target while the target is running.

ISTATistic.ListModule      ;ISTATistic windows display real-time
                           ;trace data as long as RTS is switched ON
                           ;(RTS.ON)
```

As stated in the blue Format table, the placeholder *<trace>* can be replaced with trace commands consisting of *<trace_source>* and *<trace_method>*.

Rule	Example
<trace_source> **+** *<trace_method>* → *<trace>*	HTM **+** Analyzer → HTMAnalyzer

These *<trace>* command groups are accessible via the TRACE32 softkey bar and include for example:

- CoreSightTrace, ETMTrace, ETMAnalyzer, STMAnalyzer, CoreSightCAnalyzer, …
- For an overview, see **List of <trace> Command Groups consisting of <trace_source><trace_method>**.

Using these *<trace>* command groups, you can display trace data recorded from one or more trace sources.

Example for displaying trace data from one trace source: This script assumes that the CoreSight components of the chip output their trace data to the same trace sink.

```
Trace.state                 ;select the trace method Analyzer for recording
Trace.METHOD Analyzer       ;trace data.
;<configuration>

ETM.ON                      ;switch on the trace source from which you want
;<configuration>            ;to record trace data, here the ETM.

;trace data is recorded using the commands Go, WAIT, Break

Trace.List                  ;display the ETM trace data recorded with the
                            ;trace method Analyzer as a trace listing.
Analyzer.List               ;this is the equivalent and explicit command.
```

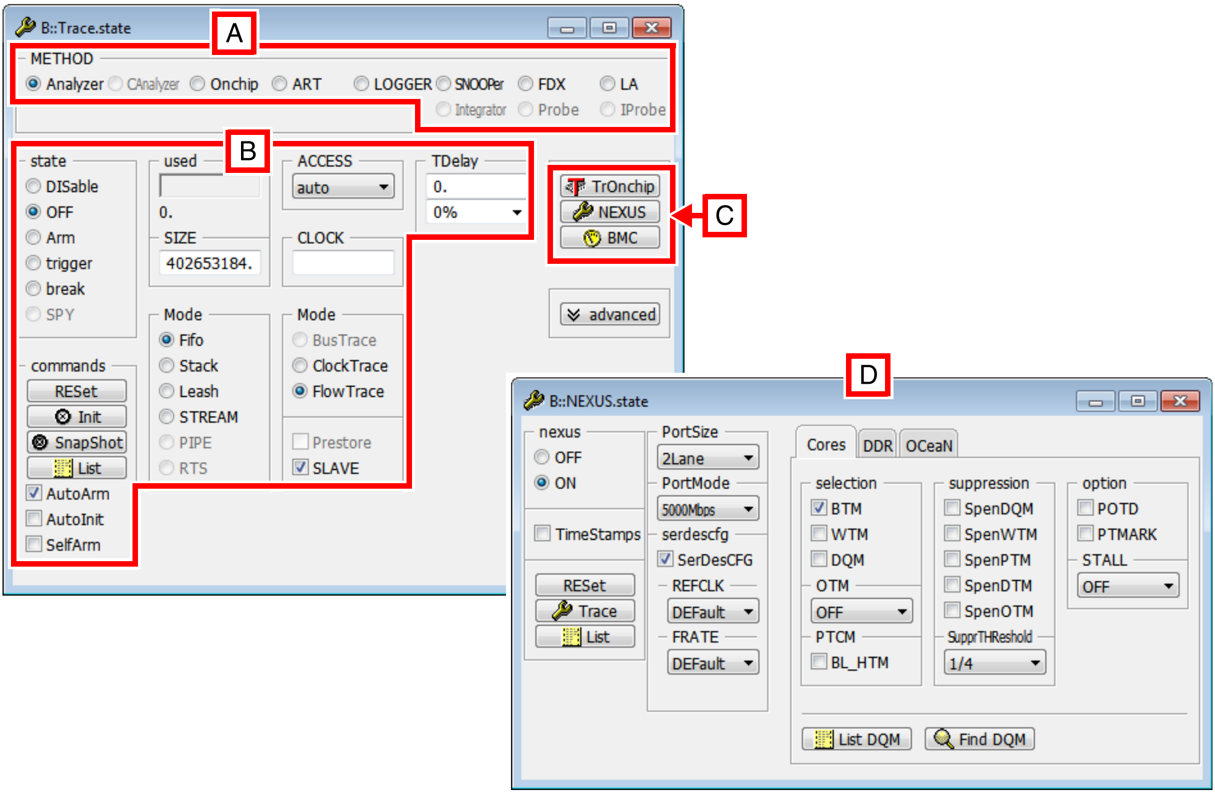
Example for displaying trace data from two trace sources: This script assumes that the CoreSight components of the chip output their trace data to the same trace sink.

```
Trace.state               ;select the trace method Analyzer for recording
Trace.METHOD Analyzer     ;trace data.
;<configuration>

ETM.ON                    ;switch the 1st trace source ETM on.
;<configuration>

HTM.ON                    ;switch the 2nd trace source HTM on.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

Trace.List                ;display the ETM trace data.
HTMTrace.List             ;display the HTM trace data.
```

As you have seen in the previous sections, the **Trace.state** window is the starting point for configuring a trace recording and recording the trace data: It provides an overview of the trace methods [**A**], and it dynamically adjusts to the trace method you have selected [**B**].

In addition, the **Trace.state** window displays buttons for each trace source found on the chip [**C**]. Clicking a button lets you access a *<trace_source>***.state** window, where you can configure the selected trace source directly in TRACE32.

Example: TRACE32 has found has three trace sources on a QorIQ chip, including a NEXUS trace source [**C**]. Click the **NEXUS** button to open the **NEXUS.state** window [**D**]. You can now configure the NEXUS trace source.



List of <trace> Command Groups consisting of <trace_source><trace_method>

Trace methods can be combined with a trace source are:

- **Trace**: method-independent analysis
- **Analyzer**: analyze information recorded by TRACE32 PowerTrace
- **CAnalyzer**: analyze information recorded by Compact Analyzer (e.g. CombiProbe, μTrace (MicroTrace))
- **HAnalyzer**: analyze information recorded by the Host Analyzer
- **Onchip** / **Onchip2**: analyze information recorded in target onchip memory / second onchip memory
- **LA**: analyze information recorded from binary source

Not all trace sources can be combined with these trace methods.The table below shows all supported combinations.

<trace_source>	**Supported <trace_source><trace_method> commands**
AET Advanced Triggering Trace (C5000, C6000, C7000)	**AETAnalyzer**
CoreSight	**CoreSightTrace** **CoreSightAnalyzer** **CoreSightCAnalyzer** **CoreSightHAnalyzer** **CoreSightOnchip** **CoreSightOnchip2** **CoreSightLA**
CMN Coherent Mesh Network trace (Arm/Cortex)	**CMNTrace** **CMNAnalyzer** **CMNCAnalyzer** **CMNHAnalyzer** **CMNOnchip** **CMNOnchip2** **CMNLA**
DDR NEXUS DDR controller debug trace (PowerPC QorIQ) See **“QorIQ Debugger and NEXUS Trace”** (debugger_ppcqoriq.pdf)	**DDRTrace** **DDRAnalyzer** **DDROnchip** **DDRLA**
DQM NEXUS Data Acquisition trace messages (PowerPC QorIQ) See **“QorIQ Debugger and NEXUS Trace”** (debugger_ppcqoriq.pdf)	**DQMTrace** **DQMAnalyzer** **DQMOnchip** **DQMLA**

DTM Data Trace Module (Arm/Cortex, ARC)	**DTMAnalyzer** **DTMCAnalyzer** **DTMHAnalyzer** **DTMLA** **DTMOnchip** **DTMTrace**
ELA Embedded Logic Analyzer (Arm/Cortex)	**ELATrace** **ELAAnalyzer** **ELACAnalyzer** **ELAHAnalyzer** **ELAOnchip** **ELAOnchip2** **ELALA**
ETM Embedded Trace Macrocell (Arm/Cortex)	**ETMTrace** **ETMAnalyzer** **ETMCAnalyzer** **ETMHAnalyzer** **ETMOnchip** **ETMLA**
ETMD ETM Data Stream (Arm/Cortex)	**ETMDTrace** **ETMDAnalyzer** **ETMDCAnalyzer** **ETMDHAnalyzer** **ETMDOnchip** **ETMDLA**
ETMX (Arm/Cortex)	**ETMXTrace** **ETMXAnalyzer** **ETMXCAnalyzer** **ETMXHAnalyzer** **ETMXOnchip** **ETMXLA**
Funnel (Arm/Cortex)	**FunnelAnalyzer** **FunnelOnchip**
HTM CoreSight HTM (AHB Trace Macrocell)	**HTMTrace** **HTMAnalyzer** **HTMCAnalyzer** **HTMHAnalyzer** **HTMOnchip** **HTMLA**
ITH Intel Trace Hub	**ITHTrace**
ITM (Arm/Cortex)	**ITMTrace** **ITMAnalyzer** **ITMCAnalyzer** **ITMHAnalyzer** **ITMOnchip** **ITMLA**

MCDSBase Non-optimized MCDS trace (TriCore)	**MCDSBaseAnalyzer** **MCDSBaseCAnalyzer** **MCDSBaseOnchip** **MCDSBaseLA**
MCDSDCA MCDS trace processing with data cycle assignment (TriCore)	**MCDSDCAAnalyzer** **MCDSDCACAnalyzer** **MCDSDCAOnchip** **MCDSDCALA**
MCDSDDTU MCDS trace processing with DDTU reordering (TriCore)	**MCDSDDTUAnalyzer** **MCDSDDTUCAnalyzer** **MCDSDDTUOnchip** **MCDSDDTULA**
NPKReorder Northpeak Reorder (Intel x86)	**NPKReorderTrace** **NPKReorderAnalyzer** **NPKReorderCAnalyzer** **NPKReorderHAnalayzer** **NPKReorderLA**
OCeaN On Chip Network debug trace (PowerPC QorIQ) See **"QorIQ Debugger and NEXUS Trace"** (debugger_ppcqoriq.pdf)	**OCeaNTrace** **OCeaNAnalyzer** **OCeaNOnchip** **OCeaNLA**
RTP RAM Trace Port (Arm/Cortex) See **"RAM Trace Port"** (trace_rtp.pdf)	**RTPAnalyzer**
SFT Software Trace (RH850)	**SFTTrace** **SFTAnalyzer** **SFTOnchip**
STM / STM2 System Trace	**STMTrace / STM2Trace** **STMAnalyzer / STM2Analyzer** **STMCAnalyzer / STM2CAnalyzer** **STMHAnalyzer** **STMOnchip / STM2Onchip** **STMOnchip2 / STM2Onchip2** **STMLA / STM2LA**
TSI / TSI2 (CEVA-X)	**TSITrace / TSI2Trace** **TSIAnalyzer / TSI2Analyzer** **TSICAnalyzer / TSI2CAnalyzer** **TSIHAnalyzer / TSI2HAnalyzer** **TSIOnchip / TSI2Onchip** **TSILA / TSI2LA**

UltraSOC	**UltraSOCTrace** **UltraSOCHAnalayzer** **UltraSOCLA**
XGate (MCS12)	**XGateOnchip**
XTI	**XTICAnalyzer**

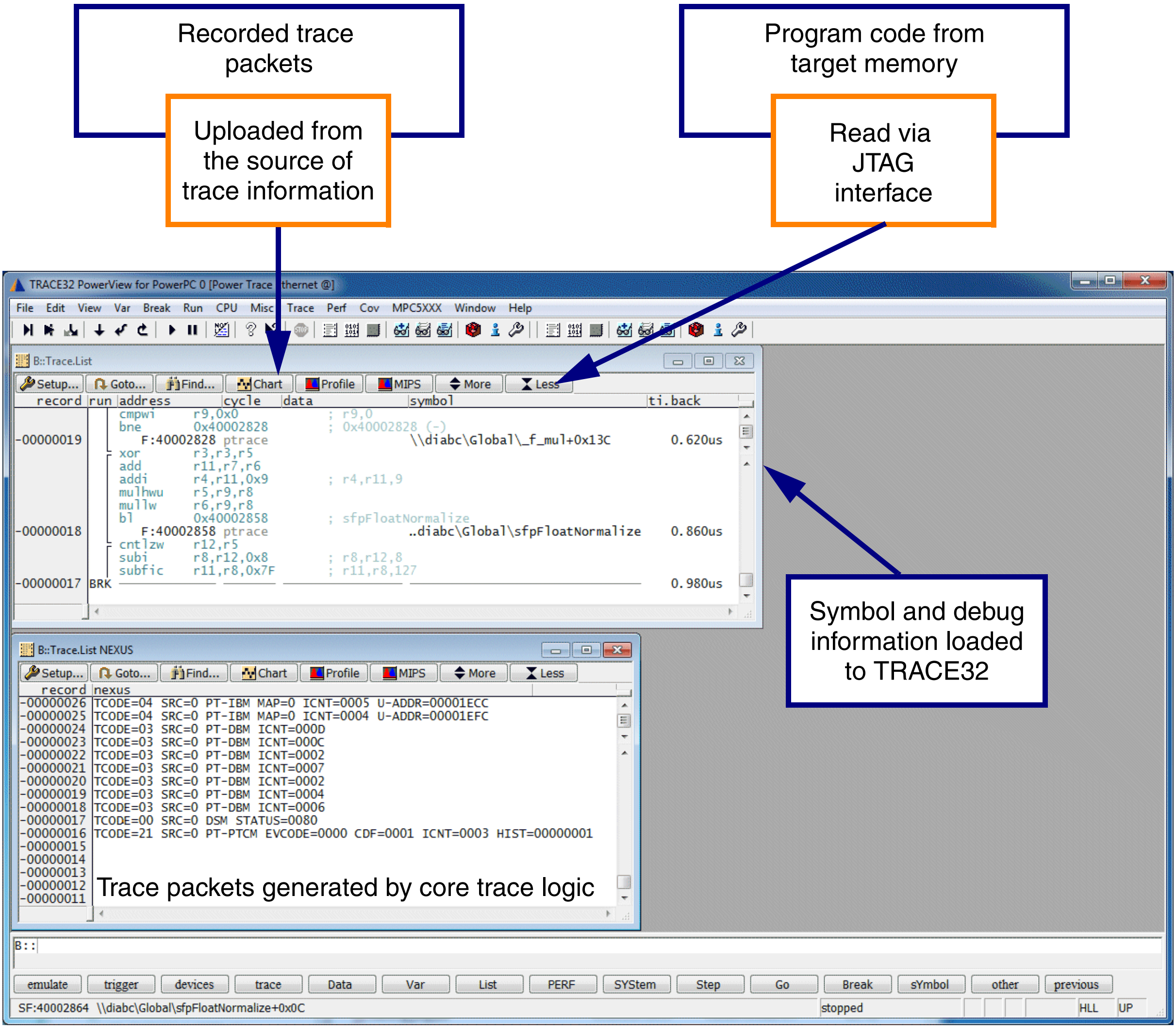
Related Trace Command Groups

CMITrace	Clock management instrumentation trace by Texas Instruments on OMAP4.
CPTracerTrace	Analyzes and displays CPT trace data.
OCPTrace	OpenCoreProtocol WatchPoint trace by Texas Instruments on OMAP4 and OMAP5.
PMITrace	Power management instrumentation trace by Texas Instruments on OMAP4.
PrintfTrace	Displays and analyzes software messages.
SLTrace	Allows to trace and analyze **SYStem.LOG** events.
StatColTrace	Statistics collector trace by Texas Instruments on OMAP4 and OMAP5.
SystemTrace	Displays and analyzes trace information generated by various trace sources.

Format:	*<trace>***.ACCESS** *<path>* **COVerage.ACCESS** [**auto** | **VM** | **DualPort**] (deprecated)
<path>:	**auto** | **AutoVM** | **CPU** | **DualPort** | **VM** | **OVS** | **DENIED**

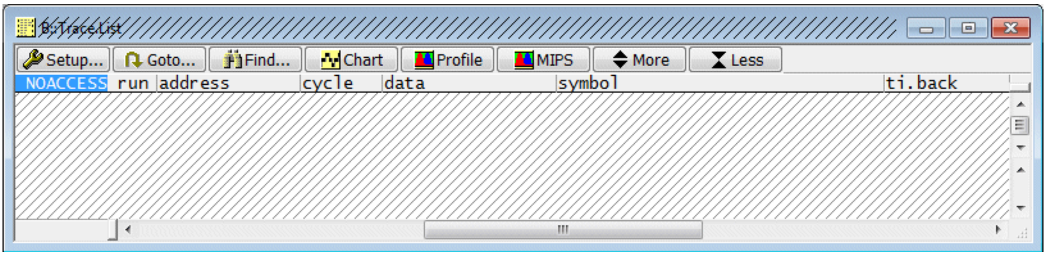
The core trace generation logic on the processor/chip generates trace packets to indicate the instruction execution sequence (program flow). TRACE32 merges the following sources of information in order to provide an intuitive display of the instruction execution sequence (flow trace).

- The trace packets recorded.
- The program code from the target memory (usually read via the JTAG interface).
- The symbol and debug information already loaded to TRACE32.


Recorded trace packets
Uploaded from the source of trace information
Program code from target memory
Read via JTAG interface
Symbol and debug information loaded to TRACE32
Trace packets generated by core trace logic
B::Trace.List
B::Trace.List NEXUS

1. **Trace information should be analyzed while the program execution is running and the debugger has no run-time access to the target memory to read the program code.**

NOACCESS in a trace display window indicates that the debugger can not read the target memory.



You can overcome this problem by loading the program code to the TRACE32 virtual memory.

```
; load the program code additional to the TRACE32 virtual memory
; whenever you load it to the target memory
Data.LOAD.Elf diabc.x /PlusVM
```

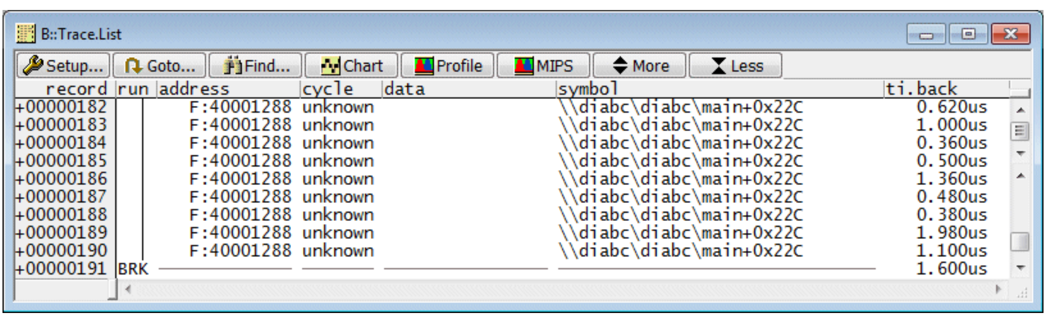
2. **Reading the target via JTAG is very slow therefore all trace display and analysis windows are slow.**

You can overcome this problem by loading the program code to the TRACE32 virtual memory and by specifying **Trace.ACCESS AutoVM**.

```
; load the program code additional to the TRACE32 virtual memory
; whenever you load it to the target memory
Data.LOAD.Elf diabc.x /PlusVM

; advise TRACE32 to read the target code from the virtual memory

; if no code is loaded to the virtual memory for a program address
; TRACE32 will read the code by using the best practice procedure
Trace.ACCESS AutoVM
```

3. **Trace information should be inspected, but there is no program code available.**

You can overcome this problem by specifying **Trace.ACCESS Denied** to advise TRACE32 not to merge program code information. The **Trace.List** window will list the available program addresses and mark all cycles as unknown.



Recommended access paths:

auto	TRACE32 uses its own best practice procedure to read the program code. (**Note**: For the ARM architecture this mode is usually *not* using the DualPort access.)
AutoVM	If the program code for a program address is available via the TRACE32 virtual memory it is read from there. Otherwise the best practice procedure is used.
VM	The program code is always read from the TRACE32 virtual memory.
Denied	No program code information is read.

Rarely used access paths:

OVS	Code overlays are handled by the best practice procedure. If the best practice procedure does not deliver correct results, you can advise TRACE32 to read the program code by using the overlay table.
CPU	Advise TRACE32 to read the code via the CPU/core.
DualPort	Advise TRACE32 to read the code via the run-time access to the target memory.

<trace>.Arm Arm the trace

Format:	*<trace>*.**Arm**

The trace memory and if available the trigger unit are prepared for recording and triggering. It is not possible to read the trace contents while the trace is in **Arm** state.

For most trace methods it is possible to **AutoArm** (**<trace>.AutoArm**) the trace. That means:

- Recording and triggering are prepared whenever the program execution is started.
- Recording and triggering are stopped whenever the program execution is stopped.

This is the default setting.

It is also possible to manually switch off the trace (**<trace>.OFF**) to read the trace contents and arm it again afterwards.

See also

■ <trace>.AutoArm ■ <trace>.AutoStart ■ <trace>.Init ■ IProbe.state
■ RunTime ■ RunTime.state

▲ 'Release Information' in 'Legacy Release History'

<trace>.AutoArm

Arm automatically

Format: *<trace>*.**AutoArm** [**ON** | **OFF**]

Default: *<trace>*.**AutoArm ON**.

- Recording and if available triggering is prepared whenever the program execution is started.
- Recording and if available triggering is stopped whenever the program execution is stopped.

See also

- <trace>.Arm
- IProbe.state
- RunTime
- RunTime.state
- SNOOPer.STATE()

<trace>.AutoFocus

Calibrate AUTOFOCUS preprocessor

Format: *<trace>*.**AutoFocus** [*<address_range>*] [**/***<option>*]

<option>: **Accumulate**
KEEP
ALTERNATE
NoTHreshold

The command **Trace.AutoFocus** configures an AutoFocus preprocessor for an error-free sampling on a high-speed trace port.

For preprocessors without AUTOFOCUS technology, but adjustable reference voltage, this command will modify the reference voltage (see **Trace.THreshold**) and try to find a value were the trace capture is free of errors. This might take anywhere from a few up to 30 s.

If available the test pattern generator of the trace port is used to generate the trace data for the auto-configuration. Otherwise a test program is loaded and started by TRACE32.

If a test program is used, TRACE32 attempts to load the test program to the memory addressed by the PC or the stack pointer. It is also possible to define an *<address_range>* for the test program.

```
Trace.AutoFocus                         ; start the auto-configuration

Trace.AutoFocus 0x24000000++0xfff       ; start auto-configuration, load
                                        ; the test program to address
                                        ; 0x24000000
```

If TRACE32 is unable to load the test program the following error message is displayed:
“Don‘t know where to execute the test code”.

By default the original RAM contents is restored after the auto-configuration and the trace contents is deleted.

Accumulate	If the application program varies the CPU clock frequency, this affects also the trace port and the auto-configuration. In such a case it is recommended to overlay the auto-configurations for all relevant CPU clock frequencies by using the option **/Accumulate**.
KEEP	When the auto-configuration is completed, the test pattern generator/test program is started once again to test the correctness of the trace recording. After this test the trace is cleared and an eventually loaded test program is removed from the target RAM. With the option **/KEEP** the test trace is not cleared and can be viewed with the **Trace.List** command. If a test program was loaded by TRACE32 it also remains in the target RAM.
ALTERNATE	If the trace port provides a test pattern generator, it is always used for the auto-configuration. The option **/ALTERNATE** forces TRACE32 to use its own test program. This is recommended e.g. if a CoreSight test pattern generator is not stimulating the TRACECLT signal.
NoTHreshold	Do not calibrate the **Trace.THreshold** reference voltage.

The option **/Accumulate** allows to overlay several auto-configurations. It is recommended to proceed as follows:

1. Execute the command **Trace.AutoFocus** at the highest CPU clock frequency.
2. Reduce the CPU clock frequency and execute the command **Trace.AutoFocus /Accumulate**.

   If a preprocessor with AUTOFOCUS technology is used, the clock and data delays are adjusted, while the termination voltage, the clock reference voltage and the data reference voltage remain unchanged.
3. Repeat step 2 for all relevant frequencies.

```
Trace.AutoFocus                  ; Execute the command for the
                                 ; highest CPU clock

Trace.AutoFocus /Accumulate      ; Re-execute the command for the
                                 ; next lower CPU clock

;…

Trace.AutoFocus /Accumulate      ; Re-execute the command for the
                                 ; lowest relevant CPU clock
```

A failure in the **Trace.AutoFocus** command results in a stop of a PRACTICE script. The following workaround can be used to avoid this behavior:

```
; go to the label error_autofocus: if an error occurred in the script

ON ERROR GOTO error_autofocus

Trace.AutoFocus

; go to the label end: if an error occurred in the script

ON ERROR GOTO end

…

end:

ENDDO

error_autofocus:

PRINT %ERROR "Trace.AutoFocus failed. Script is aborted"

ENDDO
```

NOTE:	The NEXUS AutoFocus adapter does not support this feature.

The **Trace.AutoFocus** command causes the preprocessor with AUTOFOCUS technology to configure itself. The auto-configuration searches for the best set of reference voltages and assures optimal sampling of the information broadcast by the trace port. The higher the trace port data rate, the more effort is put in the hardware configuration. For trace port data rates higher 200 Mbit/s the command may need up to 7 s for completion.

In contrast to **Trace.TestFocus,** the command **Trace.AutoFocus** does both the hardware configuration as well as a trace port test.

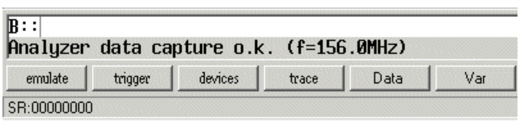
For preprocessors with AUTOFOCUS technology the hardware auto-configuration includes:

- Automatic setup of proper termination voltage to assure signal integrity.
- Automatic setup of clock reference voltage resulting in a stable clock with 50/50 duty cycle.
- Automatic setup of data reference voltage resulting in broad data eyes.
- Automatic setup of clock and data delays resulting in optimal sampling for each data channel.

The complete auto-configuration executes the following steps:

1. If available the trace port's test pattern generator is started. Otherwise a test program (maximum size 4 kB) is loaded by TRACE32 to the target RAM and started.
2. A hardware auto-configuration as described above is executed. When the optimal hardware configuration is found the test pattern generator/test program is stopped and the trace data is discarded. After executing the hardware auto-configuration the data eyes and optimal sampling points are known to the TRACE32 software and can be viewed by the user with the **Trace.ShowFocus** command.
3. The test pattern generator/test program is started once again and the program and data flow is recorded to the trace buffer to allow TRACE32 to verify the correctness of the trace recording.

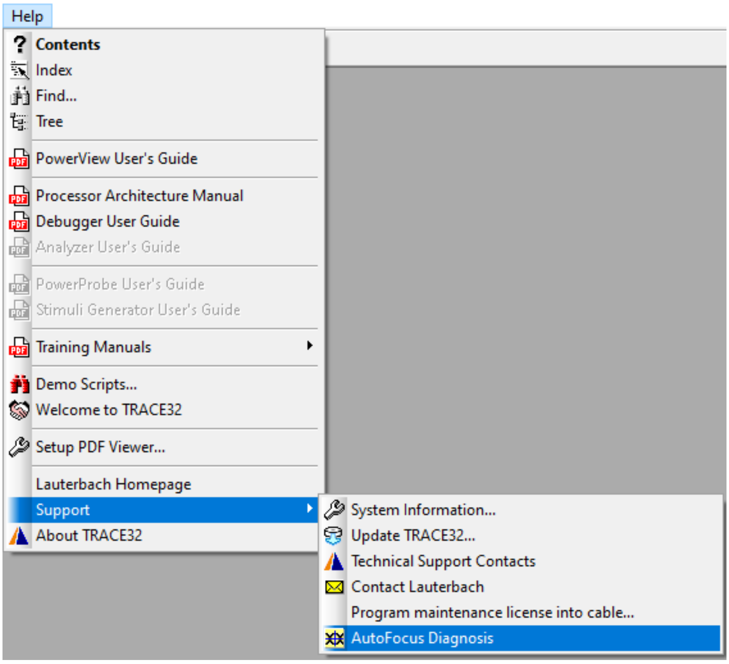
If the self calibration was successful, the following message is displayed in the message line (f=*<trace_port_frequency>*):



NOTE: The trace port frequency does not necessarily equal the CPU clock frequency. E. g. for the ARM-ETM:
- An ETMv1 or ETMv2 operating at HalfRate results in an ETM clock frequency that is half the CPU clock frequency
- An ETMv3 operating with PortMode 1/2 results in an ETM frequency that is a quarter of the CPU clock frequency.

The result of the **Trace.AutoFocus** command can be displayed with the **Trace.ShowFocus** command. If the user wants to verify that the current hardware configuration is complying with the current requirements (e.g. after a frequency change) without wanting to change this configuration, the **Trace.TestFocus** command can be used.

If the auto-configuration fails and you need technical support, please use the **AutoFocus Diagnosis** menu to prepare all relevant information for the support person.



See also

- ❏ AUTOFOCUS.FREQUENCY()
- ❏ AUTOFOCUS.OK()
- ▲ 'Release Information' in 'Legacy Release History'

<trace>.AutoInit — Automatic initialization

Format:	*<trace>*.**AutoInit** [**ON** | **OFF**]

The **<trace>.Init** command will be executed automatically, when the user program is started (or stepped through). This causes that

- Trace memory contents is erased and previous records are no longer visible.
- The trigger unit is set to its initial state.
- All used counters are initialized and all used flags are set to OFF.

In combination with the command **<trace>.SelfArm** the trace is able to generate continuous recording and display like a trace snapshot.

See also

■ <trace>.Init ■ IProbe.state ■ RunTime ■ RunTime.state
❏ SNOOPer.STATE()
▲ 'Release Information' in 'Legacy Release History'

<trace>.AutoStart — Automatic start

Format:	*<trace>*.**AutoStart** [**ON** | **OFF**]

The **<trace>.AutoStart** command will execute the **<trace>.Init** automatically, when a specified break event is encountered and a user program is re-started with the command **Go** or Step.

See also

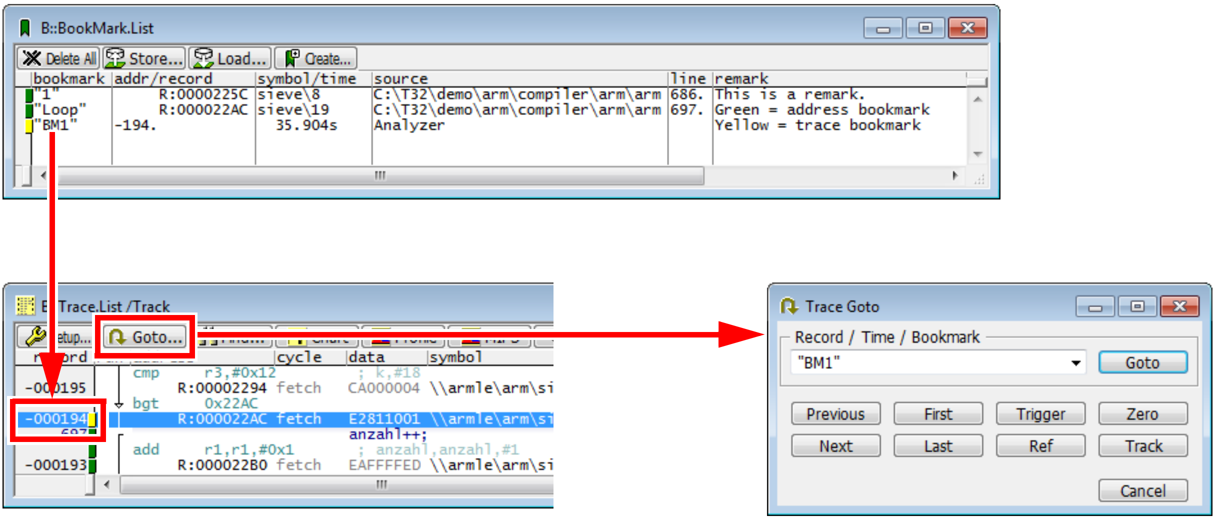
■ <trace>.Arm

<trace>.BookMark — Set a bookmark in trace listing

Format:	*<trace>*.**BookMark** *<string>* [*<time>* | *<value>*] [**/FILE**]

Sets a trace bookmark in the trace listing. A small yellow rectangle next to the record number indicates a trace bookmark.

The **BookMark.List** window provides an overview of all trace bookmarks. Clicking a yellow trace bookmark takes you to the location of that trace bookmark. Additionally, you can use the **Goto** button in a **<trace>.List** window to jump to a bookmarked trace record.



<string>	User-defined bookmark name. An auto-incremented bookmark name can be generated via the TRACE32 command line if a comma is entered instead of a user-defined name.
<time>	Creates a trace bookmark at a timestamp that is based on zero time. See example 2 below.
<value>	Creates a trace bookmark at the specified record number, e.g. -120000.

Example 1:

```
;create a trace bookmark named "BM2" for the trace record -120000.
Trace.BookMark "BM2" -120000.

Trace.List DEFault /Track    ;list the trace contents
BookMark.List                ;display all bookmarks in a list
```

Example 2 shows how to create a bookmark 0.300ms after the zero-time reference point. The optional steps are included in this example to let you view on screen what happens behind the scenes.

```
;optional step: In the trace listing, the TIme.ZERO column is displayed
;as the first column, followed by the DEFault columns
Trace.List TIme.ZERO DEFault /Track

;optional step: go to the first trace record, i.e. the record with the
;lowest record number
Trace.GOTO Trace.FIRST()

;set the zero-time reference point to the first trace record
ZERO.offset Trace.RECORD.TIME(Trace.FIRST())

Trace.BookMark "BM3"  0.300ms  ;create a bookmark 0.300ms after the
                               ;zero-time reference point

Trace.GOTO "BM3"               ;optional step: got to the new bookmark

BookMark.List                  ;optional step: display all bookmarks
```

See also

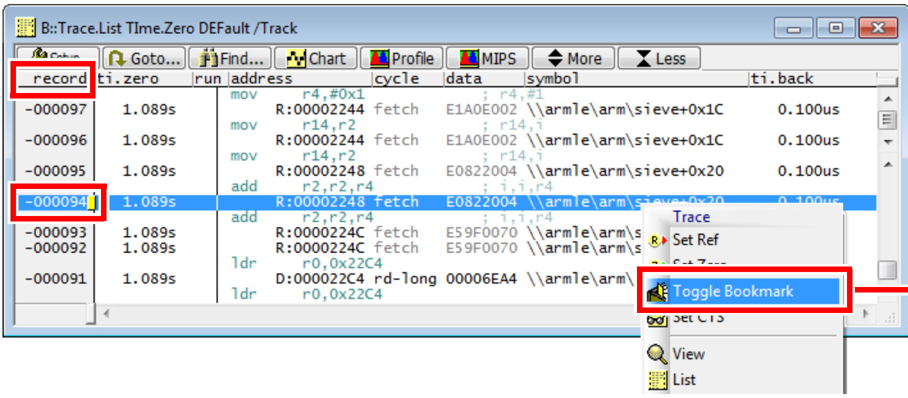
- <trace>.List
- <trace>.BookMarkToggle
- <trace>.GOTO
- BookMark
- BookMark.Create
- BookMark.EditRemark
- BookMark.List
- IProbe.state
- RunTime
- RunTime.state

▲ 'BookMark' in 'General Commands Reference Guide B'

<trace>.BookMarkToggle

Format:	*<trace>*.**BookMarkToggle** *<string>* [*<time>* | *<value>*] [**/FILE**]

Switches a single trace bookmark on or off. TRACE32 executes the same command when you right-click in a **<trace>.List** window, and then choose **Toggle Bookmark**. The resulting bookmark names are auto-incremented 1, 2, 3, etc. User-defined bookmark names can be created via the command line.
A small yellow rectangle next to the record number indicates a trace bookmark.



Trace bookmark for record -94.

<string>	User-defined bookmark name. An auto-incremented bookmark name can be generated via the command line if a comma is entered instead of a user-defined name.
<time>	Creates a trace bookmark at a timestamp that is based on zero time.
<value>	Creates a trace bookmark at the specified record number, e.g. -120000.

Example:

```
Trace.List TIme.Zero DEFault /Track ;list the trace contents

;let's toggle two trace bookmarks with user-defined names
Trace.BookMarkToggle "TStart" -Trace.Records() ;bookmark at first record
Trace.BookMarkToggle "TEnd"   -1.              ;bookmark at last record

BookMark.List        ;display all bookmarks in a list
```

See also

- ■ <trace>.BookMark ■ BookMark ■ BookMark.List ■ BookMark.Toggle
- ▲ 'BookMark' in 'General Commands Reference Guide B'

The **<trace>.Chart** command group allows to display the analyzed trace information graphically. Examples are:

- Function run-time (**Trace.Chart.Func**)
- Time chart (**Trace.Chart.sYmbol**)
- Task run-time (**Trace.Chart.TASK**)
- Variable contents (**Trace.Chart.VarState**)

Parameters

This section describes the optional *<trace_area>* parameters of the **<trace>.Chart** command group.

<record_range>	Defines which part of the trace buffer is displayed. See example.
<record>	Defines which trace record is centered on the x-axis when the window is opened. Records at the beginning or end of the x-axis are not centered. See example.
<time>	Defines which timestamp is centered on the x-axis when the window is opened. Timestamps at the beginning or end of the x-axis are not centered. **NOTE**: Only zero-time timestamps can be used as *<time>* parameters. You can display the zero-time timestamps in a **Trace** window by adding the **TimeZero** option to **Trace.Chart.*** or by adding the **TIme.Zero** column to **Trace.List**. See examples.
<time_range>	Defines which timestamp is displayed on left of the x-axis when the window is opened. **NOTE**: Only zero-time timestamps can be used as *<time_range>* parameters. You can display the zero-time timestamps in a **Trace** window by adding the **TimeZero** option to **Trace.Chart.*** or by adding the **TIme.Zero** column to **Trace.List**. See example.

<timescale>	The *<timescale>* parameter defines the display scaling as time per character. It is useful for printing operations and allows to print out any timing chart in a fixed scale on multiple pages. • See example. • For the units of measurement, see **"Parameter Types"** in PowerView User's Guide, page 43 (ide_user.pdf). Rule of thumb: The smaller the *<timescale>* value, the higher the resolution and the wider the chart in the data area of a **<trace>.Chart.*** window.
<trace_bookmark>	Defines which bookmark position is centered on the x-axis when the window is opened. Bookmark positions at the beginning or end of the x-axis are not centered. **NOTE**: You can only use the names of trace bookmarks, which are created with the **<trace>.BookMark** command. See example.

Options

This section describes the options of the **<trace>.Chart** command group. Not all options are supported by all **<trace>.Chart** commands.

Track	The cursor in the **<trace>.Chart** window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed. The zoom factor of the **<trace>.Chart** window is retained, even if the trace content changes.
ZoomTrack	Same as option **Track**. If the tracking in performed with another **<trace>.Chart** window the same zoom factor is used.

Sort [*<sort_visible>*] [*<sort_core>*] [*<sort>*]	Specify sorting criterion for analyzed items. For almost all commands the analyzed items are displayed in the order they are recorded by default. Details on the sorting criterion can be found at the description of the command **Trace.STATistic.Sort**.

INCremental	Intermediate results are displayed while TRACE32 PowerView is processing the trace analysis (default).
FULL	TRACE32 PowerView displays the result when the processing is done.

FILE	Use the trace contents loaded with the command **<trace>.FILE**.

TASK *<task_magic>*, etc.	Operating system task in OS-aware debugging and tracing. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).
SplitTASK	Trace information is analyzed independently for each task. The time chart displays these individual results.
MergeTASK	Trace information is analyzed independently for each task. The time chart summarizes these results to a single result.

Option for **SMP** multicore tracing

CORE *<n>*	Time chart is only displayed for the specified core.
SplitCORE	Trace information is analyzed independently for each core. The time chart displays these individual results.
MergeCORE	Trace information is analyzed independently for each core. The time chart summarizes these results to a single result.
JoinCORE	Core information is ignored for the time chart.

RecScale	Display trace in fixed record raster. This is the default.
TimeScale	Display trace as true time display, time relative to the trigger point (respectively the last record in the trace).
TimeZero	Display trace as true time display, time relative to zero point. For more information about the zero point refer to **ZERO**.
TimeREF	Display trace as true time display, time relative to the reference point. For more information about the reference point refer to **<trace>.REF**.

FlowTrace	Trace works as a program flow Trace. This option is usually not required.
BusTrace	Trace works as a bus trace. This option is usually not required.

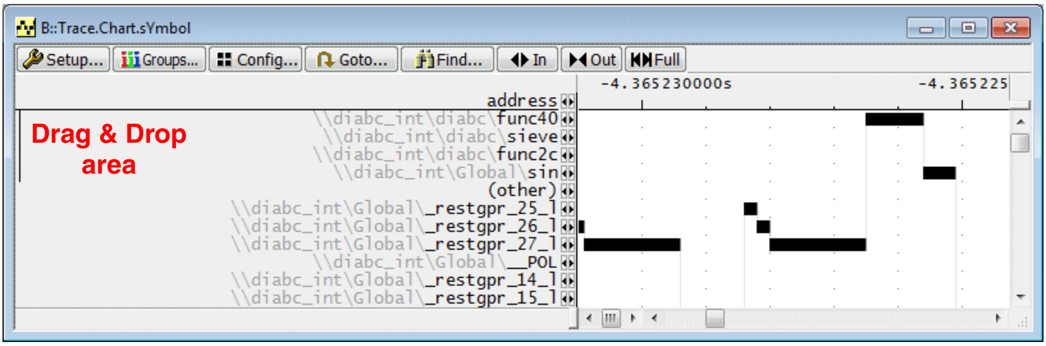
INLINE	Treat inline functions as separate functions (default).
NoINLINE	Discard inline function from the results.
LABEL	Include all symbols in the results.
NoLABEL	Only include functions in the results.

Filter *<item>*	Filter the described item.

Option for ARTIAP trace decoding

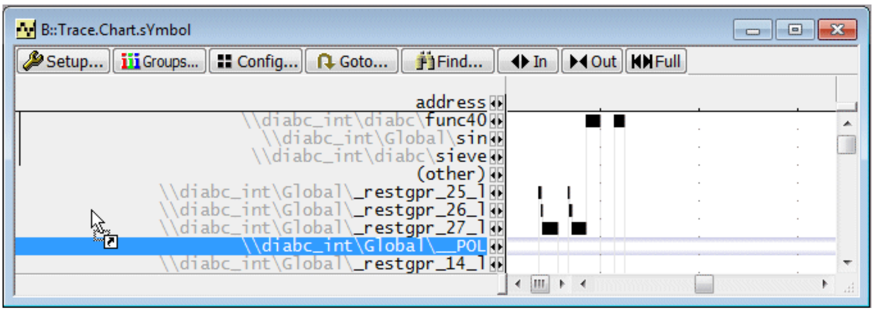
ARTIAP	Option for AUTOSAR Real-Time Interface on Adaptive Platform trace decoding. Decode MIPI STP (System Trace Protocol) format trace which is defined in ARTI Trace Driver on AUTOSAR Adaptive Platform.

A **Trace.Chart** window may contain a Drag & Drop area which is marked by a straight line.

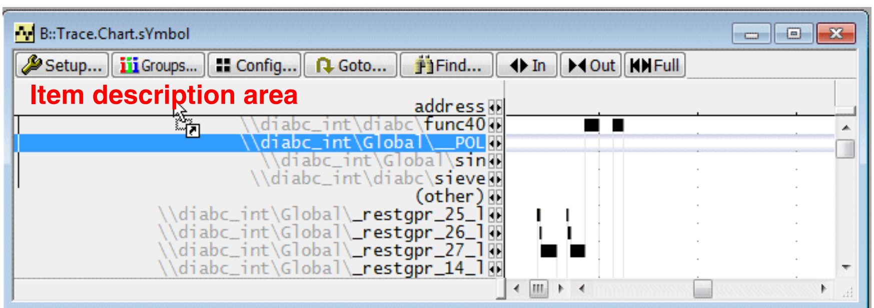


Items of interest can be dragged to the appropriate position in the Drag & Drop area with the left mouse button.

The sort order of all items outside of the Drag & Drop area remains unchanged.



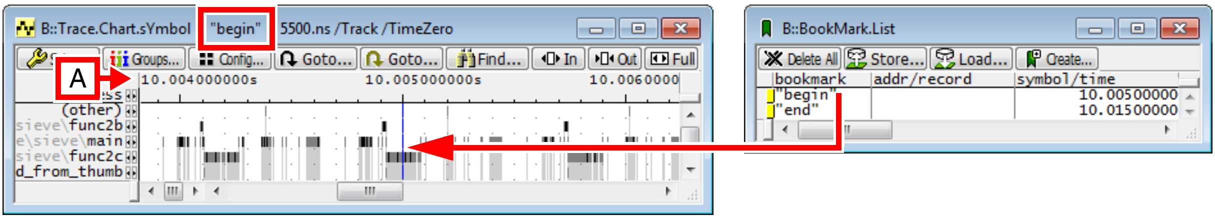
Items can be removed from the Drag & Drop area by dropping them to the item description area.



Example for <trace_bookmark>

[Parameter Descr.]

```
Trace.BookMark "begin" 10.005s
Trace.BookMark "end"   10.010s

Trace.Chart.sYmbol "begin" /Track /TimeZERO

Trace.GOTO "begin" ;highlight the bookmark in the chart

BookMark.List      ;optional: ;display all bookmarks in a list
```

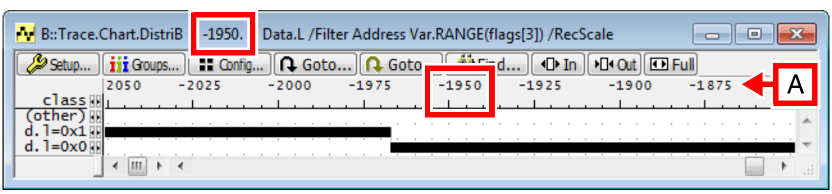


A To display the zero-time timestamps on the x-axis, the **TimeZero** option is used.

Example for <record>

[Parameter Descr.]

```
;print distribution of data values written to flags[3], with the record
;-1950. centered on the x-axis of the window
Trace.Chart.DistriB -1950. Data.L /Filter Address Var.RANGE(flags[3]) \
/RecScale
```



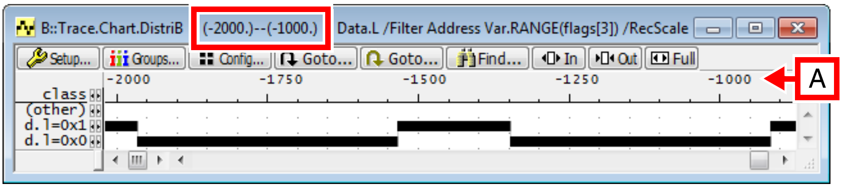
A To display the record numbers on the x-axis, the **RecScale** option is used.

NOTE:	The backslash \ can be used as a line continuation character in PRACTICE script files (*.cmm). No white space permitted after the backslash.

Example for <record_range>

[Parameter Descr.]

```
;print distribution of data values written to flags[3] for the
;record range (-2000.)--(-1000.)
Trace.Chart.DistriB (-2000.)--(-1000.) Data.L /Filter Address \
Var.RANGE(flags[3]) /RecScale
```



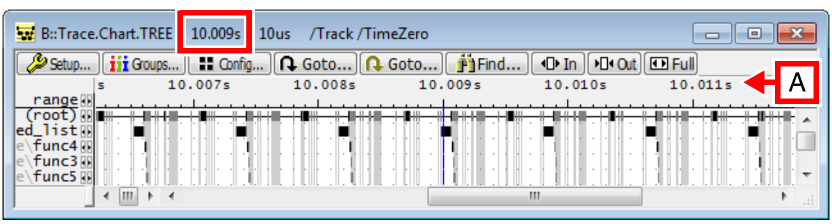
A To display the record numbers on the x-axis, the **RecScale** option is used.

Examples for <time>

[Parameter Descr.]

Example 1:

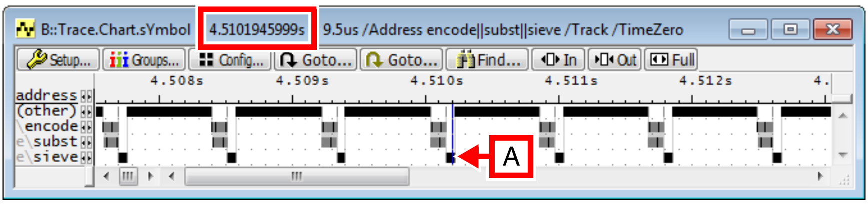
```
;open the chart window with the zero-time timestamp 10.009s and set the
;<timescale> resolution to 10us (optional)
Trace.Chart.TREE 10.009s 10us /Track /TimeZero

Trace.GOTO 10.009s   ;highlight the timestamp in the chart
```



A To display the zero-time timestamps on the x-axis, the **TimeZero** option is used.

Example 2: This PRACTICE script shows how to open the **Trace.Chart.sYmbol** window with a *<time>* parameter that is located 50 microseconds after the 4th occurrence of the HLL symbol `sieve`.

```
;find the first occurrence of the HLL symbol 'sieve'
Trace.Find , sYmbol sieve

RePeaT 3.              ;find the next three occurrences of 'sieve'
   Trace.Find

IF FOUND()==TRUE()     ;if the 4th occurrences of 'sieve' has been found
(
   ;get the timestamp of the 4th occurrence and add an offset of 50.us
   &time=TRACK.TIME()+50.us

   ;open the chart window with the calculated timestamp and set the
   ;<timescale> resolution to 9.5us
   Trace.Chart.sYmbol &time 9.5us /Address encode||subst||sieve \
   /Track /TimeZero

   Trace.GOTO &time    ;highlight the timestamp in the chart
)
```

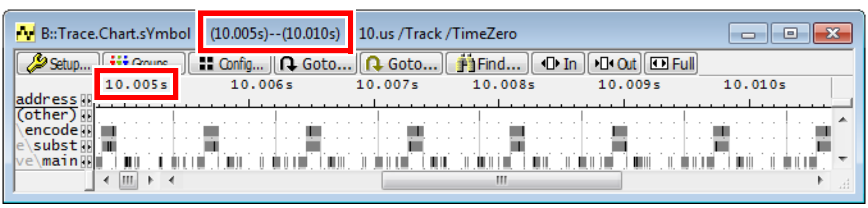


A Location of the calculated timestamp

Example for <time_range>

[Parameter Descr.]

```
Trace.Chart.sYmbol (10.005s)--(10.010s) 10.us /Track /TimeZero
```



Examples for <timescale>

[Parameter Descr.]

Example 1: Using **WinPrint**, you can print the window content without actually opening the window.

```
PRinTer.select WIN   ;select the printer to which you want to print

;print distribution of data values written to flags[3] for the
;record range (-2000.)--(-1000.), use resolution 10.us per pixel
WinPrint.Trace.Chart.DistriB (-2000.)--(-1000.) 10.us Data.L /Filter
Address Var.RANGE(flags[3])
```

Example 2: Using the **WinPOS** command, you can assign a name to a window. Then you open the window and print it with **WinPRT** *<name>*. This example illustrates three different *<timescale>* resolutions.

```
;the following resolutions are used:
;[A] 5.us per pixel, [B] 1.us per pixel, [C] 0.5us per pixel

PRinTer.select WIN   ;select the printer to which you want to print

WinPOS , , , , , ,  W0
Trace.Chart.DistriB (-2000.)--(-1000.)  5.us   Data.L /Filter Address \
Var.RANGE(flags[3])

WinPOS , , , , , ,  W1
Trace.Chart.DistriB (-2000.)--(-1000.)  1.us   Data.L /Filter Address \
Var.RANGE(flags[3])

WinPOS , , , , , ,  W2
Trace.Chart.DistriB (-2000.)--(-1000.)  0.5us  Data.L /Filter Address \
Var.RANGE(flags[3])

WinPRT W0             ;print the window named W0
```

See also

- <trace>.Chart.Address
- <trace>.Chart.AddressGROUP
- <trace>.Chart.ChildTREE
- <trace>.Chart.DatasYmbol
- <trace>.Chart.DistriB
- <trace>.Chart.Func
- <trace>.Chart.GROUP
- <trace>.Chart.INTERRUPT
- <trace>.Chart.INTERRUPTTREE
- <trace>.Chart.Line
- <trace>.Chart.MODULE
- <trace>.Chart.Nesting
- <trace>.Chart.PAddress
- <trace>.Chart.PROGRAM
- <trace>.Chart.PsYmbol
- <trace>.Chart.RUNNABLE
- <trace>.Chart.sYmbol
- <trace>.Chart.TASK
- <trace>.Chart.TASKFunc
- <trace>.Chart.TASKINFO
- <trace>.Chart.TASKINTR
- <trace>.Chart.TASKKernel
- <trace>.Chart.TASKORINTERRUPT
- <trace>.Chart.TASKORINTRState
- <trace>.Chart.TASKSRV
- <trace>.Chart.TASKState
- <trace>.Chart.TASKVSINTERRUPT
- <trace>.Chart.TASKVSINTR
- <trace>.Chart.TREE
- <trace>.Chart.Var
- <trace>.Chart.VarState
- <trace>.PROfileChart
- <trace>.PROfileSTATistic
- <trace>.STATistic
- IProbe.state
- RunTime
- RunTime.state

▲ 'Release Information' in 'Legacy Release History'

<trace>.Chart.Address — Time between program events as a chart

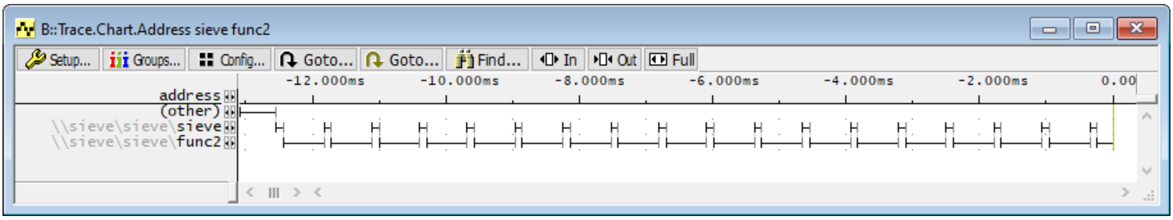
Format:	*<trace>*.**Chart.Address** *<address1>* [*<address2>* …] [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTrack** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

Displays the time interval between up to 8 program events as a chart. The *<trace>*.**Chart.Address** command is the counterpart of the **<trace>.STATistic.Address** command.

<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

Example:

```
Trace.Chart.Address sieve func2
```



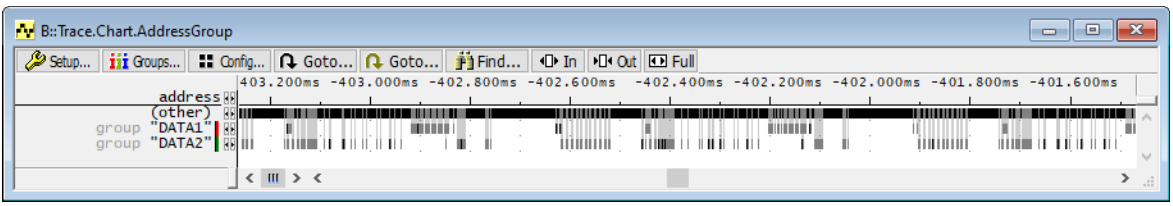
See also

■ <trace>.Chart

Format:	*<trace>*.**Chart.AddressGROUP** [*<list_item>* …] [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Track** | **ZomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Address** *<item>* | *<range>* **INCremental** | **FULL** **Sort** *<item>*

The time for accessed address **groups** is displayed as time chart (flat statistic). The results include groups for both program and data addresses.

<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



Example:

```
GROUP.Create "DATA1" 0x6800--0x68FF /RED

GROUP.Create "DATA2" 0x6700--0x67FF /GREEN

Trace.Chart.AddressGROUP
```

See also

- <trace>.Chart
- <trace>.Chart.GROUP

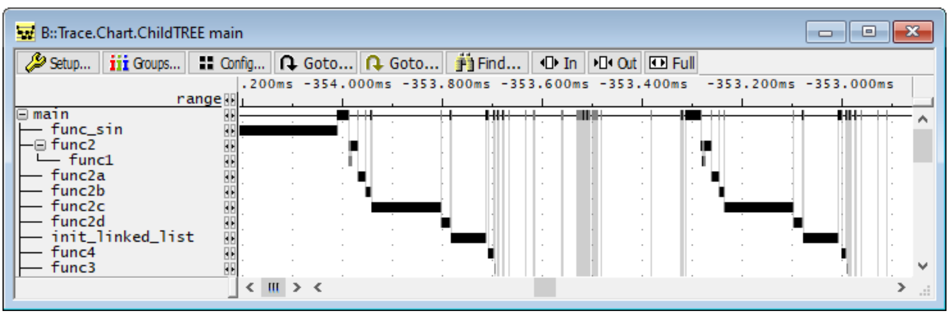
Format:	*<trace>*.**Chart.CTREE** *<address>* [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

The call tree of the selected function is displayed graphically as a chart with the time spent in different functions. The *<trace>*.**Chart.ChildTREE** command is the counterpart of the **<trace>.STATistic.ChildTREE** command.

<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

Example:

```
Trace.Chart.ChildTREE main
```

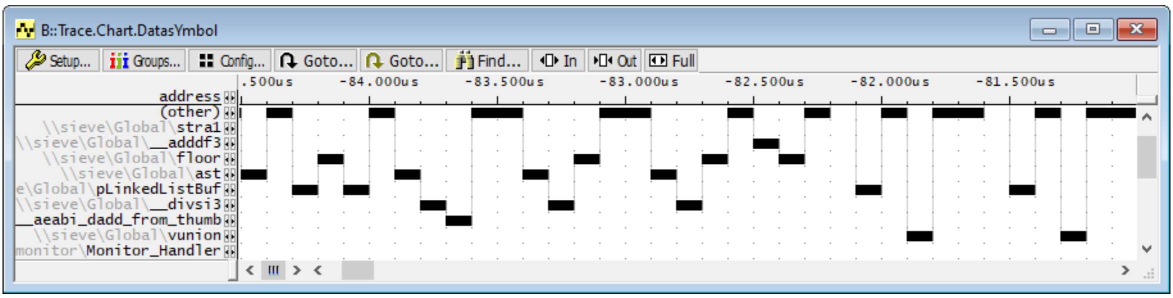


See also

- <trace>.Chart
- CTS.Chart.ChildTREE

Format:	*<trace>*.**Chart.DatasYmbol** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTrack** | **MergeTASK** **LABEL** | **NoLABEL** | **INLINE** | **NoINLINE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

The command **Trace.Chart.DatasYmbol** analyzes the contents of a pointer graphically.



<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

Examples:

```
; analyze the contents of the pointer vpchar graphically
Trace.Chart.DatasYmbol /Filter Address vpchar
```

A more effective usage of the trace memory is possible, if only write accesses to the pointer are recorded to the trace.

```
; set a filter to record only write cycles to the pointer vpchar to the
; trace
Var.Break.Set vpchar /Write /TraceEnable

…

; analyze the contents of the pointer
Trace.Chart.DatasYmbol

; analyze the contents of the pointer, sort the result by symbol names
Trace.Chart.DatasYmbol /Sort sYmbol
```

See also

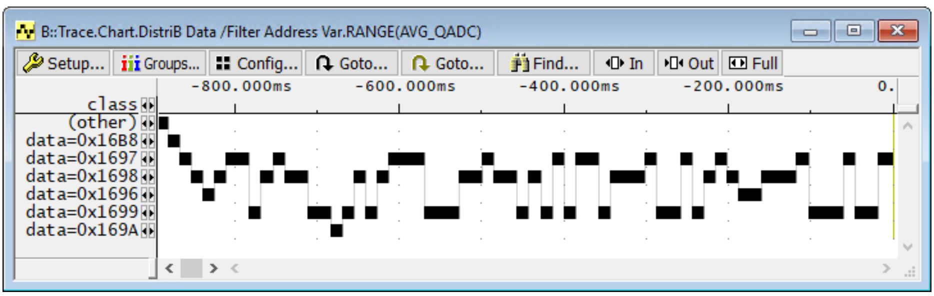
■ <trace>.Chart

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**Chart.DistriB** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTrack** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The distribution of any trace data is displayed if *<item>* is specified. Without argument the distribution of the addresses is displayed symbolically.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



If no selective tracing is done, use the option /**Filter** to filter out the *<item>* of interest.

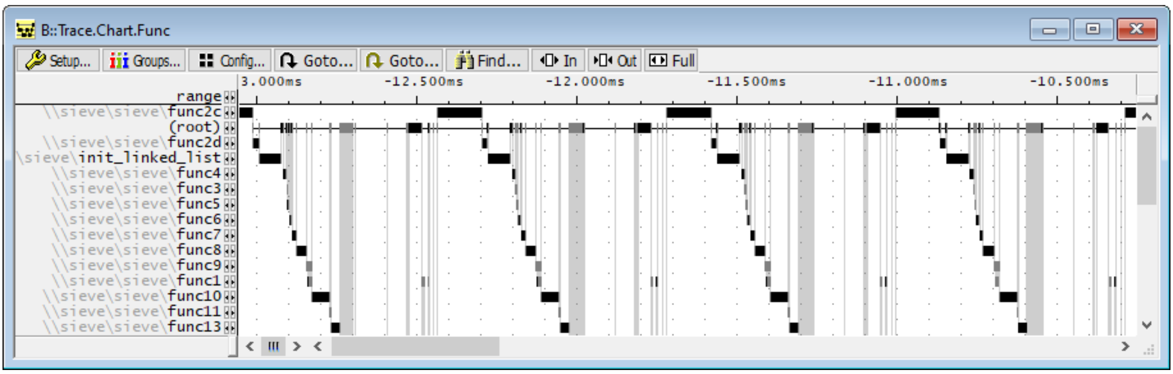
```
; Display distribution of data value for flags[3]
Trace.Chart.DistriB Data.L /Filter Address Var.RANGE(flags[3])

; Display the distribution of data value written for flags[3] for the
; record range (-2000.)--(-1000.)
Trace.Chart.DistriB (-2000.)--(-1000.) Data.L /Filter Address \
Var.RANGE(flags[3])
```

See also

- <trace>.Chart

Format:	*<trace>*.**Chart.Func** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** **INTRROOT** | **INTRTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The time spent in different functions is displayed graphically. The measurement is the same as for the command **<trace>.STATistic.Func**.



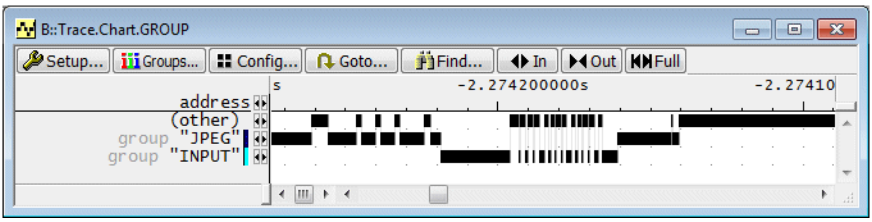
<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

- <trace>.Chart
- CTS.Chart.Func

Format:	*<trace>***.Chart.GROUP** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTrack** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

Displays a GROUP time chart (flat statistic). The results only include groups within the program range. Groups for data addresses are not included.



<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

Example:

```
GROUP.Create "INPUT" \jquant2 \jquant1 \jidctred \jdinput /AQUA

GROUP.Create "JPEG" \jdapimin \jdcolor \jddctmgr \jdcoefct /NAVY

Go

Break

Trace.Chart.GROUP
```

See also

- <trace>.Chart
- <trace>.Chart
- <trace>.Chart.AddressGROUP
- GROUP.Create

▲ 'Release Information' in 'Legacy Release History'

<trace>.Chart.INTERRUPT — Display interrupt chart

Format:	*<trace>***.Chart.INTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Sort** *<item>*

The time spent in different interrupts is displayed graphically.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

■ <trace>.Chart ■ CTS.Chart.INTERRUPT

Format:	*<trace>*.**Chart.INTERRUPTTREE** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Sort** *<item>*

Displays the interrupt nesting as time chart.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

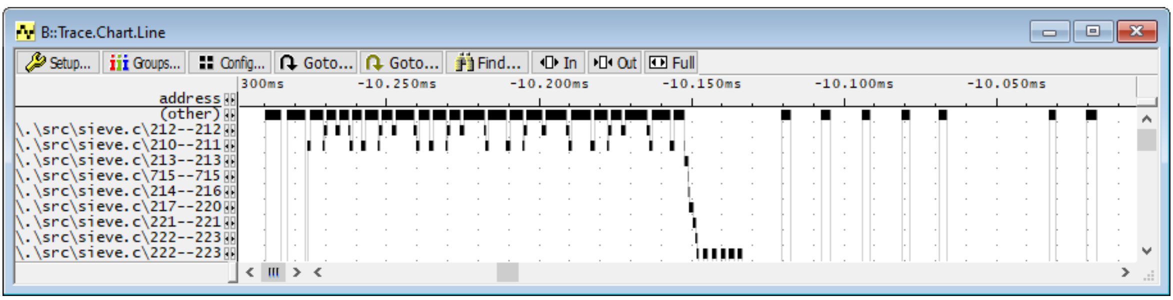
See also

- <trace>.Chart
- CTS.Chart.INTERRUPTTREE

Format:	*<trace>*.**Chart.Line** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTrack** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The time spent in different HLL lines is analyzed graphically.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



See also

- <trace>.Chart
- 'Release Information' in 'Legacy Release History'

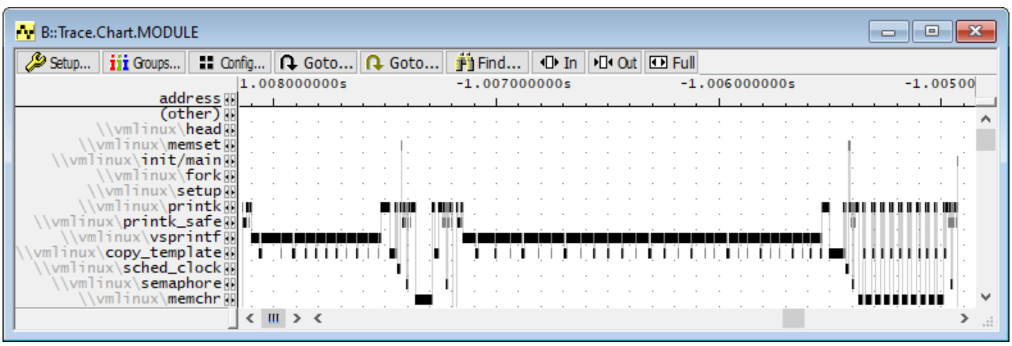
Format: *<trace>***.Chart.MODULE** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>*

<option>: **FILE**
FlowTrace | **BusTrace**
TASK *<task>* **| SplitTASK | MergeTASK**
Track | ZoomTrack
RecScale | **TimeScale** | **TimeZero** | **TimeREF**
INCremental | FULL
Filter *<item>*
Sort *<item>*
Address *<address* | *range>*

Displays the code execution brocken down by symbol module as chart. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



See also

■ <trace>.Chart

Format:	*<trace>*.**Chart.Nesting** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** **IncludeINTR** | **INTRROOT** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

Shows the function call stack as a time chart.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

- <trace>.Chart
- CTS.Chart.Nesting

Format: *<trace>***.Chart.PAddress /Filter Address** [*<trace_area>*] [**/***<option>*]

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

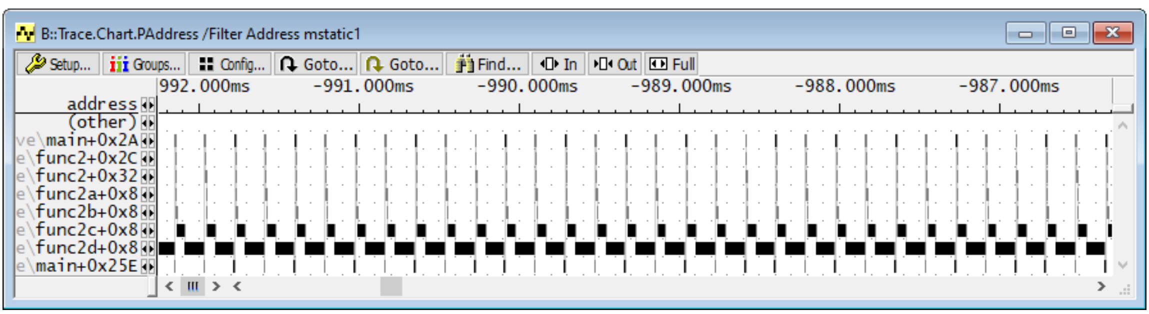
<option>: **FILE**
FlowTrace | **BusTrace**
TASK *<task>* | **SplitTASK** | **MergeTASK**
Track | **ZoomTrack**
RecScale | **TimeScale** | **TimeZero** | **TimeREF**
INCremental | **FULL**
Filter *<item>*
Sort *<item>*
Address *<address* | *range>*

The command provides a graphical chart of the instructions that accessed data addresses. You can select a specific address using the **/Filter** option.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

Example:

```
Trace.Chart.PAddress /Filter Address mstatic1
```



See also

■ <trace>.Chart

Format:	*<trace>***.Chart.PROGRAM** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

Displays the code execution brocken down by loaded object files (programs) as chart. The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

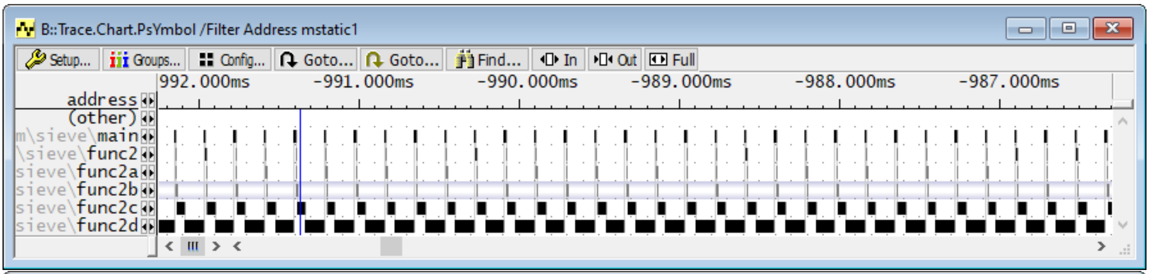
- <trace>.Chart

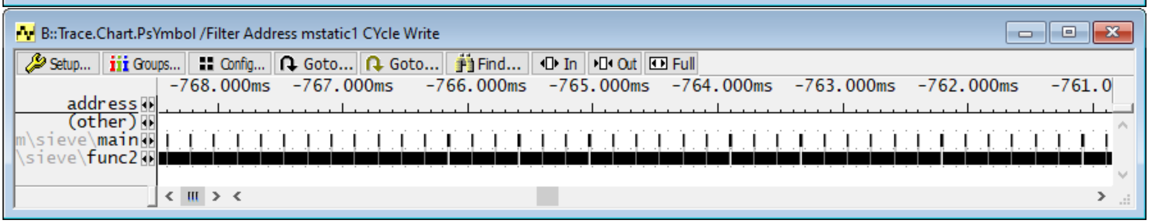
Format:	*<trace>*.**Chart.PsYmbol** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* **| SplitTASK | MergeTASK** **Track | ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental | FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The command provides a graphical chart of the functions that accessed the data addresses. You can select a specific address using the **/Filter** option.

Examples:

```
; display a chart of all functions that accessed the variable mstatic1
Trace.Chart.PsYmbol /Filter sYmbol mstatic1

; display a chart of all functions that performed a write access to the
; variable mstatic1
Trace.Chart.PsYmbol /Filter sYmbol mstatic1 CYcle Write
```





See also

- <trace>.Chart

Format:	*<trace>*.**Chart.RUNNABLE** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* **CORE** *<item>* | **SplitCORE** | **MergeCORE** **INTRROOT** | **INTRTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The time spent in different AUTOSAR Runnables is displayed graphically. 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>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

On TriCore AURIX there's a solution available for the Vector AUTOSAR tools that uses an automated instrumentation to trace runnables on all cores with minimum overhead. See *~~/demo/env/vector/rte_profiling*.

Otherwise, all functions that start an AUTOSAR "Runnable" have to be marked with the command **sYmbol.MARKER.Create RUNNABLESTARTPLUSSTOP**. Please refer to **"Trace Export for Third-Party Timing Tools"** (app_timing_tools.pdf) for more information.

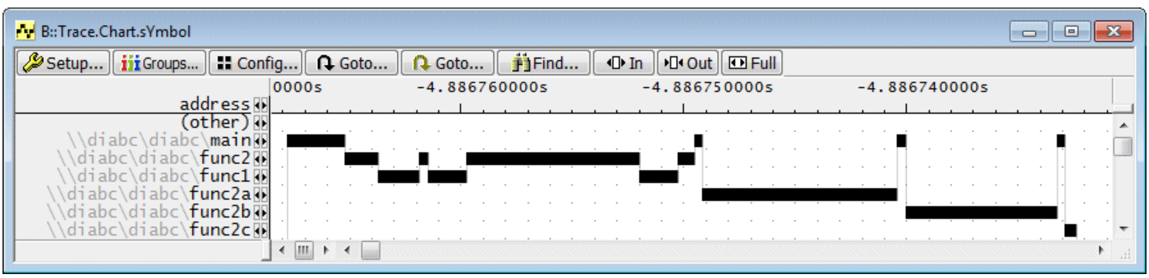
See also

- ■ <trace>.Chart ■ CTS.Chart.RUNNABLE ■ TASK.Create.RUNNABLE ■ TASK.List.RUNNABLES
- ▲ 'Runnable Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**Chart.sYmbol** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **LABEL** | **NoLABEL** **INLINE** | **NoINLINE** **Track** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Address** *<function1>***||***<function2>* … **Address** *<function1>***--***<function2>* **Filter Address** *<function1>***||***<function2>* … **Filter Address** *<function1>***--***<function2>*

The distribution of program execution time at different symbols is displayed as a time chart. This can be used to get a quick overview about the functions sampled in the trace buffer.

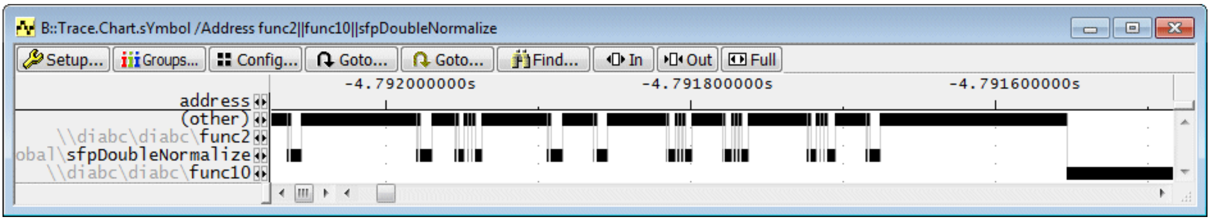
<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



Example:

```
Go
Break
Trace.STATistic.Sort sYmbol              ; sort the result alphabetically
```

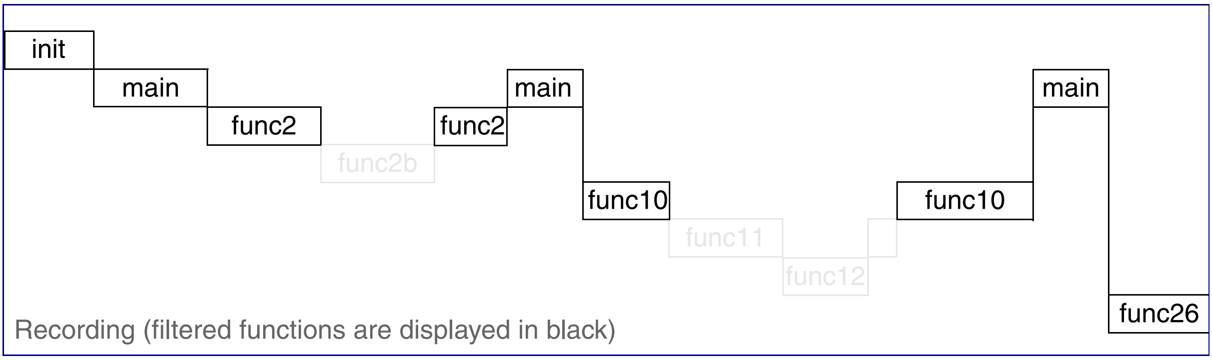
```
; draw time chart for specified functions, assign time for all other
; functions to (other)
Trace.Chart.sYmbol /Address func2||func10||sfpDoubleNormalize
```



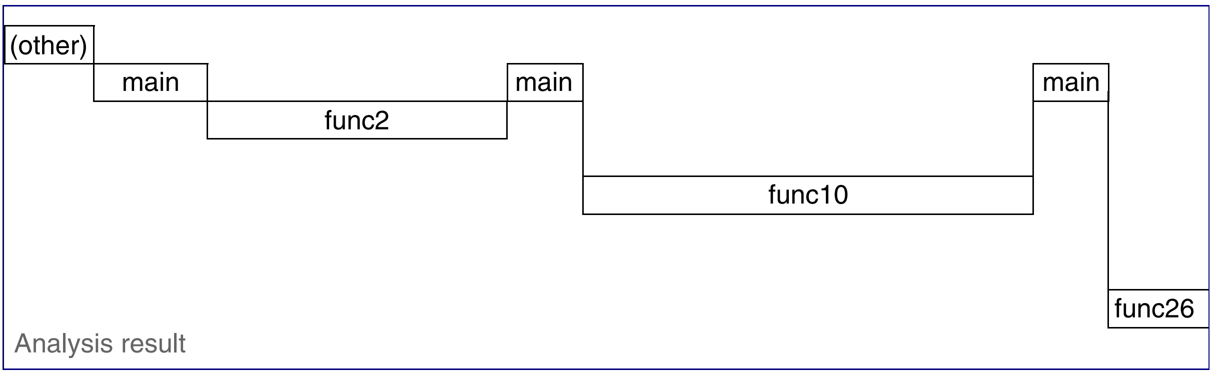

```
; draw time chart for specified functions (address range), assign time
; for all other functions to (other)
Trace.Chart.sYmbol /Address func2--func10
```

The **GROUP** command provides more features to structure your time chart.

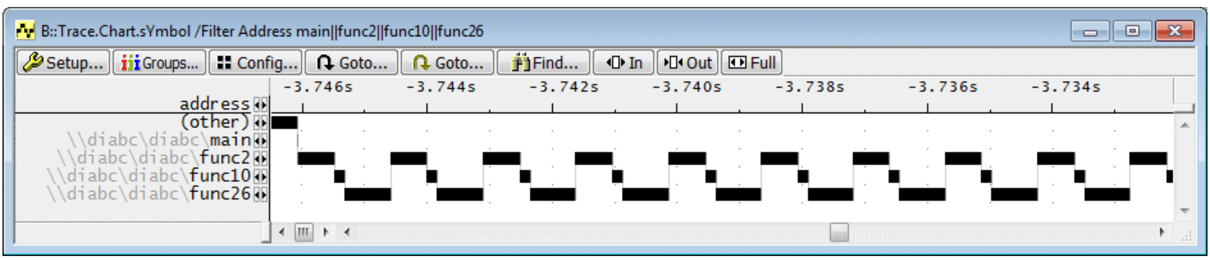
```
; filter specified functions out of the address stream
; and draw time chart for filtered trace information
Trace.Chart.sYmbol /Filter Address main||func2||func10||func26
```



Recording (filtered functions are displayed in black)



Analysis result



See also

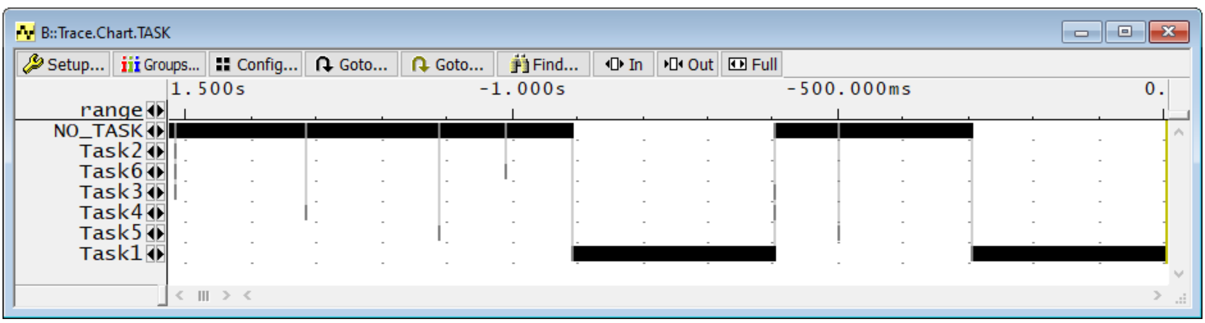
■ <trace>.Chart ■ <trace>.STATistic.sYmbol ■ CTS.Chart.sYmbol

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**Chart.TASK** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **ARTIAP**

Displays the time spent in different tasks. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



See also

■ <trace>.Chart ■ CTS.Chart.TASK

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**Chart.TASKFunc** [*<record_range>*] [*<scale>*] [**/***<option>*] (legacy)
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

For details, refer to **<trace>.Chart.Func**.

See also

■ <trace>.Chart

<trace>.Chart.TASKINFO Context ID special messages

Format:	*<trace>*.**Chart.TASKINFO** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

Displays a time chart of special messages written to the Context ID register for ETM trace. The range of special values has to be reserved with the **ETM.ReserveContextID** command. These special values are then not interpreted for task switch or memory space switch detection.

This can be used for cores without data trace to pass data by the target application to the trace tool by writing to the ContextID register.

See also

- [<trace>.Chart](#)
- [CTS.Chart.TASKINFO](#)

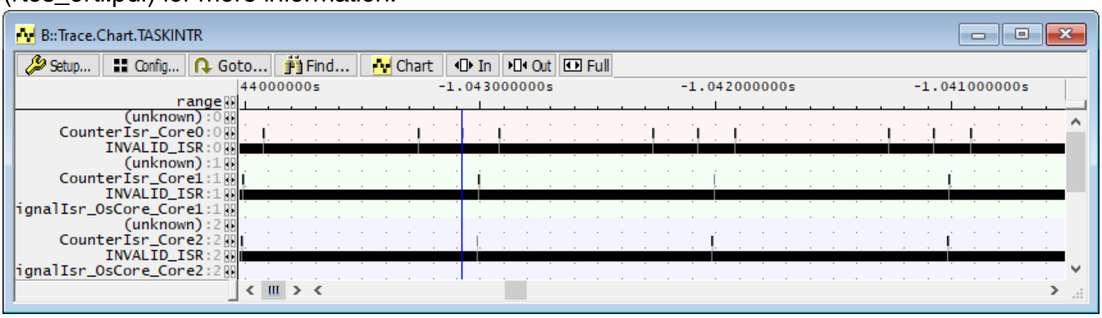
<trace>.Chart.TASKINTR — Display ISR2 time chart (ORTI)

Format: *<trace>***.Chart.TASKINTR** [*<trace_area>*] [**/***<option>*]

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

<option>: **FILE**
FlowTrace | **BusTrace**
Track | **ZoomTrack**
CORE *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE**
RecScale | **TimeScale** | **TimeZero** | **TimeREF**
INCremental | **FULL**
Filter *<item>*
Sort *<item>*

Displays an ORTI based ISR2 time chart. This feature can only be used if the 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>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

- [<trace>.Chart](#)
- [CTS.Chart.TASKINTR](#)

▲ 'ISR2 Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'
▲ 'Trace Features' in 'OS Awareness Manual OSEK/ORTI'

<trace>.Chart.TASKKernel — Task run-time chart with kernel markers (flat)

Format:	*<trace>*.**Chart.TASKKernel** [*<trace_area>*] [**/**<*option*>]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

Time chart for results of **Trace.STATistic.TASKKernel**. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

- <trace>.Chart
- CTS.Chart.TASKKernel

Format:	*<trace>*.**Chart.TASKORINTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track | ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental | FULL** **Filter** *<item>* **Sort** *<item>*

Displays the time spent in different tasks and interrupts as time chart. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

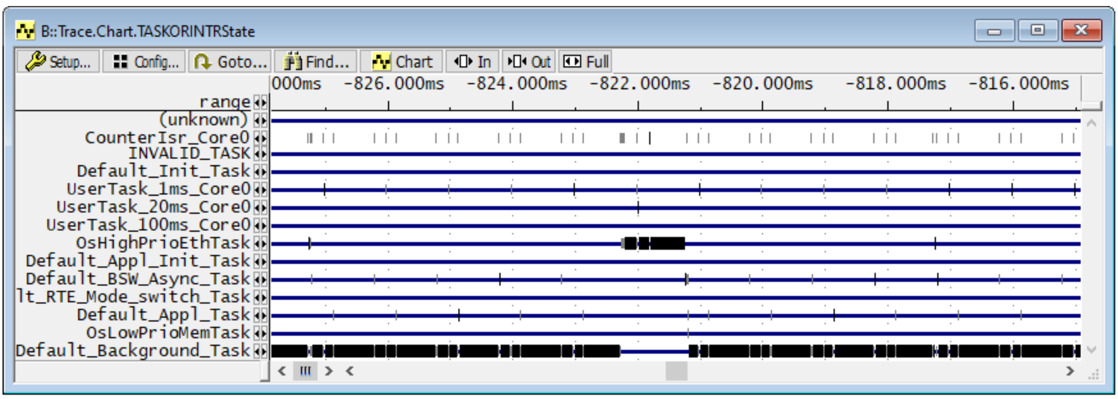
See also

- <trace>.Chart
- CTS.Chart.TASKORINTERRUPT

Format:	*<trace>*.**Chart.TASKORINTRState** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

Displays a graphical chart of task and ORTI-based ISR2 states. Before using this function the interrupt and task state transitions must be sampled by the trace. This feature is highly dependent on the used RTOS kernel, and needs the **TASK** to be configured. Please see kernel specific **"OS Awareness Manuals"** manuals for more information.

Refer for more information to **<trace>.Chart.TASKState**.



See also

■ <trace>.Chart

▲ 'ISR2 Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'

Format:	*<trace>***.Chart.TASKSRV** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

The time spent in OS service routines and different tasks is displayed. Service routines that are used by multiple tasks are displayed for each task. 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>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

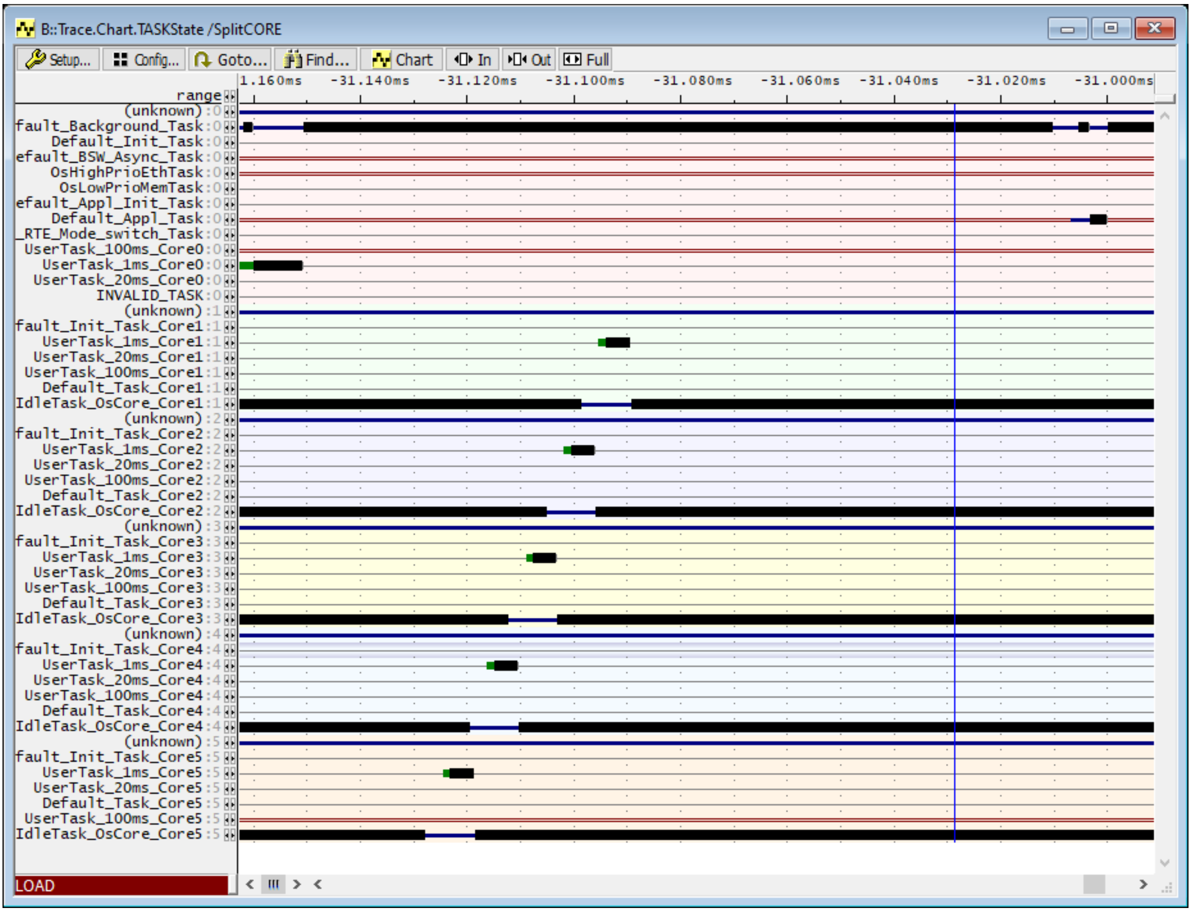
See also

- <trace>.Chart

Format:	*<trace>***.Chart.TASKState** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track | ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental | FULL** **Filter** *<item>* **Sort** *<item>* **ARTIAP**

The time different task spent in specific states is displayed. Before using this function the task state transitions must be sampled by the trace. This feature is highly dependent on the used RTOS kernel, and needs the **TASK** to be configured. Please see kernel specific **"OS Awareness Manuals"** manuals for more information.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.



Graphics		
running	solid black bar	
ready	medium blue bar	
waiting	two thin red lines	
suspended	thin grey line	
activated	green or red line	
released	double thick cyan line	
interrupted	thin red line	
undefined/unknown	no line	

See also

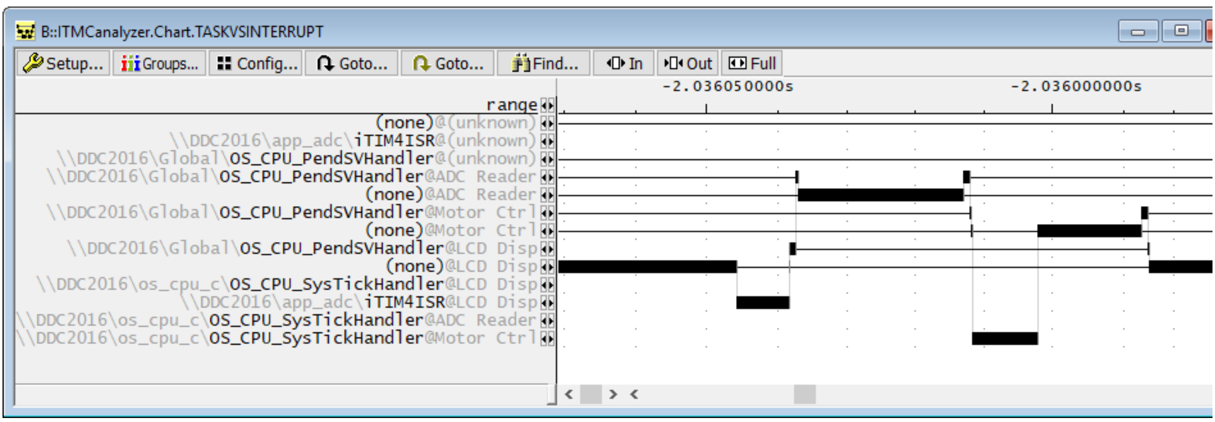
- <trace>.Chart

▲ 'Task Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'

<trace>.Chart.TASKVSINTERRUPT — Time chart of interrupted tasks

Format:	*<trace>*.**Chart.TASKVSINTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

Shows a graphical representation of tasks that were interrupted by interrupt service routines. This feature is only available if TRACE32 has been set for OS-aware debugging.



<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

■ <trace>.Chart ■ CTS.Chart.TASKVSINTERRUPT

Format:	*<trace>*.**Chart.TASKVSINTR** [*<trace_area>*] [**/***<options>* …]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

Displays a time-chart for task-related interrupt service routines. 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.

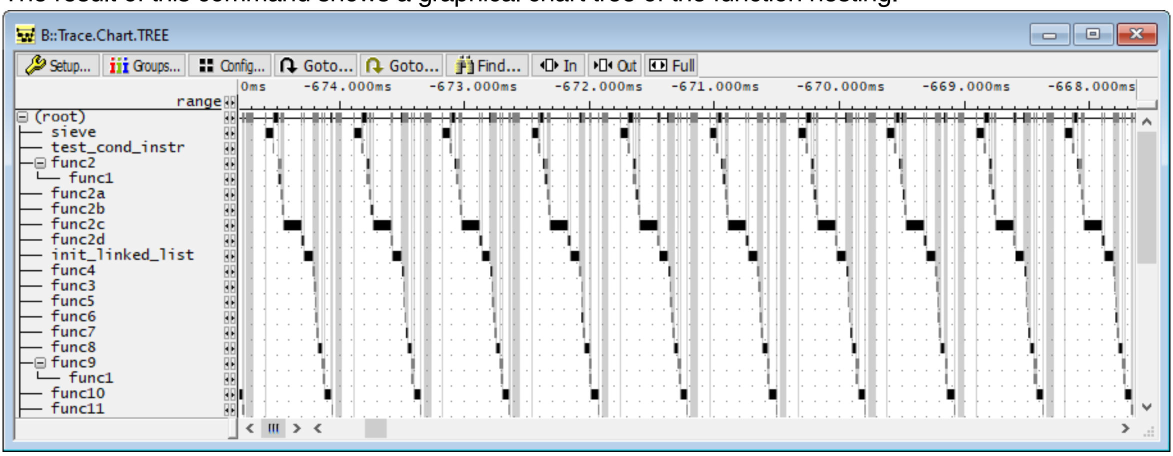
<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

- <trace>.Chart
- CTS.Chart.TASKVSINTR

Format:	*<trace>*.**Chart.TREE** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **TASK** *<task>* **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The result of this command shows a graphical chart tree of the function nesting.



<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

See also

- <trace>.Chart
- CTS.Chart.TREE

Format:	*<trace>*.**Chart.Var** [*<trace_area>*] [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>*

The command provides a graphical chart of variable accesses.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

Example:

```
; Display a graphical chart of all variable accesses:
Trace.Chart.Var /Filter sYmbol mstatic1 /Filter CYcle Write

; Display a graphical chart of write accesses to the mstatic1 variable
Trace.Chart.Var /Filter sYmbol mstatic1 /Filter CYcle Write
```

See also

■ <trace>.Chart

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**Chart.VarState** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **FILL** | **FillFirst** **DECODE** *<value>* … **INCremental** | **FULL** **Filter** *<item>* **Sort** *<item>*

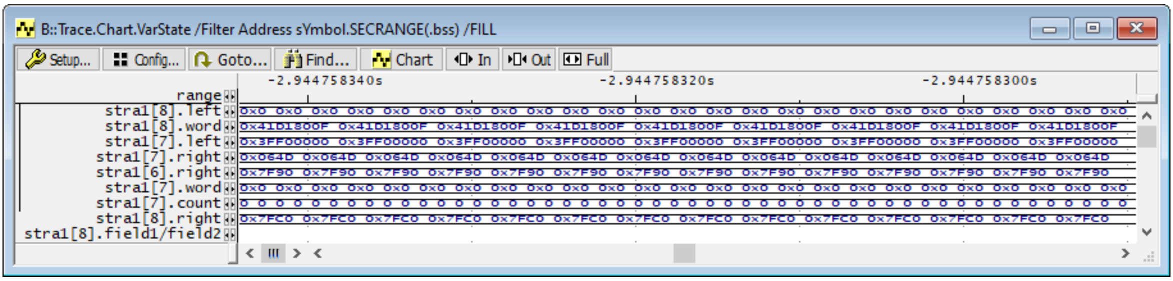
Displays the contents of variables over the time. Each variable access must be sampled with one single CPU cycle. If an address is not a variable it is displayed in form of a single marker. This can be used to track program execution addresses.

<trace_area>	For parameter descriptions and examples, see **Parameters**.
<option>	Refer to **<trace>.Chart** for a description of the **<trace>.Chart** options.

FILL	Repeat the value instead of displaying the value only directly after the transition.
FillFirst	Repeat the value without any space instead of displaying the value only directly after the transition.
DECODE *<value>*	Define a decoding for enumeration variables.

Example: use the /Filter option to filter out the variables

```
Go
Break
Trace.Chart.VarState /Filter Address sYmbol.SECRANGE(.bss)
```



See also

- ■ <trace>.Chart
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**CLOCK** *<trace>*.**CLOCK** *<frequency>* *<trace>*.**CLOCK** *<frequency0> <frequency1> …* (SMP tracing only)

Set clock frequency for processor generated timestamps. If called without parameter, time measurements using processor generated timestamps are disabled.

Some trace protocols can generate cycle count information. TRACE32 can calculate time information out of the cycle count information if the appropriate clock frequency is specified with the **Trace.CLOCK** command.

For most trace protocols cycle count indicates the number of core clock cycles. That's why *<frequency>* has to be the core clock frequency. Please be aware the specifying the core clock frequency only makes sense if the frequency was constant while recording.

Example for the ARM-ETM:

```
ETM.TImeMode CycleAccurate

Trace.CLOCK 800.MHz
```

If the cores of an SMP run at different speeds, the frequency can be specified per core.

```
ETM.TImeMode CycleAccurate

Trace.CLOCK 800.MHz 600.MHz 1.GHz
```

See also

■ ETM.CycleAccurate ■ RunTime ■ RunTime.state

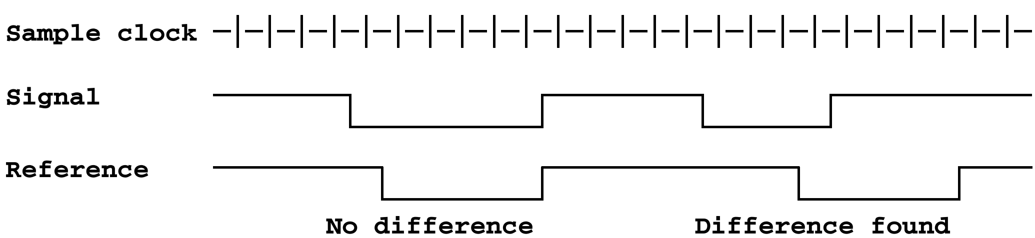
Format:	*<trace>*.**ComPare** [*<record_range>*] [*<record_number>*] [{*<items>*}] [{**/***<options>*}]
<options>:	**Tolerance** *<count>* **FILE** **Back**

Compares the trace contents. If the command **<trace>.ComPare** is used without arguments the previous compare is repeated.

<item>	Only the given *<item>* … are compared.
<record_range>, *<record_number>*	If *<record_range>* and *<record_number>* are not used, a comparison of the complete trace is performed.
FILE	Compare the trace contents with the loaded file. See also **Trace.FILE**.
Back	Compare backwards.
Tolerance *<count>*	When external asynchronous data are traced, a jitter in the signal will result in different sampling data. In this case the precision of the compare function may be controlled by the option **Tolerance**.

The compare function will set the pointers for the tracking option. All analyzer windows, which are in track mode, will follow these pointers.

For valid channel names refer to the **Processor Architecture Manuals**.



Examples:

```
; compare the current trace contents from record (500.--1000.) with the
; current trace contents starting at record number 5000. with regards to
; the address
Trace.ComPare (500.--1000.) 5000. Address
```

```
; load saved trace contents
Trace.FILE old_trace

; compare the current trace contents from record (500.--1000.) with the
; loaded trace contents starting at record number 300. with regards to
; the data on byte 0
Trace.ComPare (500.--1000.) 300. Data.B0 /FILE
```

```
; load saved trace contents
Trace.FILE old_trace

; compare the complete current trace contents with the complete
; loaded trace contents with regards to the data on byte 0
Trace.ComPare Data.B0 /FILE

; Repeat the previous compare
Trace.ComPare
```

```
; load saved trace contents
Trace.FILE old_trace

; compare the complete current trace contents with the complete
; loaded trace contents with regards to the data and address
Trace.ComPare Data Address /FILE
```

```
; compare against file TEST1 on line RXD
Port.FILE TEST1                              ; load reference file
Port.ComPare RXD /Tolerance 3. /FILE         ; compare line RXD
IF FOUND()                                   ; print result if difference
    PRINT "Difference found"                 ; will be found
;…
Port.ComPare                                 ; search for next difference
```

See also

- IProbe.state
- RunTime
- RunTime.state

Format:	*<trace>***.ComPareCODE** [*<access class>*] [/*<option>*]
<option>	**FILE** **FlowTrace** | **BusTrace**

Compares the trace with the memory contents. This command can e.g. be used to check if the loaded trace matches the loaded binary in the TRACE32 Instruction Set Simulator.

PRACTICE script examples of custom trace demos can be found in the following *_demo.cmm files:

- ~~/demo/customtrace/pipe_dll/dll_stp_demo.cmm
- ~~/demo/customtrace/pipe_dll/dll_csstm_demo.cmm
- ~~/demo/customtrace/pipe_dll/dll_itm_demo.cmm

For details about these files, refer to the readme.txt in the demo folder.

See also

- <trace>.CustomTrace.<label>.COMMAND
- <trace>.CustomTrace.<label>.ListString
- <trace>.CustomTrace.<label>.UNLOAD
- <trace>.CustomTraceLoad
- Trace

- ▲ 'Software Trace with the ITM' in 'CombiProbe for Cortex-M User's Guide'
- ▲ 'Software Trace with the ITM' in 'MicroTrace for Cortex-M User's Guide'

<trace>.CustomTrace.<label>.COMMAND Send command to specific DLL

Format:	*<trace>*.**CustomTrace.***<label>*.**COMMAND** *<command_line_args>* *<trace>*.**PipePROTO.COMMAND** [*<cmd_line_args>*] (deprecated)

Sends a command to a specific DLL that has been assigned a user-defined *<label>*.

See also

- <trace>.CustomTrace

<trace>.CustomTrace.<label>.ListString Display ASCII strings

Format:	*<trace>*.**CustomTrace.***<label>*.**ListString**

Displays ASCII strings logged by the DLL.

See also

- <trace>.CustomTrace

<trace>.CustomTrace.<label>.UNLOAD — Unload a single DLL

Format:	*<trace>*.**CustomTrace.***<label>*.**UNLOAD**

Unloads a single DLL identified by *<label>*.

See also

- <trace>.CustomTrace

<trace>.CustomTraceLoad — Load a DLL for trace analysis/Unload all DLLs

Format 1:	*<trace>*.**CustomTraceLoad** "*<name>*" *<file>* *<trace>*.**PipePROTO.load** *<dll_name>* [*<cmd_line_args>*] (deprecated)
Format 2:	*<trace>*.**CustomTraceLoad ""** *<trace>*.**PipePROTO** (deprecated)

Format 1: TRACE32 supports a mechanism for passing trace data to a shared library or DLL allowing for custom trace handling. This command loads the shared object.

Format 2: When executed with an empty string, the command unloads *all* DLLs.

NOTE:	Use the command **<trace>.CustomTrace.<label>.UNLOAD** to unload *a single* DLL.

<name>	A user-defined name for the DLL or shared object. TRACE32 supports up to 8 loaded shared objects at any one time. The *<name>* is used to differentiate them.
<file>	A shared library or DLL which is appropriate for your host Operating System. This DLL will receive trace data from TRACE32 and perform custom analysis on it.

See also

- <trace>.CustomTrace
- Trace

<trace>.DISable

Format:	*<trace>*.**DISable**

Disables the trace.

See also

- IProbe.state
- RunTime
- RunTime.state

<trace>.DisConfig — Trace disassembler configuration

For background information and examples about how to use the **<trace>.DisConfig** command group, see:

- **"PowerIntegrator Trace DisConfig Application Note"** (powerintegrator_app_dc.pdf)
- **www.lauterbach.com/publications/advanced_debug_with_powerintegrator.pdf**

See also

- IProbe.state
- <trace>.DisConfig.FlowMode
- Integrator.DisConfig.LOAD
- <trace>.DisConfig.CYcle
- <trace>.DisConfig.RESet

▲ 'General Function' in 'PowerIntegrator Trace DisConfig Application Note'

<trace>.DisConfig.CYcle — Trace disassemble setting

Format:	*<trace>*.**DisConfig.CYcle** "*<name>* [, *<ext>*]" *<cycle>*
<cycle>:	**Read** *<definition>* **Write** *<definition>* **Fetch** *<definition>* **FLOW** *<definition>* **Fetch1** *<definition>* **ReadOrFetch** *<definition>* **ReadSpecial** *<definition>* **WriteSpecial** *<definition>* **MERGE** ["*<name>*" *<offset>*...]
<definition>:	**TransientStrobe** [*<time>*] [*<channels>*] **Strobe**[2|3] [[*<channel>* [**Low** | **High** | **Falling** | **Rising**]] **Strobe**[2|3]**Sample** [**Last** | **Next** | **AT number**] [*<channel>* [**Low** | **High** | **Falling** | **Rising**]] **Address**[2]**Sample** [**Last** | **Next** | **AT number**] [*<channel>* [**Low** | **High** | **Falling** | **Rising**]] **Address**[2] [*<channels>* **Address**[2]**SHift** *<value>* **AddressBase** *<address>*

Data[**2**]**Sample** [**Last** | **Next** | **AT number**] [*<channel>* [**Low** | **High** | **Falling** | **Rising**]]

Data[**2**] [*<channels>*]

Data[**2**]**SHift** *<value>*

DataUnknown

DataWidthUnknown

SpaceID | **SpaceIDSample**

Word | **Group** | **Integrator.***<x>* | **eXt.***<x>*

The command **<trace>.DisConfig.CYcle** informs the trace software where to find program-fetch, data-read and data-write cycles in a not qualified trace recording which was taken by the PowerProbe or PowerIntegrator. With this information a standard bus trace listing can be generated.

<name>, *<ext>*	"name" is displayed in the cycle-type row of the **Trace.List** window. Its length is limited to 7. The "ext" is not displayed but used to differ between cycle types. Example: "rd_byte,0" --> rd_byte. This way it is possible to define different cycle types (rd_byte,0; rd_byte,1 …) which are displayed in the same way (rd_byte)
<cycle>	Is used by the trace disassembler • **Read**: data read cycle • **Write**: data write cycle • **Fetch**: program fetch cycle • Fetch1: first program fetch code of an instruction • **ReadOrFetch**: data-read or program-fetch cycle. The disassembler will do the final decision out of the program flow knowledge • **ReadSpecial**: special read cycle (e.g. dma) • **WriteSpecial**: special write cycle (e.g. dma) • **MERGE**: merge the data of multiple cycles
<definition>	Defines where to find a *<cycle>* in the trace, where to find the appropriate address and data, and how to display them.

See also

■ <trace>.DisConfig

<trace>.DisConfig.FlowMode — Enable FlowTrace analysis

Format:	*<trace>*.**DisConfig.FlowMode** [**ETMB** | **ETMK** | **OFF**]

Enables the analysis of certain FlowTrace protocols like ARM-ETM.

OFF	FlowTrace analysis disabled
ETMB	ARM-ETM FlowTrace analysis enabled, Mictor probe AB in use.
ETMK	ARM-ETM FlowTrace analysis enabled, Mictor probe JK in use.

See also

■ <trace>.DisConfig

<trace>.DisConfig.RESet — Reset trace disassemble setting

Format:	*<trace>*.**DisConfig.RESet**

Resets the trace disassemble setting.

See also

■ <trace>.DisConfig

The **<trace>.DRAW** command group can be used to plot the values of recorded trace data against time. An introduction to the usage of the **Trace.DRAW** commands is provided in **"Application Note for Trace.DRAW"** (app_trace_draw.pdf).

Keywords for <format>

Decimal	Display the data as decimal number.
DecimalU	Display the data as unsigned decimal number.
Hex	Display the data as hexadecimal number.
HexS	Display the data as signed hexadecimal number.
OCTal	Display the data as octal number.
Float	The following floating-point formats are available: Ieee | IeeeQuad | IeeeRev | IeeeS | IeeeHalf | ArmHalf IeeeXt80 | IeeeXt96 | IeeeXt96G IeeeDbl | IeeeDblS | IeeeDblT | MFFP | Pdp11 | Pdp11Dbl | RTOSUH | RTOSUHD | Dsp16 | Dsp16C | Dsp16Fix | Dsp32Fix | M56 | M560 | M561 | LACCUM | Fract8 | Fract16 | Fract24 | Fract32 | Fract48 | Fract64 | Fract40G | Fract72G UFract8 | UFract16 | UFract24 | UFract32 | UFract48 | UFract64 | MICRO | MICRO64 | MILLI | MILLI64 | NANO64 | PICO64
Byte, Word, ...	See **"Keywords for <width>"**, page 202.

Keywords for <width>

Byte	8-bit
Word	16-bit
TByte	24-bit (tribyte)
Long	32-bit (long word)
PByte	40-bit (pentabyte)
HByte	48-bit (hexabyte)
SByte	56-bit (septuabyte)
Quad	64-bit (quad word)

General Options

FILE	Visualize the trace contents loaded with the command **<trace>.FILE**.
BusTrace	This option is usually not required. It switches off the FlowTrace decoder. In the bus trace mode, all valid bus cycles are sampled.
RecScale	The resolution on the x-axis is based on trace record numbers. This is the default if timestamps are not available.
TimeScale	The resolution on the x-axis is based on timestamps.
TimeZero	Display the trace as a real-time display, time relative to the zero point. For more information about the zero point refer to **ZERO**.
TimeREF	Display the trace as a real-time display, time relative to the reference point. For more information about the reference point refer to **<trace>.REF**.
FIRST *<address>*	Define which address contains the first part of the data value if the data value cannot be sampled within one bus cycle (e.g. a 16 bit data value on a 8 bit data bus).
Track	The cursor in the **<trace>.DRAW** window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed. The zoom factor of the **<trace>.DRAW** window is retained, even if the trace content changes.

ZoomTrack	Same as option **Track**. If the tracking in performed with another **<trace>.DRAW** window the same zoom factor is used.
Filter *<filter_items>*	Filter the described item.

Draw Options

Points	Display each data value as a dot.
Vector	Connect the dots for the data values by vectors.
MarkedVector	Same as Vector, with every trace record holding a data value marked with a vertical line.
Steps	Connect the dots for the data values by steps.
Impulses	Draw each data value as a single pulse.
MinMax	Display minimum and maximum values.
LOG	Display the data values in a logarithmic format.
Color	Remove the color legend when multiple addresses are displayed in the **<trace>.DRAW** window.

See also

- <trace>.DRAW.channel
- <trace>.DRAW.Var
- Data.DRAW
- Data.DRAWXY
- IProbe.state
- <trace>.DRAW.Data
- CAnalyzer.<specific_cmds>
- Data.DRAWFFT
- Data.IMAGE
- Var.DRAW

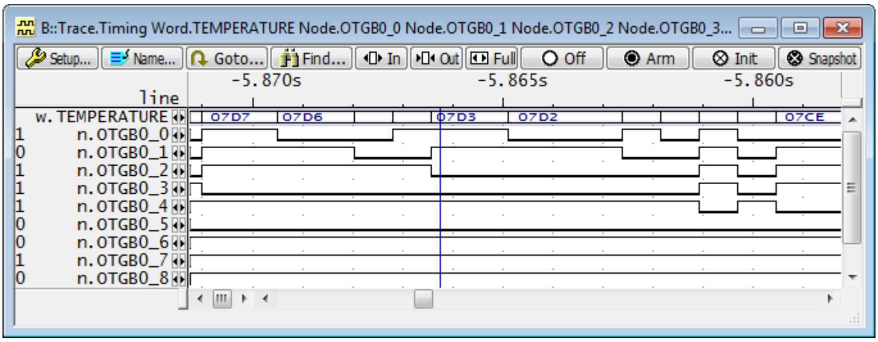
▲ 'Introduction' in 'Application Note for Trace.DRAW'
▲ 'Release Information' in 'Legacy Release History'

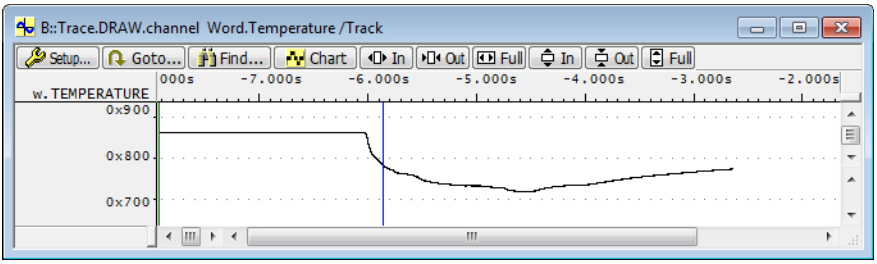
Format:	*<trace>***.DRAW.channel** [*<start>* | *<range>*] [**%***<format>*] [*<items …>*] [**/***<options>*] *<trace>***.Chart.Draw** (deprecated)
<format>:	**Decimal.** [*<width>*] **DecimalU.** [*<width>*] **Hex.** [*<width>*] **HexS.** [*<width>*] **OCTal.** [*<width>*] **Float.** [**Ieee** | **IeeeDbl** | **IeeeeXt** | **IeeeMFFP** | …] **Byte** | **Word** | **Long** | **Quad** | **TByte** | **PByte** | **HByte** | **SByte**
<width>:	**DEFault** | **Byte** | **Word** | **Long** | **Quad** | **TByte** | **PByte** | **HByte** | **SByte**
<items>:	**ENERGY.Abs** | **POWER …**
<options>:	*<draw_options>* | **FILE** | **BusTrace** | **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** | **FIRST** *<address>* | **Filter** *<filter_items>* | **Track** | **ZoomTrack**
<draw_options>:	**Points** | **Vector** | **MarkedVector** | **Steps** | **Impulses** | **MinMax** | **LOG** | **Color**
<filter_items>:	*<range>* | *<address>* | *<bitmask>*

Plot specified *<item>* against time. This command is mainly used to plot no-data items. Please refer **<trace>.DRAW.Data** for a description of the different parameters and options.

<start>	Start point of the plot which could be a trace bookmark, a trace record number or a time.
<range>	Trace record range or time range displayed in the plot.
<format>	Refer to **“Keywords for <format>”**, page 201.
<option>	Refer to **“General Options”**, page 202 for a description of the general options. Refer to **“Draw Options”**, page 203 for a description of t*<draw_options>.*

The example below shows a temperature measurement recorded by a logic analyzer. The **Trace.DRAW.channel** command is used to show the temperature profile.





See also

■ <trace>.DRAW

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.DRAW.Data** [*<start>* | *<range>*] [*<hscale>*] *<vscale>* *<v_offset>* {[**%***<format>*] *<data_address>* | *<data_range>*} [**/***<options>*]
<format>:	**Decimal.** [*<width>*] **DecimalU.** [*<width>*] **Hex.** [*<width>*] **HexS.** [*<width>*] **OCTal.** [*<width>*] **Float.** [**Ieee** | **IeeeDbl** | **IeeeeXt** | **IeeeMFFP** | …] **Byte** | **Word** | **Long** | **Quad** | **TByte** | **PByte** | **HByte** | **SByte**
<width>:	**DEFault** | **Byte** | **Word** | **Long** | **Quad** | **TByte** | **PByte** | **HByte** | **SByte**
<option>:	*<draw_options>* | **FILE** | **BusTrace** | **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** | **FIRST** *<address>* | **Filter** *<filter_items>* | **Track** | **ZoomTrack**
<draw_ option>:	**Points** | **Vector** | **MarkedVector** | **Steps** | **Impulses** | **MinMax** | **LOG** | **Color**

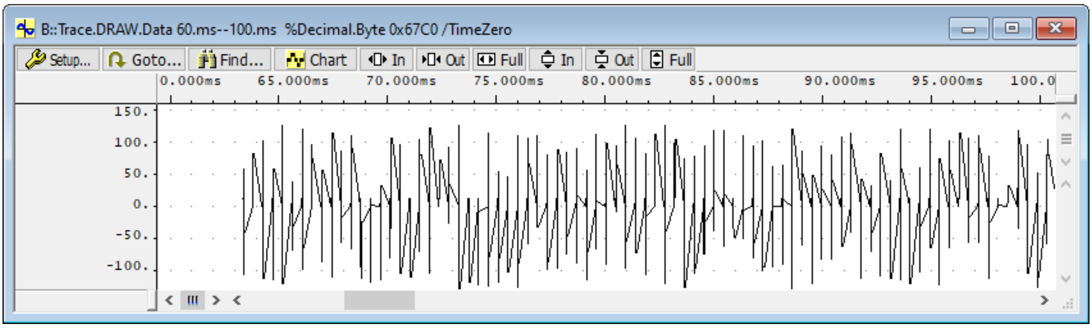
Plots one or more data values. An introduction to the usage of the **Trace.DRAW.Data** command is provided in **“Application Note for Trace.DRAW”** (app_trace_draw.pdf).

<start>	Start point of the plot which could be a trace bookmark, a trace record number or a time.
<range>	Trace record range or time range displayed in the plot.
<hscale>	time scale of the x-axis e.g. 100.us
<vscale>	Units per pixel of y-axis (floating point). E.g. a signal has a max. height of 50 units shall be visualized window that has a height of 400 pixels: 50 units divided by 400 pixels = 0.125 By default the scale factor is set so that the window displays the complete possible value range for the selected variable.
<v_offset>	Offset of y-axis (floating point). Default: **0.0**.
<format>	Refer to **“Keywords for <format>”**, page 201.
<data_address>	The content at the specified data address(es) is displayed graphically. The access width (Byte, Word, ...) has to be specified within the format.

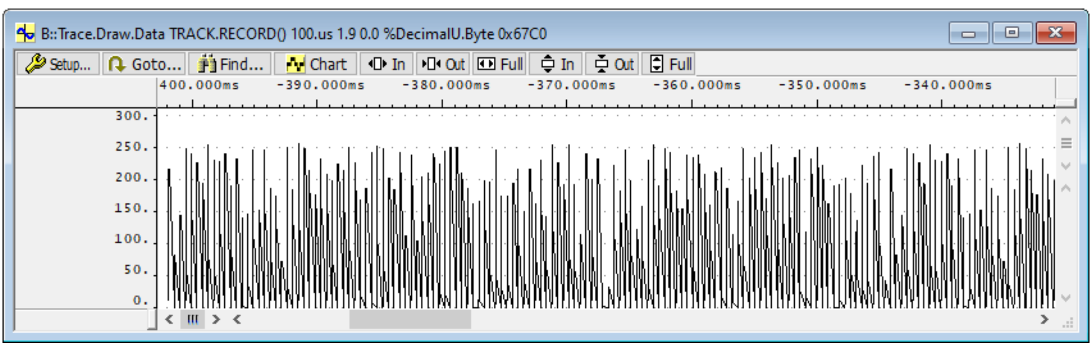
<data_range>	If no access width is specified, the access width is determined by the size of the *<data_range>.*
<option>	Refer to **"General Options"**, page 202 for a description general options. Refer to **"Draw Options"**, page 203 for a description of *<draw_ options>.*

Examples:

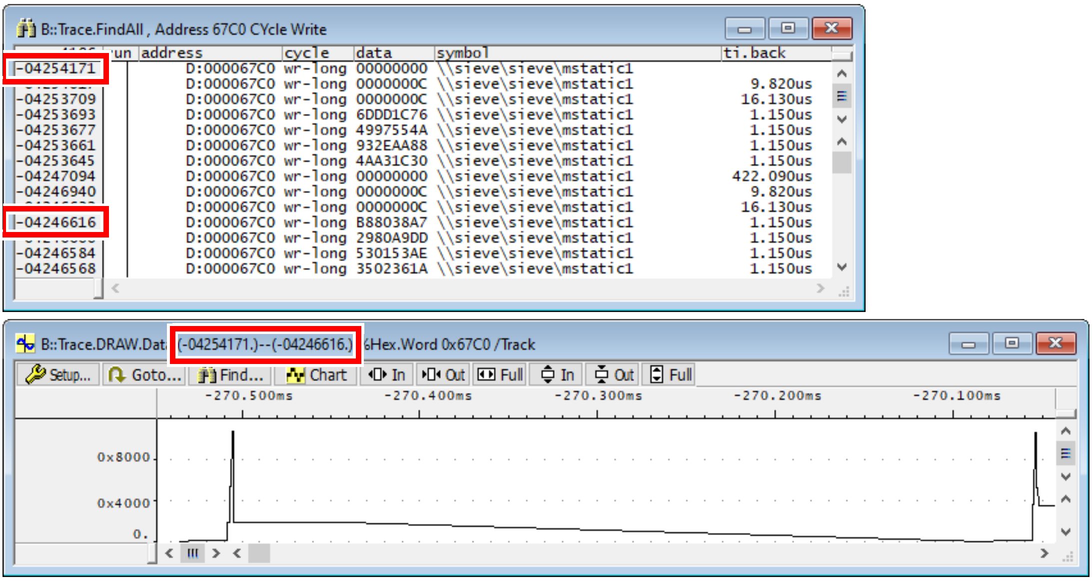
```
; display 8-bit value at address 0x67C0
; restrict the display to the time range 60.ms--100.ms
; the option /TimeZero is used to display the trace as true time display
; relative to zero
Trace.DRAW.Data 60.ms--100.ms  %Decimal.Byte 0x67C0 /TimeZero
```




```
; Display 8-bit unsigned value at address 0x67C0 starting at its first
; occurrence in the trace. Use horizontal time scale 100.us, vertical
; scale factor 1.9 and vertical offset 0x0

; find address 0x67C0 in the trace.
Trace.Find Address 0x67C0
; FOUND() returns TRUE if the address is found. TRACK.RECORD() returns
; the trace record number containing the found item
IF FOUND()
  Trace.Draw.Data TRACK.RECORD() 100.us 1.9 0.0 %DecimalU.Byte 0x67C0
```




```
; Plot graph for the specified record range.
Trace.DRAW.Data (-04254171.)--(-04246616.) %Hex.Word 0x67C0 /Track
```



See also

■ <trace>.DRAW

▲ 'Release Information' in 'Legacy Release History'

Format 1:	*<trace>*.**DRAW.Var** **%**[*<format>*] {*<var>*} [**/***<options>*]
Format 2:	*<trace>*.**DRAW.Var** [*<start>* | *<range>*] [*<hscale>*] *<vscale>* *<v_offset>* [**%***<format>*] {*<var>*} [**/***<options>*]
<format>:	**DEFault** | **STandDard** | **Decimal** | **Hex**
<option>:	*<draw_options>* | **FILE** | **BusTrace** | **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** | **FIRST** *<address>* | **Filter** *<filter_items>* | **Track** | **ZoomTrack**
<draw_ option>:	**Points** | **Vector** | **MarkedVector** | **Steps** | **Impulses** | **MinMax** | **LOG**

Plots the value changes of one or more variables against time, based on the recorded trace information. An introduction to the usage of the **Trace.DRAW.Var** command is provided in **"Application Note for Trace.DRAW"** (app_trace_draw.pdf).

<start>	Start point of the plot which could be a trace bookmark, a trace record number or a time.
<range>	Trace record range or time range displayed in the plot.
<hscale>	time scale of the x-axis e.g. 100.us
<vscale>	Units per pixel of y-axis (floating point). E.g. a signal has a max. height of 50 units shall be visualized window that has a height of 400 pixels: 50 units divided by 400 pixels = 0.125 By default the scale factor is set so that the window displays the complete possible value range for the selected variable.
<v_offset>	Offset of y-axis (floating point). Default: **0.0**.
<format>	Refer to **"Keywords for <format>"**, page 201.
<data_address>	The content at the specified data address(es) is displayed graphically. The access width (Byte, Word, ...) has to be specified within the format.

<data_range> If no access width is specified, the access width is determined by the size of the *<data_range>.*

<option> Refer to **“General Options”**, page 202 for a description general options. Refer to **“Draw Options”**, page 203 for a description of *<draw_options>.*

Examples:

```
; plot value of a single variable
Trace.DRAW.Var %DEFault mstatic1

; plot values of two variables
; colors are assigned by TRACE32
Trace.DRAW.Var %DEFault mstatic1 mstatic2

; plot values of three variables
; colors are assigned by TRACE32
; <display_option> Steps
Trace.DRAW.Var %DEFault mstatic1 fstatic fstatic2 /Steps
```

```
; plot values of variable vchar for specified <record_range>
Trace.DRAW.Var (-30000.)--(-29000.) %DEFault vchar
```

See also

- <trace>.DRAW

▲ ’Release Information’ in ’Legacy Release History’
▲ ’Filter and Trigger - Single-Core and AMP’ in ’Training AURIX Tracing’

<trace>.EXPORT — Export trace data for processing in other applications

Using the **<trace>.EXPORT** command group, you can export trace data for processing *in other applications*. Various export file formats are available, including ASCII, binary, PGT, VERILOG, etc.

NOTE:	The various export formats are primarily designed for import into other applications. Trace data exported with the **<trace>.EXPORT.*** commands ***can only be imported back*** into TRACE32 if you inform the debugger about all the trace-relevant circumstances. We recommend the following approach if you want to view and analyze recorded trace data in a subsequent TRACE32 session: 1. Save the trace data to file using **<trace>.SAVE**. 2. To load this file back into TRACE32, use **<trace>.LOAD**.

See also

- <trace>.SAVE
- <trace>.EXPORT.ARTI
- <trace>.EXPORT.ARTIAP
- <trace>.EXPORT.Ascii
- <trace>.EXPORT.Bin
- <trace>.EXPORT.BRANCHFLOW
- <trace>.EXPORT.CSVFunc
- <trace>.EXPORT.cycles
- <trace>.EXPORT.Func
- <trace>.EXPORT.MDF
- <trace>.EXPORT.MTV
- <trace>.EXPORT.TASK
- <trace>.EXPORT.TASKEVENTS
- <trace>.EXPORT.TracePort
- <trace>.EXPORT.VCD
- <trace>.EXPORT.VERILOG
- <trace>.EXPORT.VHDL
- LA.IMPORT
- RunTime
- RunTime.state

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**EXPORT.ARTI** *<file>* [*<trace_area>*] [**/***<options>*]
<option>:	**TRaceRecord** **CORE** **ENHanced**

Exports the trace contents to an ARTI file for AUTOSAR Classic Platform. White spaces are used as delimiters.

<file>	The default extension of the file name is ***.csv**
TRaceRecord	ARTI data are exported with trace record numbers.
CORE	Only the trace information of the specified core is exported.
ENHanced	For enhanced ARTI format.
TimeZero	Exports the timestamp relative to **ZERO** (TRACE32 has Global zero point **time.zero**, **B::ZERO**). Without this option, the exported timestamp starts with zero at the first event.

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles
- TASK.Create.RUNNABLE
- TASK.List.RUNNABLES
- ▲ 'Export' in 'Application Note Profiling on AUTOSAR CP with ARTI'

[build 141966 - DVD 02/2022]

Format:	**<trace>.EXPORT.ARTIAP** *<file>* [*<trace_area>*] [**/***<options>*]
<option>:	**TRaceRecord** **CORE**

Exports the trace contents for AUTOSAR Adaptive Platform. White spaces are used as delimiters.

<trace>	Currently only support for **SystemTrace**.
<file>	The default extension of the file name is ***.csv**
TRaceRecord	ARTI data are exported with trace record numbers.
CORE	Only the trace information of the specified core is exported.

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

Format:	*<trace>***.EXPORT.Ascii** *<file>* [*<record_range>*] [*<items>* …] [**/***<options>*]
<option>:	**FILE** | **BusTrace** **FILTER** **ShowRecord** **Append**

Exports the trace contents to an ASCII file. White spaces are used as delimiters.

<file>	The default extension of the file name is ***.ad**.
FILE	Exports the trace contents loaded with the command **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
FILTER	Exports only records matching the filter. For an example, see below.
ShowRecord	Includes the trace record numbers in the export file.
Append	Appends trace contents at the end of the file.
<option>	For a description of the other *<options>*, see **<trace>.EXPORT.flow**.

Example:

```
Trace.EXPORT.Ascii ~~\myfile.ad (-120000.)--(-1.) /ShowRecord \
                                  /FILTER ADDRESS Var.RANGE(sieve)
```

The backslash **** is used as a line continuation character. No white space permitted after the backslash.

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

Format:	*<trace>***.EXPORT.Bin** *<file>* [*<record_range>*] [*<items>* …] [**/***<options>*]
<option>:	**FILE** | **BusTrace** | **NoDummy** | **NoHeader** | **NoTimeStamps** | **NoFetch**

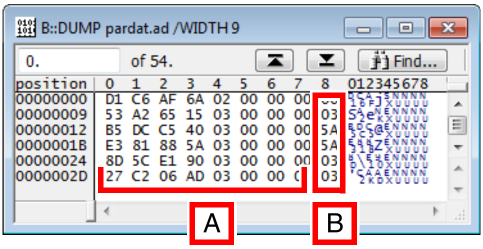
Exports the trace contents to a file in binary format. This command is used to export logic analyzer (PowerProbe, Integrator, IProbe) recordings. The data is stored in little endian format.

The file starts with a text header describing item names and byte size of each item. Each record begins with an 8-byte timestamp (1 ns per tick), followed by the selected items in the order as given in the command. Each item has a minimum width of 1 byte (max. 8 byte). The following options are available:

FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
NoDummy	Exclude records which do not hold flow information (do not use when exporting logic analyzer data).
NoHeader	The resulting file does not contain a header.
NoTimeStamps	The records do not contain the 8 byte timestamp.
NoFetch	Exclude control cycles from export.

Example: This script export data from a parallel port recorded with the IProbe.

```
;define the data word of the port, connected to signals ip.00...ip.07
NAME.WORD w.PARPORT ip.00 ip.01 ip.02 ip.03 ip.04 ip.05 ip.06 ip.07

;export analyzer data
IProbe.EXPORT.Bin pardat.ad W.PARPORT /NoHeader

;show resulting file: one record has 9 byte (w.PARPORT has 1 bytes)
DUMP pardat.ad /WIDTH 9
```



A Timestamp **B** Data of W.PARPORT

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

Format:	*<trace>***.EXPORT.BRANCHFLOW** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**FILE** | **BusTrace** | **TRaceRecord** | **NOINNER** | **NOSYMBOL** | **CALLer**

Exports the branch events from the trace data.

FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
TRaceRecord	Branch events are exported with trace record numbers.
NOINNER	Only branch events that jump to the current symbol are exported. The internal branch is not exported.
NOSYMBOL	Branch events are exported with addresses instead of symbols.
CALLer	Branch events are exported with caller events.

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

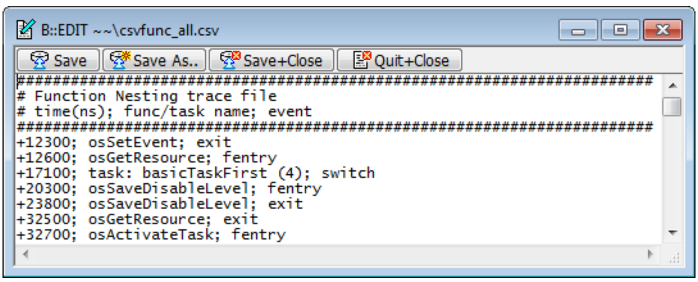
Format:	*<trace>*.**EXPORT.CSVFunc** *<file>* [*<trace_area>*]
<trace_area>:	*<string>* *<range>* *<value>* *<time_range>*

Exports the function nesting of the recorded trace data to a CSV file for processing by an external tool.

<file>	The default extension of the file name is ***.csv**.

Example:

```
;export the entire function nesting
Analyzer.EXPORT.CSVFunc ~~\csvfunc_all.csv

EDIT ~~\csvfunc_all.csv
```



See also

■ <trace>.EXPORT ■ <trace>.EXPORT.cycles

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**EXPORT.cycles** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**FILE** | **BusTrace** | **ZIP** | **NoDummy** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only)

Exports the trace contents for postprocessing by an external analysis tool.

The trace contents can only be exported when the trace is in **OFF** or **break** state. Please refer to the **Trace.state** command for more information.

The default export format is binary. A description of the binary format is given at the end of this command description.

<file>	The default extension of the file name is ***.ad**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
ZIP	File is compressed with the gzip archive format.
NoDummy	Exclude records which do not hold flow information (do not use when exporting logic analyzer data).

In the case of an SMP system, the following options are provided:

MergeCORE (default)	The trace information for all cores is exported.
SplitCORE	Same as MergeCORE.
JoinCORE	Same as MergeCORE.
CORE *<number>*	Only the trace information for the specified core is exported.

When an exported file contains a file header (not the case e.g. for **/ByteStream**, **/CoreByteStream**, ...) it has the following format:

Byte Nr.	Meaning
0..31	Export file header string (“trace32 analyzer export data“ 0x1a 0x00)
32	Reserved (set to zero for IMPORT)
33	CPU code
34	Timestamp available flag
35	Prestore mode flag
36	Trigger unit available flag
37	Port analyzer available/mode flag
38	Analyzer type
39	Reserved
40	Length of one record in bytes (0x20)
41..43	Reserved
44..47	Number of records in file (if record number can exceed 32 bits, e.g. Trace.Mode.STREAM, calculate number of records based on file size)
48..51	Record number of last recorded record
52..55	Reference record number
56..63	Reserved

Byte Nr.	Meaning
0..3	Cycle information flags: Bit 0: data cycle Bit 1: program cycle Bit 6: write cycle Bit 8: Power Architecture MPC5XXX: read/write cycle of peripheral NEXUS bus master Bit 21: FLOW ERROR Bit 25: **FIFO OVERFLOW** Bit 31: OWNERSHIP Cycle

Byte Nr.	Meaning
4	Data byte enable mask Bit 0: Byte 0 valid Bit 1: Byte 1 valid …
5	CPU specific information SH2A I-bus marker (bit meaning is device specific): Bit 0: iadma bus Bit 1: idma bus Bit 2: icpu1 bus Bit 3: icpu2 bus ARM Bustrace: Bit 0: EXEC signal (relevant only when **SYStem.Option.EXEC** is set to ON) ARM Flowtrace (ETM/PTM): Bit 1: Thumb Mode Bit 2: ARM Mode Bit 3: AArch64 Mode Bit 5: not executed Bit 6: executed
6	Reserved
7	Core number (on SMP targets)
8..15	Address (64 bits)
16..23	Data (64 bits)
24..31	Timestamp (time relative to ZERO in ns)

See also

- <trace>.EXPORT
- <trace>.EXPORT.ARTI
- <trace>.EXPORT.ARTIAP
- <trace>.EXPORT.Ascii
- <trace>.EXPORT.Bin
- <trace>.EXPORT.BRANCHFLOW
- <trace>.EXPORT.CSVFunc
- <trace>.EXPORT.Func
- <trace>.EXPORT.MDF
- <trace>.EXPORT.MTV
- <trace>.EXPORT.TASK
- <trace>.EXPORT.TASKEVENTS
- <trace>.EXPORT.TracePort
- <trace>.EXPORT.VCD
- <trace>.EXPORT.VERILOG
- <trace>.EXPORT.VHDL

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**EXPORT.Func** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**FILE** | **BusTrace** | **ZIP**

Exports the function nesting from the trace contents to a binary file.

Exported function nestings contain the function entries and exits as well as task switches with task entries and exits. Function nestings are displayed in the **<trace>.ListNesting** window.

<file>	The default extension of the file name is ***.ad**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
ZIP	File is compressed with the gzip archive format.

Example:

```
Analyzer.EXPORT.Func ~~~\trace.ad (-131072.)--(-100000.)
```

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

Format:	*<trace>***.EXPORT.MDF** *<file>* [*<trace_area>*] [**/***<options>*]
<option>:	**STanDard** **ENHanced** **ZIP**

Exports the trace contents to an MDF file as specified by the “ASAM Run-Time Interface Base Standard” (ASAM ARTI BS).

<file>	The default extension of the file name is ***.mf4**
STanDard	Default. Uses the **standard** task state machine, as specified in the ASAM standard.
ENHanced	Uses the **enhanced** task state machine, as specified in the ASAM standard.
ZIP	File is compressed with the gzip archive format.

See also

■ <trace>.EXPORT ■ <trace>.EXPORT.cycles

▲ ’Export’ in ’Application Note Profiling on AUTOSAR CP with ARTI’

TriCore, GTM, C166

Format:	*<trace>***.EXPORT.MTV** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**FILE** | **BusTrace** | **NoDummy**

Exports a trace recording in the MCDS Trace Viewer format.

<file>	The default extension of the file name is ***.mcds**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
NoDummy	Exclude records which do not hold flow information (do not use when exporting logic analyzer data).

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

Format:	*<trace>***.EXPORT.TASK** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**FILE** | **BusTrace** | **ZIP**

Exports task switching information from the trace contents to a binary file.

<file>	The default extension of the file name is ***.ad**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
ZIP	File is compressed with the gzip archive format.

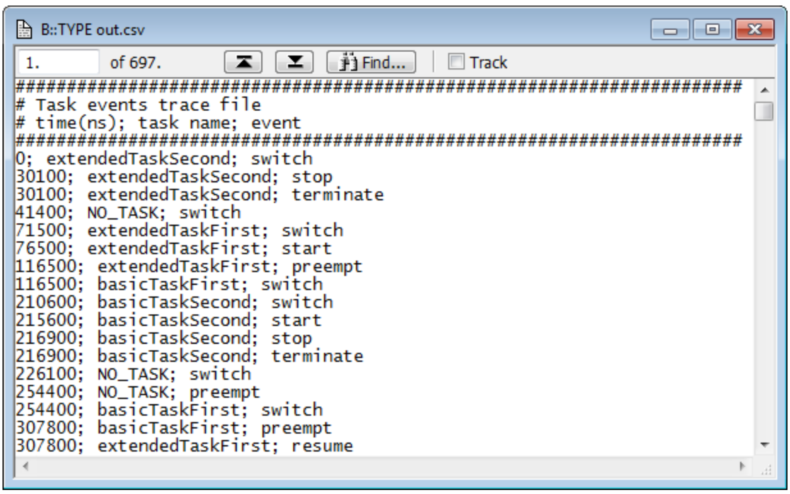
See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

Format:	*<trace>*.**EXPORT.TASKEVENTS** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**TRaceRecord** | **NOSTATEDATA** | **NOSTATEFLOW** **CORE** *<number>*

Generates a CSV file that contains task event (state) information and time information.

<file>	The default extension of the file name is ***.csv**.
TRaceRecord	Trace information is exported with trace record numbers.
NOSTATEDATA	Data trace based Task event (state) information is not exported.
NOSTATEFLOW	Flow trace based Task event (state) information is not exported.
CORE *<number>*	Only the trace information of the specified core is exported.



For details refer to **"Trace Export for Third-Party Timing Tools"** (app_timing_tools.pdf).

On TriCore AURIX there's a solution available for the Vector AUTOSAR tools that uses an automated instrumentation to trace task states and runnables on all cores with minimum overhead. See *~~/demo/env/vector/rte_profiling*.

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles
- 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**EXPORT.TracePort** *<file>* [*<record_range>*] [**/***<options>*]
<option>:	**FILE** **ByteStream** | **CoreByteStream** | **TimedByteStream** | **TPStream** | **TimedCoreByteStream** | **NibbleStream**

Exports the recorded trace data in a low-level binary format. Available options depend on the used processor architecture and trace port.

<file>	The default extension of the file name is ***.bin**.
ByteStream	Exports the byte stream broadcast by the ETM (same as **TP** column if command **Trace.List TP DEFault** is used).
CoreByteStream	Similar to the option **ByteStream**, but strips away synchronisation patterns (continuous mode) and trace source identifiers (e.g. in case of multicore systems). The exported data is that shown in the **TPC** column in the command **Trace.List TPC DEFault.** By default, the data corresponding to the currently active core is exported (selected by the **CORE** command), but this can be overridden by the **/CORE** *<number>* option.
TimedByteStream	Exports the byte stream broadcast by the ETM together with the **Time.Zero** timestamp information. For a description of the file format, see below.
TPStream	Power Architecture only. Exports NEXUS packets received through Aurora interface.
TimedCoreByteStream	Exports the unwrapped byte stream broadcast by the ETM together with the **Time.Zero** timestamp information. This format also supports multiple cores in SMP configuration.
NibbleStream	Exports just pure STP data, excluding non-STP headers (STP = System Trace Protocol).

File Format produced by the option TimedByteStream

The **TimedByteStream** format consists of two-byte records; possible formats are:

- 0y0xxxxxxx *<tracedata_byte>*
    - xxxxxxx: Time relative to previous records (in nanoseconds).
- 0y10xxxxxx 0yxxxxxxxx
    - xxxxxx: Time relative to previous record (bits 7 to 12).
    - xxxxxxxx: Upper bits (bits 13 to 20).
- 0y11000xxx 0yxxxxxxxx
    - xxx: Selects which part of the absolute time is transferred.
    - xxxxxxxx: Byte of absolute timestamp.
- 0y11001000 0yxxxxxxxx
    - xxxxxxxx: Selects to which core the following data belongs (only in **CoreByteStream** with SMP).

See also

■ <trace>.EXPORT ■ <trace>.EXPORT.cycles

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.EXPORT.VCD** *<file>* [*<record_range>*] [*<items>* …] [**/***<option>*]
<option>:	**FILE** | **NoDummy** | **BusTrace**

Exports the trace contents collected by the TRACE32 logic analyzers PowerProbe and PowerIntegrator to a file in VCD format.

<file>	The default extension of the file name is ***.vcd**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
NoDummy	Exclude records which do not hold flow information (do not use when exporting logic analyzer data).

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

<trace>.EXPORT.VERILOG

Format:	*<trace>*.**EXPORT.VERILOG** *<file>* [*<record_range>*] [*<items>* …] [**/***<option>*]
<option>:	**FILE** | **NoDummy** | **BusTrace**

Exports the trace contents collected by the TRACE32 logic analyzers PowerProbe and PowerIntegrator to a file in VERILOG format.

<file>	The default extension of the file name is ***.v**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
NoDummy	Exclude records which do not hold flow information (do not use when exporting logic analyzer data).

See also

- <trace>.EXPORT
- <trace>.EXPORT.cycles

<trace>.EXPORT.VHDL

Export trace data in VHDL format

Format:	*<trace>*.**EXPORT.VHDL** *<file>* [*<record_range>*] [*<items>* …] [**/***<option>*]
<option>:	**FILE** | **NoDummy** | **BusTrace**

Exports the trace contents collected by the TRACE32 logic analyzers PowerProbe and PowerIntegrator to a file in VHDL format.

<file>	The default extension of the file name is ***.vhd**.
FILE	Exports the trace contents loaded with **<trace>.FILE**.
BusTrace	The trace works as a bus trace. This option is usually not required.
NoDummy	Exclude records which do not hold flow information (do not use when exporting logic analyzer data).

See also

■ <trace>.EXPORT ■ <trace>.EXPORT.cycles

<trace>.ExtractCODE

Extract code from trace

Format:	*<trace>*.**ExtractCODE** [*<access class>*] [/*<option>*]
<option>]	**FILE** **BusTrace** | **FlowTrace**

Extracts code from trace and writes it to the memory.

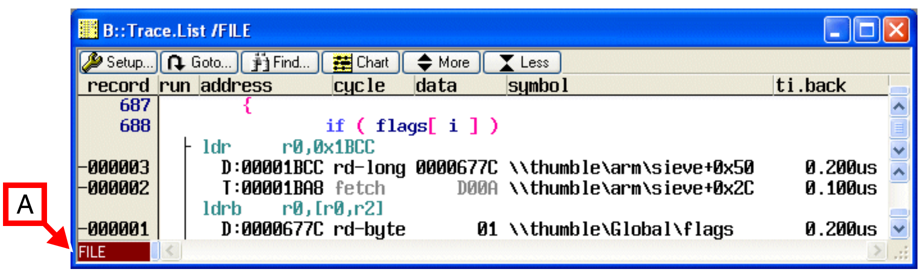
Format:	*<trace>*.**FILE** *<file>* [**/Config**]

Loads trace data from a file into a dedicated **file trace buffer** on the host. Typically this feature is used to analyze data in a simulator or to compare different recordings.

<file>	The default extension of the file name is ***.ad**.
Config	Restore analyzer and **NAME** settings contained in *<file>*. Only applicable for **Trace.METHOD Probe** and **Trace.METHOD Integrator**.

Example: To use the *file trace buffer* as source for trace-related commands, the commands need to be invoked with the additional parameter **/FILE**

```
Trace.FILE myfile.ad
Trace.List /FILE
```



A Windows working on trace contents loaded with the **Trace.FILE** command are marked with a red label **FILE** in the bottom-left corner:

Trace-related commands without the parameter **/FILE** keep operating on the trace data stored in the “normal” trace buffer which is filled when recording data using the analyzer hardware (e.g. PowerTrace, PowerProbe, PowerIntegrator).

Using the *file trace buffer* and the “normal” trace buffer concurrently allows to compare trace data stored in a file from a previous recording with recently recorded data as shown in the following example:

```
Trace.FILE test4          ; load trace contents from test4.ad

Trace.List /FILE          ; display loaded trace contents

Trace.Chart.sYmbol /FILE  ; works on loaded trace data

; compare the recently recorded trace with the trace contents loaded
; from test4.ad regarding to the addresses
Trace.ComPare , Address /FILE
```

NOTE:	In addition to **Trace.FILE** there is a command **Trace.LOAD** for loading trace data from a file into the “normal” trace buffer. Therefore data loaded with **Trace.LOAD** is treated as if it was recently recorded by the analyzer hardware. As a consequence all standard trace commands automatically work on the loaded via **Trace.LOAD** (without specifying additional parameters).

See also

- <trace>.LOAD
- IProbe.state
- RunTime
- RunTime.state

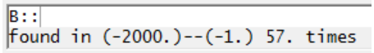
▲ ’Release Information’ in ’Legacy Release History’

Format:	*<trace>*.**Find** [*<record_number>* | *<record_range>*] [*<item>* …] [**/***<options>*]
<item>: (mostly with specified value)	**Address** *<address>* | *<address_range>* **Address.MATCH** *<address>* | *<address_range>* **FAddress** *<address>* | *<address_range>* **Data** *<value>* | *<value_range>* **Data !***<value>* | **!***<value_range>* **Data** *<value>*||*<value>*||*…* **Data** *<value_range>*||*<value_range>*||*…* **CYcle** *<cycle_type>* **Var** **GROUP** *<group_name>* **TIme.Back** *<time_range>* **TIme.Zero** *<time_range>* **TIme.AddressBack** *<time_range>* **TIme.AddressFore** *<time_range>*
<item>: (special events)	**EXCEPTION** | **INTERRUPT** | **TRAP** **FIFOFULL** | **FLOWERROR** | **TRACEENABLE** **CORE** | **IGNORE** | **Var** **OR** || (Address item only) **AT** *<offset>* (bus trace only) **CHANGE** *<item>*
<option>:	**Back** **FILE** **NoFind** **ALL** **FlowTrace** | **BusTrace** **TASK** *<task>*

Searches for matching items in the given range of trace records. The default search range is the complete trace. When the command is invoked without parameters, the previous search is repeated.

If the search finds a matching trace record, the PRACTICE function **FOUND()** will return **TRUE()**. If a matching trace record was found, **TRACK.RECORD()** returns the record number of the matching record.

Details about the **<trace>.Find** command as well as examples can be found in **"Application Note for Trace.Find"** (app_trace_find.pdf).

ALL	Searches for all occurrences and displays the result in the message line.  The number of occurrences can be returned with the function **FOUND.COUNT()**.
Back	Search backwards.
FILE	Takes trace memory contents loaded by **Trace.FILE**.
NoFind	Set up search, but don't search. Search can be done at a later point by using the **<trace>.Find** command without parameters.
FlowTrace	The trace works as a program flow trace. This option is usually not required.
BusTrace	The trace works as a bus trace. This option is usually not required.
TASK *<task>*	Filters search results for selected task.

See also

■ <trace>.FindAll ■ <trace>.FindChange ■ <trace>.FindProgram ■ IProbe.state
■ RunTime ■ RunTime.state ❏ FOUND() ❏ FOUND.COUNT()

▲ 'The Trace Find Dialog' in 'Application Note for Trace.Find'
▲ 'Introduction' in 'Application Note for Complex Trigger Language'
▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**FindAll** [*<record_number>* | *<record_range>*] *<items>* … [**/***<options>*]
<option>:	**Back** **FILE** **FlowTrace** | **BusTrace** **List** **Track** **TASK** *<task>*

Searches for and displays all entries matching the item specification. Without range, the complete trace memory is searched for matching entries. Details about the **<trace>.FindAll** command can be found in **"Application Note for Trace.Find"** (app_trace_find.pdf).

Back	The option **Back** reverses the direction of the search command.
BusTrace	The trace works as a bus trace. This option is usually not required.
FILE	Takes trace memory contents loaded by **Trace.FILE**.
FlowTrace	The trace works as a program flow trace. This option is usually not required.
List	Change the default display of the result.
TASK *<task>*	Filters search results for selected task.
Track	The cursor in the **<trace>.FindAll** window follows the cursor movement in other trace windows.

Example:

```
Trace.FindAll , sYmbol sieve /List TIme.Zero DEFault
```

See also

- ■ <trace>.Find
- ■ <trace>.FindChange
- ■ IProbe.state
- ■ RunTime
- ■ RunTime.state
- ❏ FOUND()
- ❏ FOUND.COUNT()

▲ 'Introduction' in 'Application Note for Complex Trigger Language'
▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**FindChange** [*<record_number>* | *<record_range>*] [{*<items>*]} [**/***<options>*]
<items>:	**OR** *<channels>* **AT** *<offset>*
<option>:	**Back** **FILE** **FlowTrace** | **BusTrace** **NoFind** **ALL**

Searches for entries in the given range where the specified items have new values. Without range the entry is searched within the complete trace memory. Without items the command searches for changes in program flow. This is useful to search for the end of a complex program loop, or in general to search for “something happens” in a traced program flow.

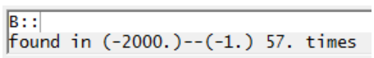
Back Reverses the direction of the search command.

FILE Takes trace memory contents loaded by **Trace.FILE**.

FlowTrace The trace works as a program flow trace. This option is usually not required.

BusTrace The trace works as a bus trace. This option is usually not required.

ALL Searches for all occurrences and displays the result in the message line.



The number of occurrences can be returned with the function **FOUND.COUNT()**.

NoFind Set up search, but don’t search. Search can be done at a later point by using the **<trace>.FindChange** command without parameters.

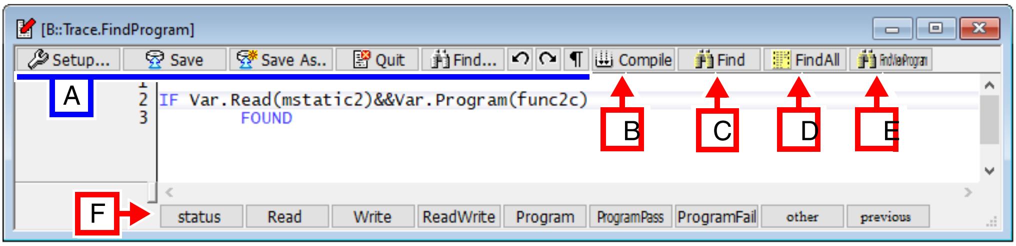
See also

- <trace>.Find
- <trace>.FindAll
- IProbe.state
- RunTime
- RunTime.state

Format:	**<trace>.FindProgram** [*<file>*]

Opens the **<trace>.FindProgram** editor window, where you can create an advanced trace search program using the Complex Trigger Language (CTL). The editor provides syntax highlighting, configurable auto-indentation and an online syntax check. The input is guided by softkeys.

Example: find all read accesses to the variable `mstatic1` in the trace within the function `func2c`



Buttons common to all TRACE32 editors:

A For button descriptions, see **EDIT.file**.

Buttons specific to this editor:

B **Compile** performs a syntax check and, if an error is found, displays an error message. If the file is error free, then the advanced trace search is programmed.

C Executes **Trace.Find** command.

D Opens a **Trace.FindAll** window with all occurence in the trace.

E Opens the **Trace.FindViewProgram** window showing the programming resources.

F Commands for advanced trace search programming.

See also

■ <trace>.Find ■ <trace>.FindReProgram ■ <trace>.FindViewProgram

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

<trace>.FindReProgram Activate advanced existing trace search program

Format: **<trace>.FindReProgram** [*<file>*]

Activates an existing advanced trace search program file.

See also

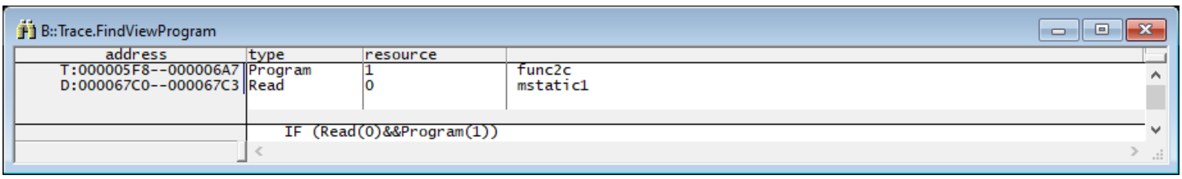
■ <trace>.FindProgram

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

<trace>.FindViewProgram State of advanced trace search programming

Format: **<trace>.FindViewProgram**

Opens a windows that shows the state of the advanced trace search programming.



See also

■ <trace>.FindProgram

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

<trace>.FLOWPROCESS   Process flowtrace

Format:	*<trace>*.**FLOWPROCESS** [*<address>*]

Processes **all trace data in the analyzer** and calculates the instruction flow for all of it. This is in contrast to **<trace>.FLOWSTART** which discards the processing results and thus indirectly causes a reprocessing of the limited set of trace data required to draw the currently open windows (reprocessing on demand).

The command is used mostly for diagnostic purposes.

<trace>.FLOWSTART   Restart flowtrace processing

Format:	*<trace>*.**FLOWSTART** [*<address>*]

Discards all results from previous decoding of instruction flow. This indirectly causes a reprocessing of the limited set of trace data required to draw the currently open windows (reprocessing on demand). Effectively the decoding of flow information is done again “from the start”.

The command is typically used when the memory contents at the time of decoding was wrong and the decoding is therefore incorrect (contains flow errors). The command is executed after providing a correct memory image (e.g. by activating chip selects) to re-initialize the flow processing.

The optional address parameter can be used to indicate the address of the first instruction executed by the processor. In this way the debugger can correctly decode code sequences even before the first sync message appears in the trace stream.

See also

❏ FOUND()

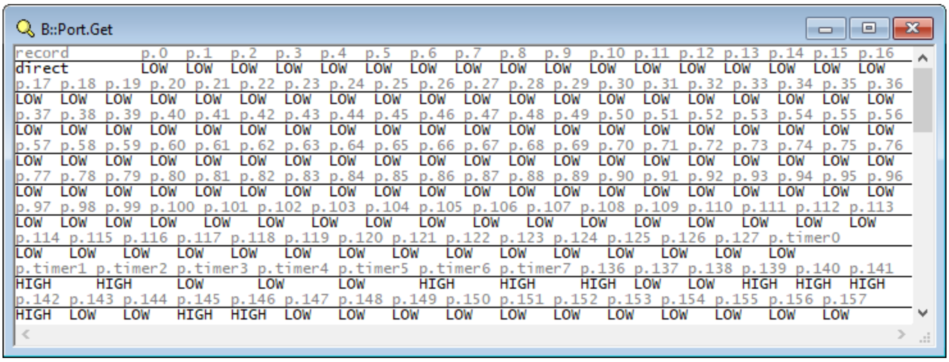
Format:	*<trace>*.**Get** [**%**<*format*>] [<*item*> …]
<*items*>:	**%**<*format*> **DEFault** | **ALL** | **CPU** | **LINE** | **PORTS** **Run** **CYcle** | **Data**[**.**<*subitem*> | **BDATA** | **List**[**.**<*subitem*>] **Address** | **BAddress** | **FAddress** | **sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** | **Var** **TIme**[**.**<*subitem*>] **FUNC** | **FUNCR** | **FUNCVar** | **IGNORE** **LeVel** | **MARK**[**.**<*marker*> | **FLAG**[**.**<*flag_index*>] **Trigger** | **Trigger.A** | **Trigger.B** **SPARE** <*special_lines*>
<*format*>:	**Ascii** | **BINary** | **Decimal** | **Hex** | **Signed** | **Unsigned** **HighLow** | **Timing** **TimeAuto** | **TimeFixed** **LEN** <*size*>
<*option*>:	**FILE** **Track** **FlowTrace** | **BusTrace** **Mark** <*item*>

Displays the current state of all input lines. The format of the channel definition is similar to the **<trace>.View** command. This command can be executed, while the port analyzer is running.

For valid channel names refer to the **Processor Architecture Manuals**.

Examples:

```
; Display the state of all port lines in hex and HIGH/LOW format.
Port.Get
```




```
; Display the state of port lines P2 in binary format, lines P3.0, P3.1
; and P3.2 in timing waveform, port lines P5 in decimal format, port
; lines P4 in hex format and port PX in ASCII format.Port.Get

Port.Get p.2 p.30 p.31 p.32 %Decimal p.5 %Hex p.4 %Ascii p.x
```

```
; Display the state of all port lines in timing waveform.
Port.Get ALL
```

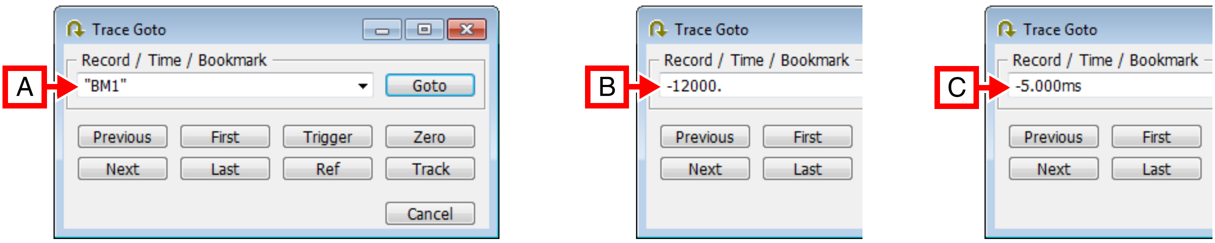
See also

■ IProbe.state

[Examples]

Format:	*<trace>*.**GOTO** "*<bookmark>*" | *<record_number>* | *<time>* [**/***<options>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** | **CORE** *<number>*

Goes to the specified trace record in a **Trace.*** window by moving the cursor to that trace record. Alternatively, click the **Goto** button in a **Trace.*** window, and enter a record number, a time index, or the name of a trace bookmark.



A To go to a trace *<bookmark>*, enclose the bookmark name in quotation marks.

B To go to a trace *<record_number>*, append a period (.). Mind the + or - sign of the record number.

C To go to a *<time>*, prepend a plus or minus sign and append the unit of measurement. To view the *<time>*, include the **TIme.ZERO** column in the **Trace.List** command, as shown in the example below.

BusTrace	The trace works as a bus trace. This option is usually not required.
FILE	Takes trace memory contents loaded by **Trace.FILE**.
FlowTrace	The trace works as a program flow trace. This option is usually not required.
CORE	The goto operation takes the specified core number into account. Only available for SMP multicore tracing.

Description of Buttons in the Trace Goto Dialog

Previous / Next	Go to the previous / next user-defined trace bookmark. Trace bookmarks are created with **Trace.BookMark**.
First / Last	Go to the first / last trace record.
Trigger	Go to the trigger record.
Ref	Go to the reference point, which has been set with the **Trace.REF** command. You can also set the reference point by right-clicking in a **Trace.*** window and pointing to **Set Ref** in the **Trace** popup menu.

Zero	Go to the zero reference point, which has been set with the **ZERO.offset** command. You can also set the zero reference point by right-clicking in a **Trace.*** window and pointing to **Set Zero** in the **Trace** popup menu.
Track	Go to the last tracked record.

Examples

The **Trace.List** window is always opened with the **Track** option. Thanks to the **Track** option, the subsequent **Trace.GOTO** command scrolls to the desired trace record in the **Trace.List** window.

Example 1: Go to a *<record_number>*.

```
;open the Trace.List window
Trace.List /Track

;go to this <record_number> in the Trace.List window
Trace.GOTO     -12000.
```

Example 2: Go to a trace *<bookmark>*.

```
;create a trace bookmark named 'BM1' for record -14000.
Trace.BookMark "BM1" -14000.

;open the Trace.List window
Trace.List /Track

;go to this <bookmark> in the Trace.List window
Trace.GOTO    "BM1"
```

Example 3: Go to a *<time>* index.

```
;          <first_column>  <other_columns>
Trace.List    TIme.ZERO       DEFault        /Track

;go to this <time> in the Trace.List window
Trace.GOTO  -5.000ms
```

See also

- ■ <trace>.BookMark
- ■ <trace>.TRACK
- ■ IProbe.state
- ■ RunTime.state
- ❏ Analyzer.RECORD.DATA()
- ❏ Analyzer.RECORDS()
- ■ <trace>.REF
- ■ BookMark
- ■ RunTime
- ❏ Analyzer.RECORD.ADDRESS()
- ❏ Analyzer.RECORD.OFFSET()
- ❏ Analyzer.REF()

<trace>.Init — Initialize trace

Format:	*<trace>*.**Init**

The contents of the trace memory/streaming file is erased. All user setups, like the trace mode or trace memory size, remain unchanged.

If the chip includes an onchip trigger unit, counters and trigger levels are cleared. The detailed behavior strongly depends on the onchip trigger unit.

The trace is in OFF state, after a **Trace.Init** was executed.

See also

- <trace>.Arm
- <trace>.AutoInit
- IProbe.state
- RunTime
- RunTime.state

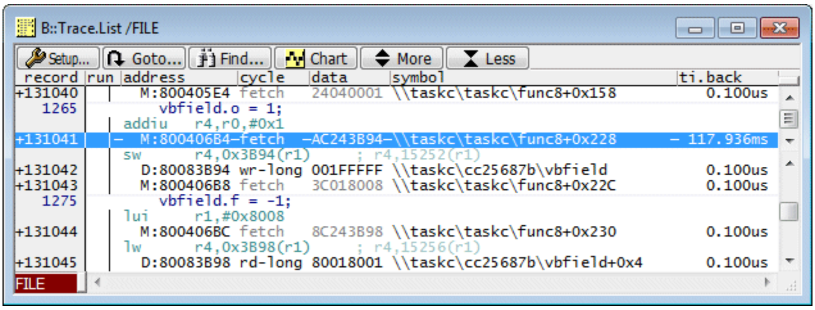
▲ 'Release Information' in 'Legacy Release History'

<trace>.JOINFILE — Concatenate several trace recordings

Format:	*<trace>*.**JOINFILE** *<file>* [*<records>*] [**/***<option>*]
<records>:	*<string>* | *<range>* | *<value>* | *<time_range>*
<option>:	**ZIP** | **NoCompress** | **Compress** | **QuickCompress** | **TIMEGAP**

Concatenates several trace recordings to increase the volume of trace information to be analyzed.

The reference point is automatically set to the start of the last added trace recording.



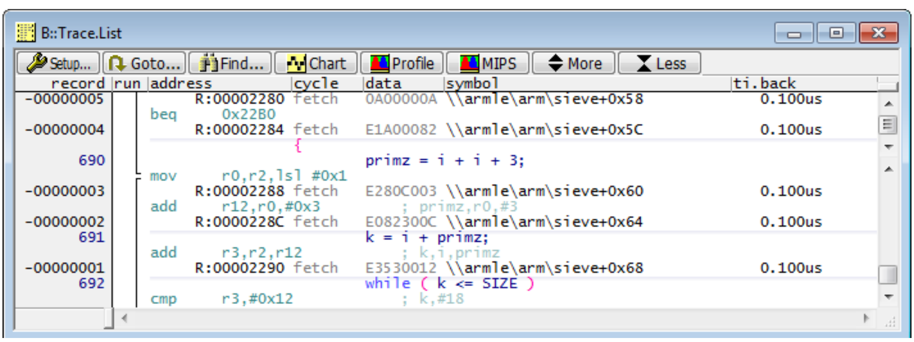
Time gaps between the trace recording result in a large **TIme.Back** time (see screenshot above). The option **TIMEGAP** *<time>* allows a seamless concatenation with regards to the timestamp.

```
Trace.SAVE my_joinfile.ad                        ; save current trace
                                                 ; contents to file

Trace.FILE my_joinfile.ad                        ; load trace contents
                                                 ; from file

;…                                               ; run program to fill the
                                                 ; trace

Trace.JOINFILE my_joinfile /TIMEGAP 0.1us        ; append current trace
                                                 ; contents to loaded
                                                 ; trace contents

;…                                               ; run program to fill the
                                                 ; trace

Trace.JOINFILE my_joinfile /TIMEGAP 0.1us        ; append current trace
                                                 ; contents

Trace.Chart.sYmbol /FILE                         ; display timing for
                                                 ; concatenated trace

;…

Trace.FILE                                       ; close loaded trace file
```

```
; use record numbers to specify the trace recording to be added
Trace.JOINFILE my_joinfile (4665.)--(5168.)

; use bookmarks to specify record range
Trace.JOINFILE my_joinfile "start" "end" /TIMEGAP 0.1us
```

Format:	*<trace>*.**List** [*<record>* | *<record_range>* | *<time>* | *<time_range>*] [*<items>* …] [**/***<options>*]
<option>:	**FILE** | **Track** | **FlowTrace** | **BusTrace** **NorthWestGravity** **CORE** *<number>* | **SplitCORE** | **TASK** *<task>* **Mark** *<item>* | **Raw** | **TimeZero**
<items>:	**%***<format>* **DEFault** | **ALL** | **CPU** | **LINE** | **PORT** **Run** **CYcle** | **Data**[**.***<subitem>* | **BDATA** | **List**[**.***<subitem>*] **Address** | **BAddress** | **FAddress** |**sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** **Var** | **VarName** | **VarValue** **TIme**[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*] **FUNC** | **FUNCR** | **FUNCVar** | **IGNORE** **LeVel** | **MARK**[**.***<marker>* | **FLAG**[**.***<flag_index>*] **Trigger** | **Trigger.A** | **Trigger.B** **SPARE** *<special_lines>* **FLOWERROR** | **FIFOFULL** (trace error diagnosis) **FAddress** | **FsYmbol** | **FCOUNT** | **FLen** (program flow diagnosis) **TP** | **TPC** | **TPINFO** (raw trace data and decoded packet) **TSINFO** (timestamp calculation diagnosis) **TP0** | **TP1** | **TP2** | … (trace port pins) **ARTIAPCore** | **ARTIAPEvent** | **ARTIAPData** | **ARTIAP**
<format>:	**Ascii** | **BINary** | **Decimal** | **Hex** | **Signed** | **Unsigned** **HighLow** | **Timing**| **TimeAuto** | **TimeFixed** | **LEN** *<size>*

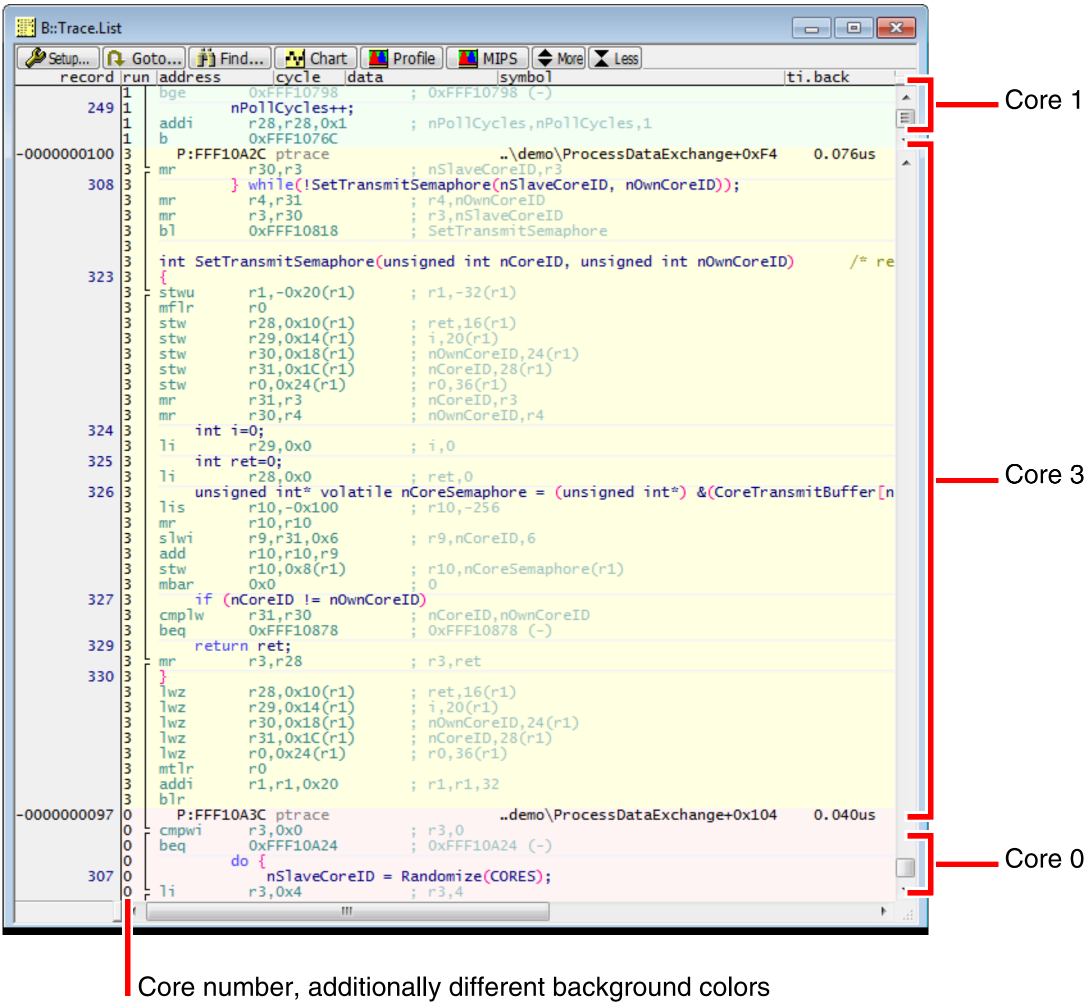
Opens a window showing the recorded trace data. For trace modes other than RTS (see **<trace>.Mode**), the trace contents can only be displayed if the trace is in **OFF** or **break** state. Please refer to the **<trace>.state** command for more information.



Please refer to your **Processor Architecture Manual** for target-specific information and options.

<record>	The recorded trace data is displayed starting at the selected record.
<record_range>	The recorded trace data is displayed for the selected range of trace records (e.g. (`-10000.)--(-2000.)`).
<time>	Defines which timestamp is centered on the x-axis when the window is opened. Timestamps at the beginning or end of the x-axis are not centered. **NOTE:** Only zero-time timestamps can be used as *<time>* parameters. You can display the zero-time timestamps in a **Trace** window by adding the **TIme.Zero** column to **Trace.List**.
<time_range>	Defines which timestamp is displayed on the left of the x-axis when the window is opened. NOTE: Only zero-time timestamps can be used as *<time_range>* parameters. You can display the zero-time timestamps in a **Trace** window by adding the **TIme.Zero** column to **Trace.List**.
<bookmark>	Defines which bookmark position is centered on the x-axis when the window is opened. Bookmark positions at the beginning or end of the x-axis are not centered. **NOTE**: You can only use the names of trace bookmarks, which are created with the **<trace>.BookMark** command.
<items>	The columns of the **<trace>.List** window can be defined using the *<items>*. The order of the columns in the window is according to the order of the *<item>* parameters given (with a few exceptions like the **run** column that always appears at the very left). Note that the default columns are hidden, when you manually specify the columns you want to display. The default columns can be included again in the user-defined column display using the option **DEFault**. Example: `Trace.List List.ADDRESS DEFault` For details on the available columns, see further down.

For SMP systems, each core is represented in the **Trace.List** window by a different background color.


B::Trace.List
Setup... Goto... Find... Chart Profile MIPS More Less
record run address cycle data symbol ti.back
1 bge 0xFFF10798 ; 0xFFF10798 (-)
249 1 nPollCycles++;
1 addi r28,r28,0x1 ; nPollCycles,nPollCycles,1
1 b 0xFFF1076C
-0000000100 3 P:FFF10A2C ptrace ..\demo\ProcessDataExchange+0xF4 0.076us
3 mr r30,r3 ; nSlaveCoreID,r3
308 3 } while(!SetTransmitSemaphore(nSlaveCoreID, nOwnCoreID));
3 mr r4,r31 ; r4,nOwnCoreID
3 mr r3,r30 ; r3,nSlaveCoreID
3 bl 0xFFF10818 ; SetTransmitSemaphore
3 int SetTransmitSemaphore(unsigned int nCoreID, unsigned int nOwnCoreID) /* re
323 3 {
3 stwu r1,-0x20(r1) ; r1,-32(r1)
3 mflr r0
3 stw r28,0x10(r1) ; ret,16(r1)
3 stw r29,0x14(r1) ; i,20(r1)
3 stw r30,0x18(r1) ; nOwnCoreID,24(r1)
3 stw r31,0x1C(r1) ; nCoreID,28(r1)
3 stw r0,0x24(r1) ; r0,36(r1)
3 mr r31,r3 ; nCoreID,r3
3 mr r30,r4 ; nOwnCoreID,r4
324 3 int i=0;
3 li r29,0x0 ; i,0
325 3 int ret=0;
3 li r28,0x0 ; ret,0
326 3 unsigned int* volatile nCoreSemaphore = (unsigned int*) &(CoreTransmitBuffer[n
3 lis r10,-0x100 ; r10,-256
3 mr r10,r10
3 slwi r9,r31,0x6 ; r9,nCoreID,6
3 add r10,r10,r9
3 stw r10,0x8(r1) ; r10,nCoreSemaphore(r1)
3 mbar 0x0 ; 0
327 3 if (nCoreID != nOwnCoreID)
3 cmplw r31,r30 ; nCoreID,nOwnCoreID
3 beq 0xFFF10878 ; 0xFFF10878 (-)
329 3 return ret;
3 mr r3,r28 ; r3,ret
330 3 }
3 lwz r28,0x10(r1) ; ret,16(r1)
3 lwz r29,0x14(r1) ; i,20(r1)
3 lwz r30,0x18(r1) ; nOwnCoreID,24(r1)
3 lwz r31,0x1C(r1) ; nCoreID,28(r1)
3 lwz r0,0x24(r1) ; r0,36(r1)
3 mtlr r0
3 addi r1,r1,0x20 ; r1,r1,32
3 blr
-0000000097 0 P:FFF10A3C ptrace ..demo\ProcessDataExchange+0x104 0.040us
0 cmpwi r3,0x0 ; r3,0
0 beq 0xFFF10A24 ; 0xFFF10A24 (-)
0 do {
307 0 nSlaveCoreID = Randomize(CORES);
0 li r3,0x4 ; r3,4
Core 1
Core 3
Core 0
Core number, additionally different background colors

Setup …	Open a **<trace>.state** window, to configure the trace.
Goto …	Open a **<trace>.GOTO** dialog box, to move the cursor to a specific record.
Find …	Open a **<trace>.Find** dialog box, to search for specific entries in the trace.
Chart	Display the program execution time at different symbols as a time chart. See the **<trace>.Chart.sYmbol** command.
Profile	Open a **<trace>.PROfileChart.sYmbol** window.
MIPS	Open a **MIPS.PROfileChart.sYmbol** window.
More/Less	Switch step-by-step from full display (all CPU cycles including dummies) to HLL display and vise versa.

If no parameters are specified, a predefined set of items will appear in the window. By selecting items, specific items can be displayed in any order defined by the user. It is possible to remove a selection from the list by appending the keyword**.OFF**. The display format of the entries can be changed by the **%**<*format*> options.

FILE	Displays trace memory contents loaded with **Trace.FILE**.
FlowTrace	The trace works as a program flow trace. This option is usually not required.
BusTrace	The trace works as a bus trace. This option is usually not required.
Track	Track the **<trace>.List** window with other trace list windows (tracking to record number or time possible).
Mark *<item>*	Bold print all cycles on a yellow background which contain the specified item.
NorthWestGravity	With **NorthWestGravity**: The record numbering in the top left corner stays fixed as you resize the **<trace>.List** window. Without **NorthWestGravity**: The record numbering scrolls as you resize the window.
Raw	Displays all channels as raw hexadecimal values (where applicable)
TASK *<task>*	Displays the trace recording of the specified task only.
TimeZero	Use timestamp of first entry in listing as global reference (item **TIme.Zero**).

```
Trace.FILE test1                ; load trace file
Trace.List /FILE                ; display trace listing, source for the
                                ; trace data is the loaded file


Trace.List /Mark Address sieve  ; mark all trace lines which contain the
                                ; address sieve
```

In the case of an SMP system, the following options are provided:

SplitCORE	Displays the trace recording of all cores side by side.
CORE *<number>*	Displays the trace recording of the specified core.

Ascii	Displays single bytes as ASCII characters
BINary	Displays single bytes in binary values
Decimal	Displays single bytes in decimal values
Hex	Displays single bytes in hex values
HighLow	Displays single bits as 'H' or 'L' character
LEN *<size>*	Specifies the width of non numeric fields (e.g. symbols)
Signed	Displays single bytes signed
TimeAuto	Displays time values in a floating display format (short)
TimeFixed	Displays time values in a fixed point format (long format)
Timing	Displays single bits as vertical timing
Unsigned	Displays single bytes unsigned

Examples:

```
; display trace listing, limit the symbol names to 20 characters
Trace.List Address CYcle Data.L %LEN 20. sYmbol TIme.Back

; display trace listing, show the external trigger input 0 as vertical
; timing
Trace.List %Timing TIme.ZERO DEFault
```

The following <items> define the columns shown in the <trace>.List windows

DEFault	Default trace display. The default trace display can be configured with the command **SETUP.ALIST**.
ALL	Select all available channels (superset of **DEFault**)
CPU	Set of channels describing the CPU state (similar to the original setting of **DEFault** but no source code display).
LINE	Set of channels which contains all CPU control lines.

Run	Gives various information about the execution of the current record. • GO: the first instruction that was executed by the CPU after starting program execution with **Go**. BRK Indicates that the program execution was stopped. • T : Indicates a trigger event. • f : Foreground program • b : Background program • ft : Trigger event occurred in the foreground program • bt: Trigger event occurred in the background program • 0,1,2,3 ... in SMP systems, the run column indicates the number of the core that executed the given code; additionally, the background color of the records changes to high-light the relevant core (light red, light green, ...).
Address	start address of each displayed **block of executed opcodes;** for displaying the address of each single opcode, use the channel `List.Address`.
sYmbol	Symbolic address with path and offset (as find item will search on all processor busses)
sYmbolN	Symbolic address without path but with offset
sYmbolInline	Inline symbol name with path.
sYmbolInlineN	Inline symbol name without path.
AAddress	Physical (absolute) CPU address
AAddress.0--31	Physical address bits A0..A31
PAddress	This column display the address of the instruction that was executed before a read or write access was performed.
PsYmbol	This column display the address of the instruction that was executed before a read or write access was performed.
FAddress	Flowtrace execution address (when flowtrace available)
FsYmbol	Symbolic flowtrace execution address
BAddress	Bus address, same like physical address, but also displayed when the bus is not transferring data

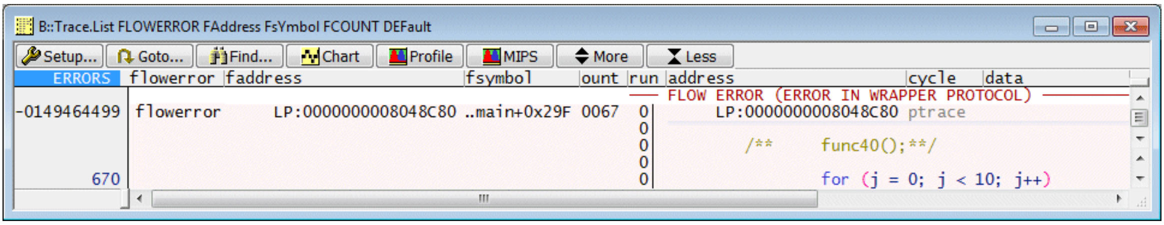
Var	Symbolic display of data accesses to HLL variables
VarName	Returns the names of the HLL variables.
VarValue	Returns the values of the HLL variables.
CYcle	Bus cycle
Data	CPU data full width
Data.B	CPU data single byte
Data.B0	CPU data lower byte
Data.W0	CPU data lower word
Data.T0	CPU data lower triple
Data.0..31	CPU data bit 0 to 31
Data.0--7	CPU data bits 0 to 7 as single bits (8 bit processor)
Data.0--15	CPU data bits 0 to 15 as single bits (16 bit processor)
Data.0--31	CPU data bits 0 to 31 as single bits (32 bit processor)
Data.sYmbol	Display the data value symbolically
BData	Like Data, but always displays the data even when the bus is idle
List.Address	Lists the address for each individual opcode (instead of the start address of blocks of executed opcodes)
List.Asm	Disassembled mnemonics
List.Mix	Disassembled mnemonics and HLL source
List.Hll	HLL source only, dequeueing based on disassembler
List.HllOnly	HLL source only no dequeueing
List.NoFetch	Suppresses the display of op-fetches
List.NoPFetch	Suppresses the display of prefetch cycles
List.NoCycle	Suppresses the display of more than one cycle between lines
List.Label	Label of disassembled mnemonic

List.Comment	Comments to disassembled mnemonics
List.Queue	Start address of disassembled mnemonic
List.TASK	Displays OS Awareness information (system-calls etc.)
List.Reorder	Reorders bus cycles logically (only some processors)
List.NoDummy	Suppresses the display of dummy cycles (where applicable)
List.Bondout	Display internal bondout information (where applicable)
List.TIme	Display time information in assembler or HLL lines
List.CTS	Display CTS information (Context Tracking System)
List.SOURCE-FILE	Display source file name for each line
TIme	Time marker (default Time.Fore)
TIme.Fore	Time marker, relative time to next record
TIme.Back	Time marker, relative time to previous record
TIme.Zero	Time marker, relative to global reference
TIme.REF	Time marker, relative to reference point
TIme.Trigger	Time marker, relative to trigger point
TIme.FUNC	Time spent in a function (*1)
TIme.FUNCEX	Time spent in calls (*1)
TIme.FUNCIN	Time spent in code of function (*1)
TIme.MARKAB	Time relative back to the last marker A
TIme.MARKAF	Time relative forward to the next marker A
TIme.MARKBB	Time relative back to the last marker B
TIme.MARKBF	Time relative forward to the next marker B
TIme.MARKCB	Time relative back to the last marker C
TIme.MARKCF	Time relative forward to the next marker C

TIme.MARKDB	Time relative back to the last marker D
TIme.MARKDF	Time relative forward to the next marker D
CLOCKS.Back	Number of clocks relative time to previous record
CLOCKS.Fore	Number of clocks relative time to next record
CLOCKS.Trigger	Number of clocks relative to trigger point
CLOCKS.REF	Number of clocks relative to reference point
CLOCKS.Zero	Number of clocks relative to global zero point
FUNC	Function nesting display (*1)
FUNCVar	Function nesting plus variables
FUNCR	Record number associated with this entry/exit point (*1)
IGNORE	Record ignored or used for performance/nesting analysis
LeVel	Trigger unit logical level
MARK.all	Display markers
MARK.A	Display marker A
FLAG.all	Flags of the trigger unit in a short form
FLAG.0	Flag 0 of the trigger unit
Trigger.0	External trigger bit 0
Trigger.0--7	External trigger input bit 0--7
SPARE	Displays an empty block
VarsYmbol	HLL display of accesses to variables including bitfields and symbols.
traceID	If a context ID or ownership packet is decoded and if it can not be assigned to a task or any other protocol-specific content such as service, intr etc. the cycle type traceID and the packet content is displayed.

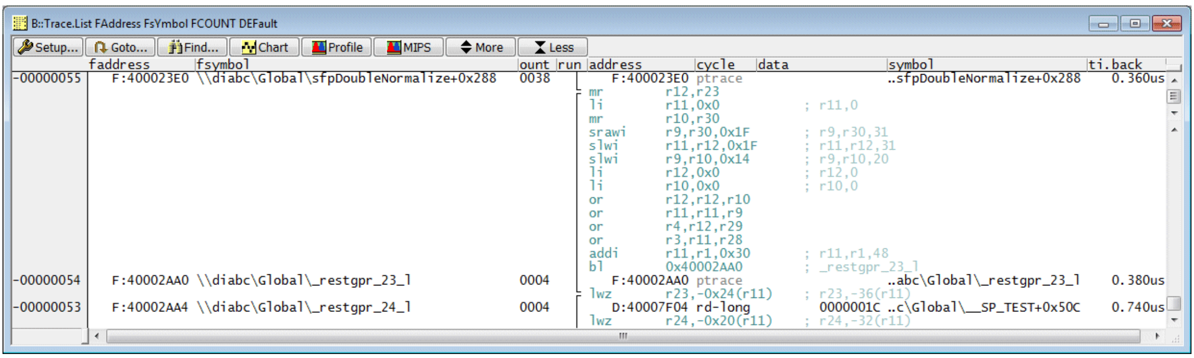
(1): The trace must be the same as for the command **<trace>.STATistic.Func**. The combination of the **FUNC** keyword with the **List.TASK** keyword makes the function nesting display task sensitive.

FLOW ERROR Diagnosis

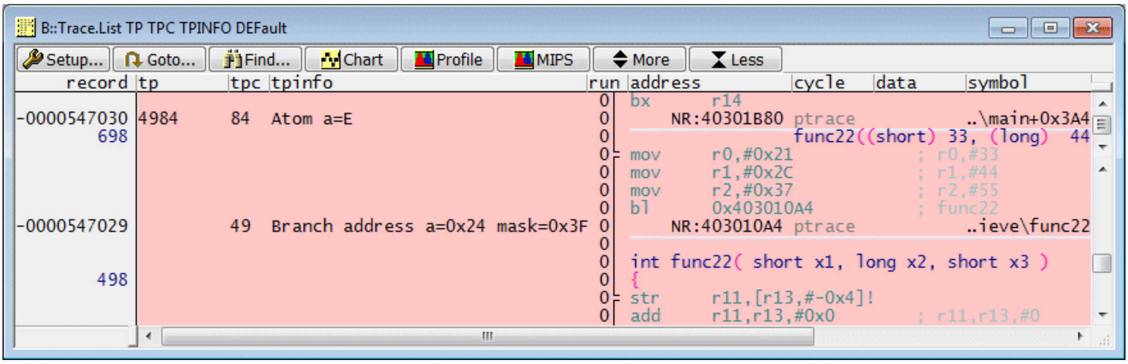


FLOWERROR	Display flow error column

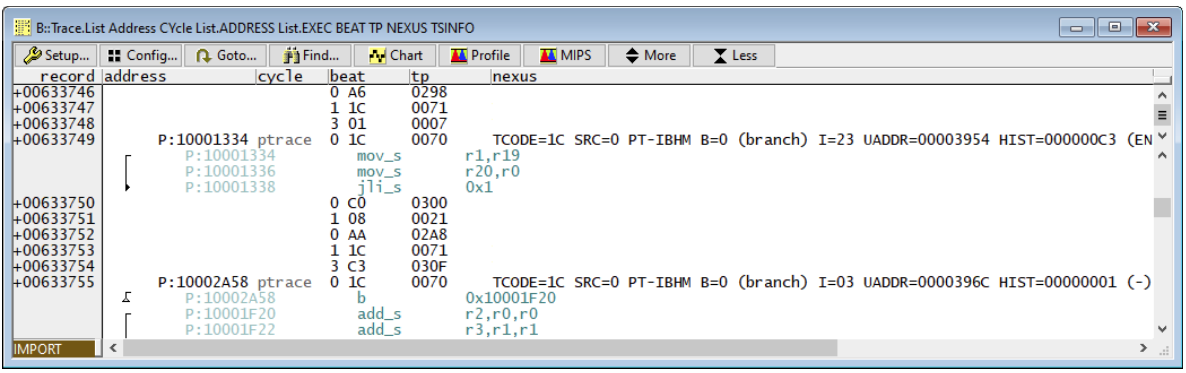
Flow Trace Decoding



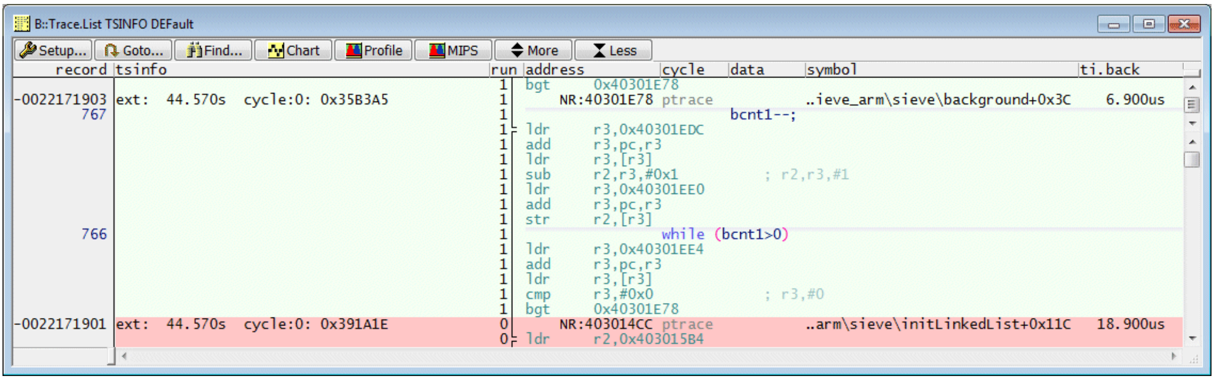
FAddress	To decompress the recorded trace information the program code starting at FAddress is read.
FsYmbol	Symbolic address of FAddress.
FCOUNT	To decompress the recorded trace information FCOUNT number of byte is read.
FLen	(deprecated)



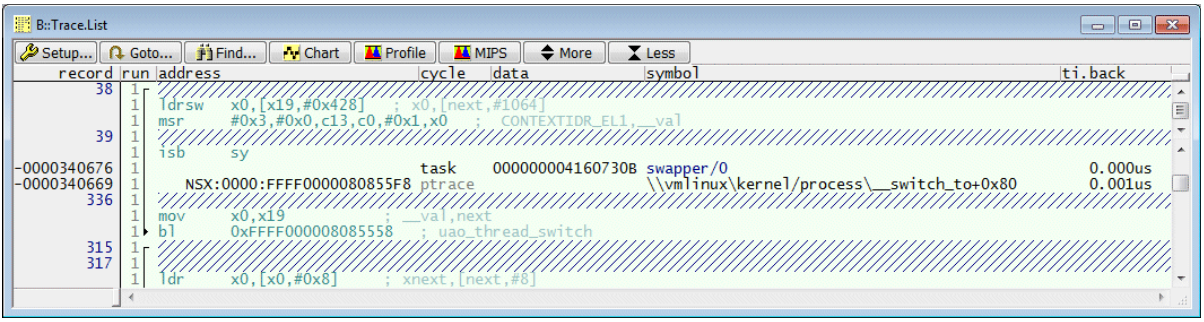
TP	*All raw trace data* as recorded at the trace port. For multicore systems the stream of trace data may contain information for *multiple cores* (e.g. in “wrapped mode” for CoreSight systems).
TPC	Raw trace data pertaining to a *single core*. For multicore systems this data is extracted from the overall trace data stream.
TPINFO	Information obtained by decoding a single trace packet (which may consist of multiple bytes).

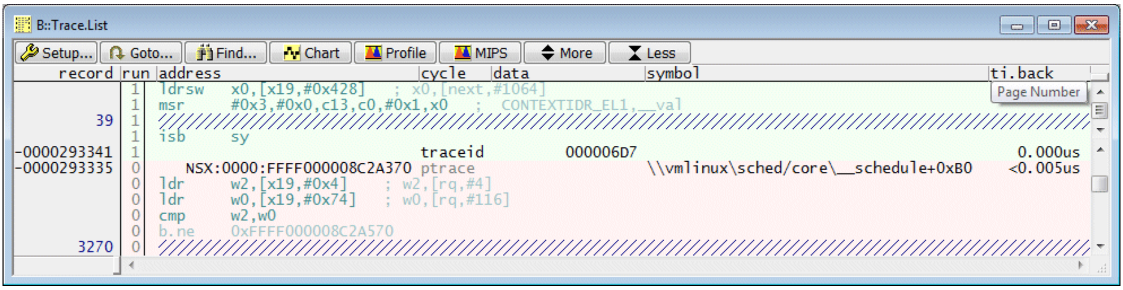


BEAT	All raw trace data as recorded at the Nexus port. Displays the same data as **TP**, but in a different format. BEAT displays the data in the format “MSEO[1..0]-BLANK-MDO[7..0]”, e.g. “3 42”, whereas TP displays the same data in the format “MDO[7..0]-MSEO[1..0]”, e.g. “10B”.



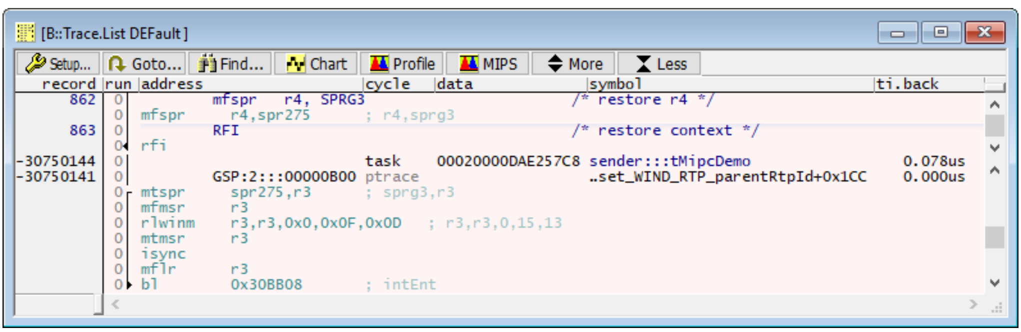
TSINFO	Timestamp calculation background information, required for the diagnosis of complex timing scenarios.





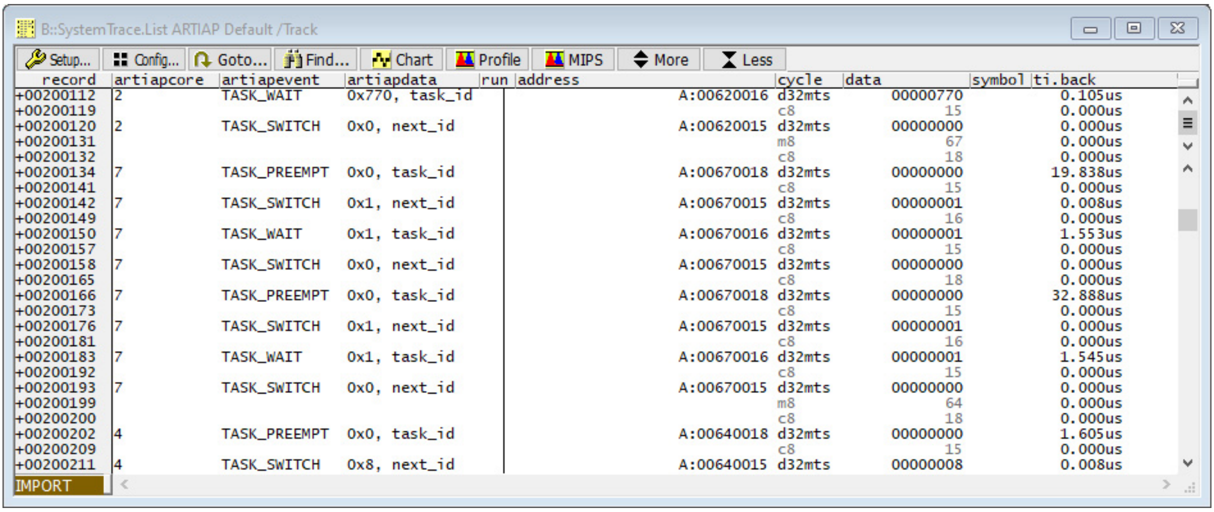
task	If a Context ID or ownership packet is decoded and if it is assignable to a task, the "task" cycle type and the task name is displayed. The displayed data value is a TRACE32 internal value.
traceid	If a Context ID or ownership packet is decoded and if it can not be assigned to a task or any other protocol-specific content such as service, intr etc. the cycle type "traceid" and the packet content is displayed.

If the machine ID is encoded, the machine name is also displayed ("sender" in the screenshot below).



ARTIAP Trace Decoding

ARTI (AUTOSAR Real-Time Interface) Trace Driver format defined on AUTOSAR Adaptive Platform is in MIPI STP (SYStem Trace Protocol) format.



The items are only supported when the hardware contains Arm System Trace Macrocell (STM).

ARTIAPCore	Decode STM MasterID into Core index based on board information.
ARTIAPEvent	Decode STM Channel as OS Event.
ARTIAPData	Decode STM data into meaningful data based on OS Event.
ARTIAP	ARTIAP shows 3 items: **ARTIAPCore ARTIAPEvent ARTIAPData**.

See also

■ <trace>.BookMark
■ <trace>.Timing
■ <trace>.View
■ IProbe.state
❏ Analyzer.RECORD.ADDRESS()
❏ Analyzer.RECORD.DATA()
❏ Analyzer.RECORD.OFFSET()
❏ Analyzer.RECORDS()
❏ Analyzer.REF()

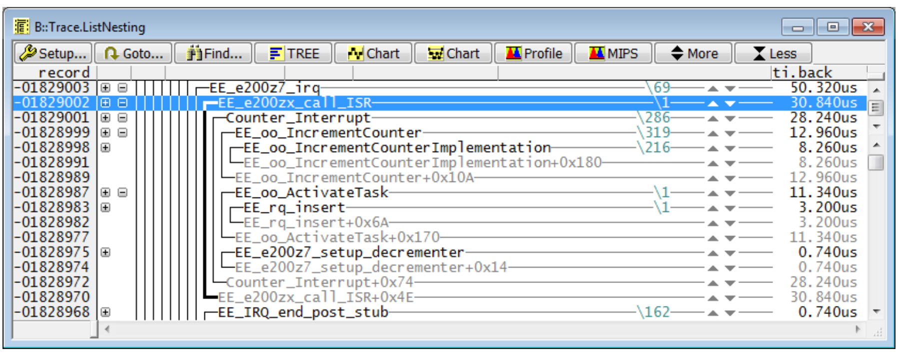
▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**ListNesting** [**/***<option>*]
<option>:	**CORE** *<number>* **SplitCORE** *<generic_options>*

The command **Trace.ListNesting** is mainly used to investigate issues in the construction of the call tree for the nesting function run-time analysis. Typical commands for the nesting function run-time analysis are the commands **Trace.STATistic.Func** or **Trace.STATistic.TREE**.

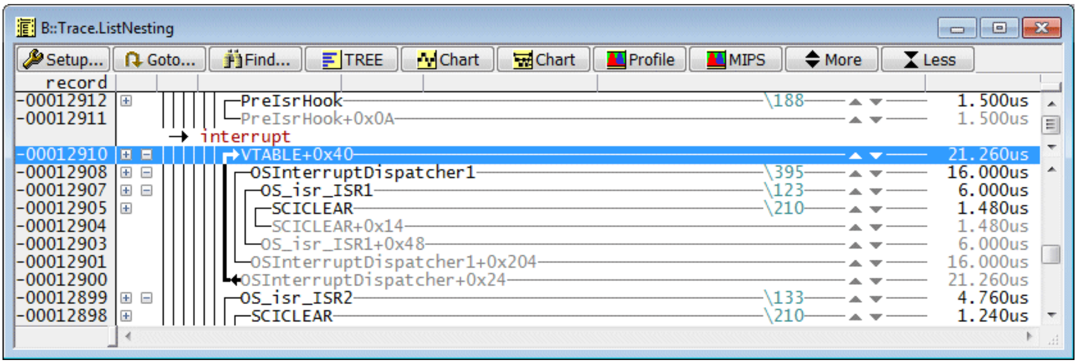
<option>	For a description of the generic options, see **<trace>.List**.
CORE *<number>* SMP tracing only.	Filters the **Trace.ListNesting** window by the specified core. Processing is done for all cores, but only the specified core is displayed. All other cores are temporarily hidden in the window.
SplitCORE SMP tracing only.	Displays the trace recording of the cores side by side in the **Trace.ListNesting** window.

The **Trace.ListNesting** window provides the nesting details. If a function entry point is selected, the path to the function exit is highlighted.



If the function exit is located far apart, you can use the Down Arrow (v) to jump to the function exit.

The interrupt nesting if marked specially (see screenshot below).



Code optimizations are the main reason for issues in the construction of the call tree. TRACE32 indicates these issues as PROBLEMs or WORKAROUNDs.

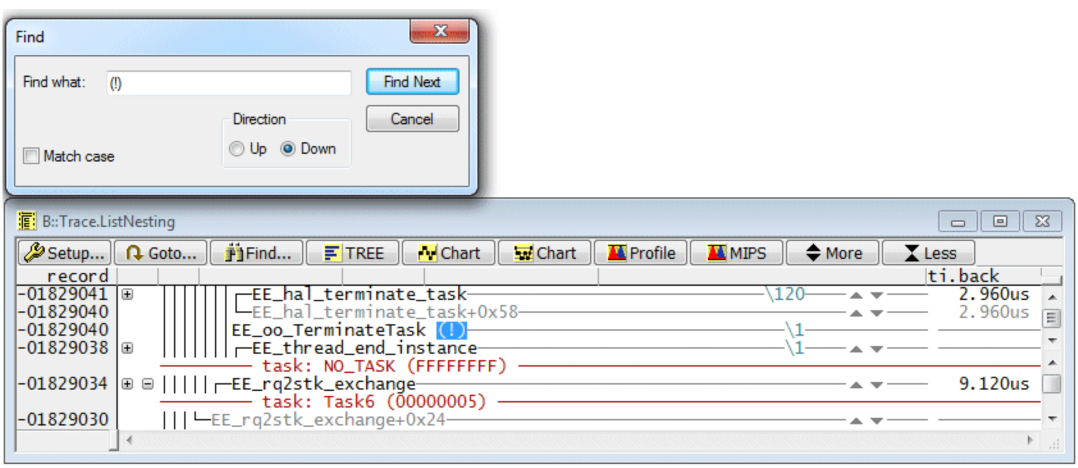
PROBLEMs

- A PROBLEM is a point in the trace recording that TRACE32 can not integrate into the current nesting.
- PROBLEMs are marked with (!) in the **Trace.ListNesting** window. The name of the expected function is shown.
- PROBLEMs are ignored in the construction of the call tree.
- PROBLEMs may affect the construction of the call tree, so it is important to inspect them. The **Statistic Markers** can be used to solve a PROBLEM.

To inspect a PROBLEM, proceed as follows:

1. Go to the start of the trace recording.
2. Use the **Find…** command from the **Edit** menu. Type `(!)` as find item.
3. Open a Trace Listing to inspect the problem in detail.

```
Trace.List List.TASK List.ADDRESS DEFault /Track
```

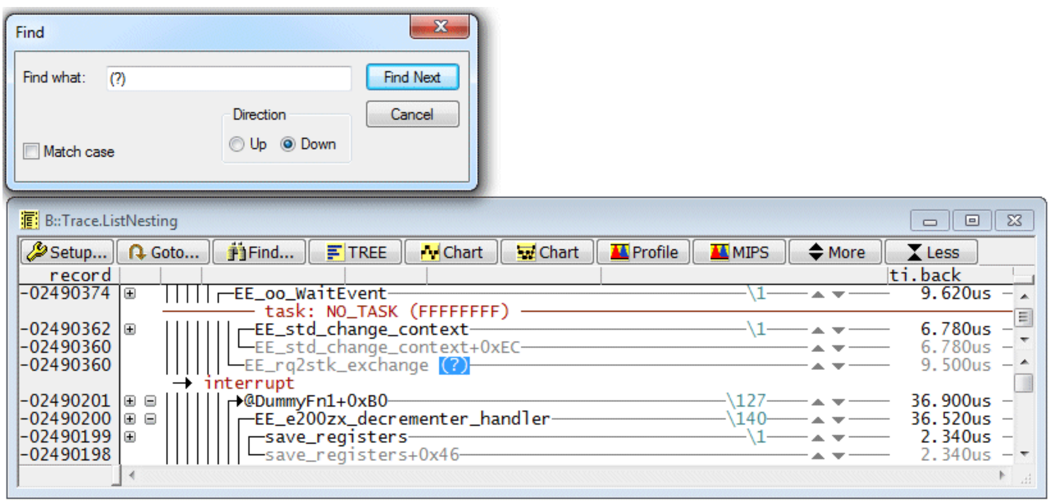


- A WORKAROUND is a point in the trace recording that TRACE32 can not integrate into the current nesting.
- TRACE32 attempts to integrate this point into the function nesting, by deriving information from previous scenarios in the nesting.
- WORKAROUNDs are marked with (?) in the **Trace.ListNesting** window.
- WORKAROUNDs may affect the construction of the call tree, if the derived information is wrong. It is recommended to inspect the WORKAROUNDs.

To inspect a WORKAROUND, proceed as follows:

1. Go to the start of the trace recording.
2. Use the **Find…** command from the **Edit** menu. Type `(?)` as find item.
3. Open a Trace Listing to inspect the problem in detail.

```
Trace.List List.TASK List.ADDRESS DEFault /Track
```



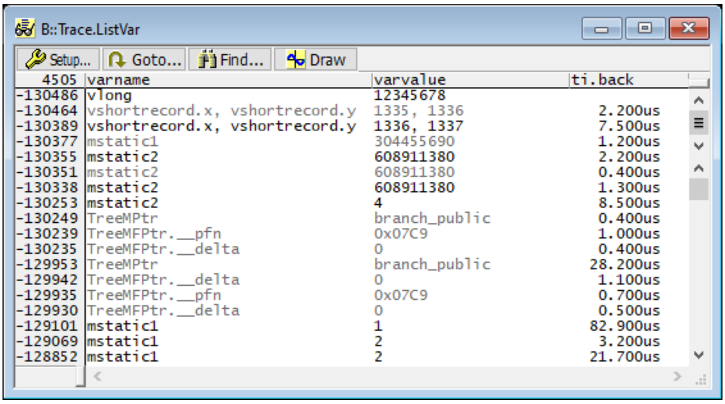
See also

■ IProbe.state

▲ 'Release Information' in 'Legacy Release History'

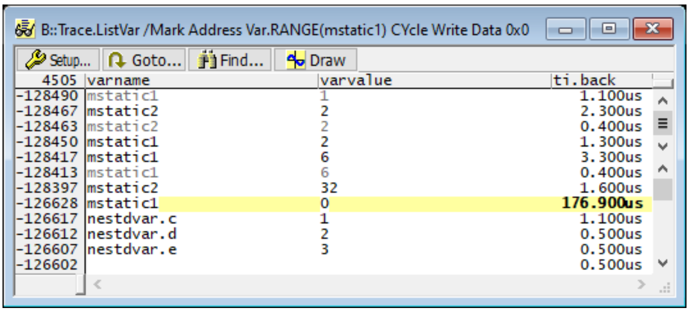
Format 1:	*<trace>***.ListVar %**[*<format>*] [{*<var>*}] [{**/***<options>* }]
Format 2:	*<trace>***.ListVar** *<range>* [**%**[*<format>*]] [{*<var>*}]
<format>:	**DEFault** | **STandDard** | **Decimal** | **Hex**
<range>:	*<record_range>* | *<time_range>*
<options>	**TASK** *<task>* **CORE** *<core_number>* **Split | SplitFill** **List** | **Mark** | **Track** **FILE** **Filter** *<filter>*

Displays a list of all variable recorded if it is used without parameters.



The option **Mark** allows to mark the specified variable access.

```
; mark trace entry when a 0x0 is written to variable mstatic1
Trace.ListVar /Mark Address Var.RANGE(mstatic1) CYcle Write Data 0x0
```

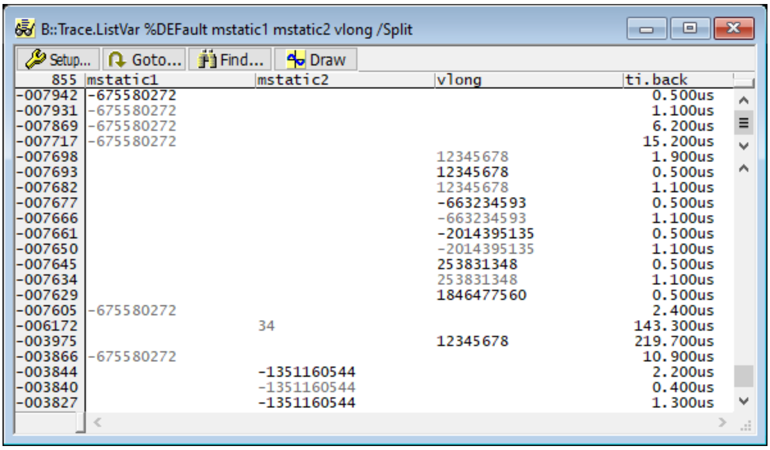


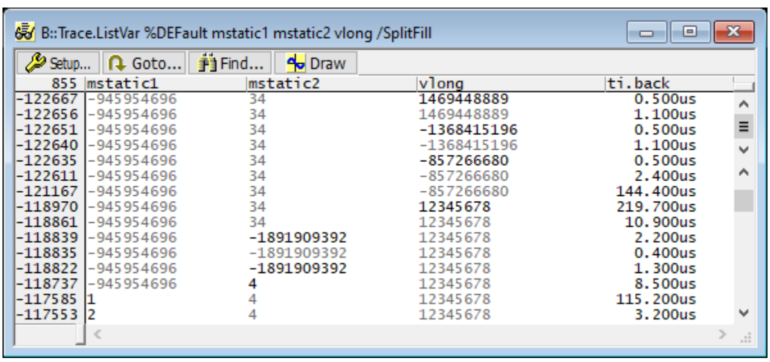
Format 1 represents the standard syntax, in which the variable names follow the **%**<*format*> parameter. The following options provide a representation in which variable values can be better compared:

Split	Each specified variable gets its own column. If a variable is accessed its value is displayed. Write accesses are printed in black, read accesses are printed in gray.
SplitFill	Each specified variable gets its own column. Whenever one of the specified variables is displayed, the current values of all other specified variables are displayed as well. Write accesses are printed in black, everything else is printed in gray.

Examples for Format 1:

```
//Display all accesses to the variable mstatic1
Trace.ListVar %DEFault mstatic1
//Display all accesses to the listed variables
Trace.ListVar %DEFault mstatic1 fstatic fstatic2
//Display all accesses to the listed variables, but display
//the values of each variable in a separate column
Trace.ListVar %DEFault mstatic1 mstatic2 vlong /Split
Trace.ListVar %Hex mstatic1 fstatic fstatic2 /Split
//Display all accesses to the listed variables, but display
//the values of each variable in a separate column
//fill in the current value of the not accessed variable to each line
Trace.ListVar %DEFault mstatic1 mstatic2 vlong /SplitFill
```





Format 2 represents the advanced syntax, here it is possible to restrict the display to the specified *<record_range>* or *<time_range>.*

Examples for Format 2:

```
Trace.ListVar (-14874903.)--(-14874761.)
Trace.ListVar 1.8s--10.8s
Trace.ListVar (-14874903.)--(-14874761.) vfloat
Trace.ListVar 1.8s--10.8s mstatic1 fstatic fstatic2 /SplitFill
```

See also

■ IProbe.state ■ Trace

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**LOAD** [*<file>*] [**/Config**]

Loads trace data from a file into the debugger. Typically **<trace>.LOAD** is used to analyze data in a simulator or to compare different recordings.

The command loads the data into the “normal” trace buffer i.e. the same buffer that is filled when recording data using an analyzer (e.g. via PowerTrace, PowerProbe, PowerIntegrator etc.). As the standard trace commands work on this buffer, they automatically work on the loaded data. To highlight that loaded data is displayed, windows are marked by a red label **LOAD** label in the bottom-left corner.

To save trace data, use the command **<trace>.SAVE**.

<file>	The default extension for the file name is ***.ad**.

NOTE:	There is a similar but slightly different command **<trace>.FILE**. It loads the trace data into a dedicated *file trace buffer*. To have trace commands (e.g. **Trace.List**) work on the *file trace buffer*, they need to be invoked with the parameter **/FILE**.

An example for working on loaded trace data:

```
Trace.LOAD test4                      ; load trace contents from file

Data.LOAD.Elf demo.elf /NoCODE        ; load symbol information for the
                                      ; post-processing

Trace.List                            ; display loaded trace contents

Trace.Chart.sYmbol                    ; symbol analysis of trace

Trace.STATistic.Func                  ; function run-time analysis
```

The TRACE display and analysis commands are re-directed to the selected **trace method** if:

- **Trace.LOAD** is executed without the parameter <*file*>.

```
Trace.LOAD                        ; Re-direct trace display and
                                  ; analysis commands to the selected
                                  ; trace method
```

- A trace configuration command is executed.

```
Trace.Init                        ; the trace configuration command
                                  ; Trace.Init re-directs the trace
                                  ; display and analysis commands to
                                  ; the selected trace method
```

- The program execution is started while **Trace.AutoArm** is set to ON.

If the **Trace.METHOD Probe** or **Trace.METHOD Integrator** was selected, when the trace contents were saved, the option **/Config** can be used to re-activate the **Probe/Integrator** and **NAME** settings.

```
Trace.METHOD Probe                     ; select the trace method Probe
                                       ; for the PowerProbe

;…

Trace.SAVE probetest1                  ; save the trace contents to the
                                       ; file probetest1

;…

QUIT                                   ; end TRACE32
```

```
                                       ; use a TRACE32 instruction set
                                       ; simulator to postprocess the
                                       ; PowerProbe trace data

Trace.LOAD probetest1 /Config          ; load the trace contents from the
                                       ; file probetest1

                                       ; load the Probe settings and NAMEs

Trace.List
```

See also

■ <trace>.FILE ■ <trace>.SAVE ■ IProbe.state ■ RunTime
■ RunTime.state

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**MERGEFILE** *<file>* [*<trace_area>*] [**/***<options>* …]
<trace_area>:	*<string>* *<range>* *<value>* *<time_range>*
<option>:	**TIMEGAP** *<time>* **ZIP** **QuickCompress** **Compress** **NoCompress**

Combines two trace files into one. This is useful for traces recorded for different cores working in AMP mode.

TIMEGAP *<time>*	Allows a seamless concatenation with regards to the timestamp
ZIP, QuickCompress, Compress, NoCompress	Control the compressing of the resulting file. These option are obsolete because the resulting file is compressed by default.

See also

- Trace

Format:	**Trace.METHOD** *<method>*
<method>:	**Analyzer** **ART** **CAnalyzer** **CIProbe** **FDX** **Integrator** **IProbe** **LA** **LOGGER** **Onchip** **Onchip2** **Probe** **SNOOPer** **NONE**

Selects the trace method you want to use. This allows you to work with a trace method other than the one suggested by TRACE32.

For information about how TRACE32 makes its suggestion, see **"What to know about the TRACE32 default settings for Trace.METHOD"**, page 119.

Trace Methods	Description
Analyzer	Trace memory is provided by one of the following TRACE32 tools: • TRACE32 PowerTrace or RiscTrace • TRACE32 Instruction Set Simulator • TRACE32 Front-End to virtual targets supporting trace
ART	Advanced Register Trace.
CAnalyzer	Compact Analyzer. Trace memory is provided as follows: • TRACE32 CombiProbe • µTrace (MicroTrace) • PowerDebug/Debug Cable configuration
CIProbe	The Lauterbach Analog Probe within the CombiProbe / µTrace (MicroTrace) is used to record signals.
FDX	Fast Data eXchange. The target application needs to write the required trace information to a small ring buffer (min. size 2 trace records). The contents of the ring buffer is transferred to the TRACE32 software while the program execution is running and saved there for later display. If the on-chip debug unit provides a Debug Communications Channel (DCC) the required trace information can be transferred directly to the TRACE32 software.
HAnalyzer	Trace RAM is provided by the host. This method is used for targets that provide a specifically implemented trace channel over interfaces like USB3.
Integrator	The Lauterbach logic analyzer PowerIntegrator is used to record the trace information.
IProbe	The Lauterbach IProbe logic analyzer within the PowerTrace II / PowerTrace III is used to record signals.
LA	LA (Logic Analyzer). Trace information not recorded by TRACE32 can be loaded and processed. This requires that the TRACE32 software is familiar with the format of the trace information.
LOGGER	The target application can write the required trace information to target RAM. TRACE32 loads the trace information from the target RAM for display and processing.
Onchip **Onchip2**	The trace information is saved in the first/second onchip trace buffer provided by the chip.

Trace Methods	Description
Probe	The Lauterbach logic analyzer PowerProbe is used to record the trace information.
SNOOPer	SNOOPer trace. For details, see **"Application Note for the SNOOPer Trace"** (app_snooper.pdf).
NONE	A dummy trace method indicating that the trace feature, including the **Trace.*** commands, is not yet operational. The only command exceptions are **Trace.METHOD** and **Trace.state**. Select the trace method you want to use, using either the **Trace.METHOD** command, the **Trace.state** window, or a PRACTICE script (*.cmm). For more information including illustrations, see **"What to know about the TRACE32 default settings for Trace.METHOD"**, page 119.

See also

- <trace>.Mode
- <trace>.state
- Analyzer
- ART
- CAnalyzer
- FDX
- HAnalyzer
- Integrator
- IProbe
- LA
- LOGGER
- Onchip
- Probe
- SNOOPer
- SystemTrace
- Trace.METHOD.Analyzer()
- Trace.METHOD.ART()
- Trace.METHOD.CAnalyzer()
- Trace.METHOD.FDX()
- Trace.METHOD.HAnalyzer()
- Trace.METHOD.Integrator()
- Trace.METHOD.IProbe()
- Trace.METHOD.LA()
- Trace.METHOD.LOGGER()
- Trace.METHOD.ONCHIP()
- Trace.METHOD.Probe()
- Trace.METHOD.SNOOPer()

- 'Trace Functions' in 'General Function Reference'
- 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**Mode** [*<mode>*]
<mode>:	**Fifo** **Stack** *<other modes>*

Selects the trace operation mode. The supported modes depend on the selected trace method. Please refer to the description of the trace operation mode for the selected trace method for more information, e.g. for the trace method Analyzer, refer to **Analyzer.Mode**. The most common operation modes are:

Fifo	If the trace is full, new records will overwrite older records. The trace records always the last cycles before the break. This mode is supported by all trace methods.
Stack	If the trace is full recording will be stopped. The trace always records the first cycles after starting the trace. This mode is supported by all trace methods.
Leash	Stops the program execution when trace is nearly full.

STREAM	The trace data is immediately conveyed to a file on the host after it was placed into the trace memory of the TRACE32 trace tool. This procedure extends the size of the trace memory to up to several T Frames. STREAM mode can only be used if the average data rate at the trace port does not exceed the maximum transmission rate of the host interface in use. Peak loads at the trace port are intercepted by the trace memory of the TRACE32 trace tool, which can be considered to be operating as a large FIFO. Depending on the command group, STREAM mode can only be used for some TRACE32 trace tools: • Analyzer: - PowerTrace Serial / PowerTrace Serial 2 - PowerTrace II / PowerTrace III - TRACE32 POWERTRACE/ ETHERNET supports STREAM mode for some trace protocols. If it is not supported, the command **Trace.Mode STREAM** is blocked. • CAnalyzer: - All configurations • Integrator: - PowerIntegrator II (Probe A-E only) • IProbe: - All tools except PowerTrace II • CIProbe: - All tools The streaming file is placed into the TRACE32 temp directory (**OS.PresentTemporaryDirectory()**) by default and is named *<trace32_instance_id>***stream***<method>***.t32** (trace32_instance_id is the value of **OS.ID()**, method is one of a/ca/i/ip/cip). If you explicitly want to specify a location for the streaming file use the command **<trace>.STREAMFILE** *<file>*.
PIPE	The trace data is immediately conveyed to the host and distributed to user-defined trace sinks. Not supported with PowerTrace Ethernet 256/512MB. See **<trace>.PipeWRITE**.
RTS	The RTS radio button is only an indicator that shows if Real-time Profiling is enabled. For enabling RTS use the command **RTS.ON**.

See also

- Trace.METHOD
- <trace>.STREAMFILE
- <trace>.STREAMLOAD
- Analyzer.Mode
- CIProbe.Mode
- HAnalyzer.Mode
- IProbe.Mode
- LOGGER.Mode
- Probe.Mode
- <trace>.STREAMCompression
- <trace>.STREAMFileLimit
- <trace>.STREAMSAVE
- ART.Mode
- FDX.Mode
- Integrator.Mode
- LA.Mode
- Onchip.Mode
- SNOOPer.Mode

<trace>.OFF   Switch off

Format: *<trace>*.**OFF**

Disables both trace memory and the trigger unit. The trace memory can be read and the trigger unit be programmed.

See also

■ IProbe.state ■ RunTime ■ RunTime.state ❏ Analyzer.STATE()

<trace>.PipeWRITE   Connect to a named pipe to stream trace data

Format: *<trace>*.**PipeWRITE** [*<name>*]

Connect to a named pipe to stream the raw trace data to an external application. If *<name>* is omitted, the debugger disconnects from the named pipe.

Example: (for Windows)

```
Trace.Mode PIPE                        ; switch to PIPE mode

Trace.PipeWRITE \\.\pipe\ptrace        ; connect to named pipe

; run test
Go                                     ; trace data now streamed to
;...                                    ; external application
Break

Trace.PipeWRITE                        ; disconnect from named pipe
```

See also

■ Trace

<trace>.PlatformCLOCK   Set clock for platform traces

PowerPC QorIQ

Format: *<trace>*.**PlatformCLOCK** *<frequency>*

Sets the clock for platform traces (DDRTrace, OCeaNTrace).

This command is only available for PowerPC QorIQ cores. Refer to **"QorIQ Debugger and NEXUS Trace"** (debugger_ppcqoriq.pdf) for more information.

<trace>.PortFilter   Specify utilization of trace memory

Format:	*<trace>*.**PortFilter AUTO** | **OFF** | **MIN** | **PACK** | **MAX**

If a TRACE32 trace tool is used and the trace information is conveyed to the host computer at the recording time, it is advantageous to reduce the amount of data to be conveyed. This goal can be achieved by the following:

- Reducing the recording of idle cycles (applies only if the on-chip trace logic generates idle cycles).
- Not conveying TRACE32 **tool time stamps** to the host computer, if they are not required for the intended analysis.

The command **Trace.PortFilter** allows the following configurations:

AUTO	Best setting is done automatically by TRACE32 (default). This means in detail: • With streaming (**Trace.Mode STREAM**) or PIPE Mode (**Trace.Mode.PIPE**) all TRACE32 trace tools operate in PACK mode. • With real-time profiling (**RTS.ON**) all TRACE32 trace tools operate in MAX mode. Otherwise: • CombiProbe and µTrace (MicroTrace) operate in PACK mode. • A PowerTrace with an AutoFocus II or AutoFocus MIPI preprocessor operates in PACK mode. • All other PowerTrace setups operate in MIN mode.
OFF	All generated trace information is recorded (for diagnostic purposes only).
MIN	Idle cycles are partly not recorded.
PACK	No idle cycles are recorded. Caveats: The accuracy of the TRACE32 **tool time stamps** is reduced.
MAX	No idle cycles are recorded and no TRACE32 **tool time stamps** are conveyed to the host computer.

See also

■ RTS.ON

<trace>.PortSize — Set external port size

Format:	*<trace>*.**PortSize 1** | **2** | **3** | … | **16** | **AUTO**

Informs the debugger that the externally visible port size differs from the internal port size setting of **TPIU.PortSize** and sets the specified external port size. Use this command if there is application-specific logic between the TPIU and the analyzer, for example in the programmable logic part of an FPGA SoC.

The external port size value refers to the number of data pins that are physically connected to the analyzer.

The internal port size value refers to the setting that will be programmed into the target's TPIU.

AUTO (default)	The external port size value of *<trace>*.**PortSize** equals the internal port size value of **TPIU.PortSize**.
1 … **16**	Use the specified number of data pins as the external port size.

See also

■ TPIU.PortSize
▲ 'Introduction' in 'Debugging Embedded Cores in Xilinx FPGAs [Zynq]'

<trace>.PortType — Specify trace interface

Format:	*<trace>*.**PortType TPIU** | **STM** | **SWV** (CombiProbe) *<trace>*.**PortType TPIU** | **TPIUX2** | **TPIUX3** | **TPIUX4** | **STM** | **RTP** | **TPIU+RTP** (Preprocessor AutoFocus II) *<trace>*.**PortType HSSTP** | **SETM3** (Preprocessor Serial)

Inform TRACE32 PowerView about the trace port interface type provided by your target. This might be necessary for the following TRACE32 trace tools:

TRACE32 CombiProbe:

TPIU (default)	CombiProbe is connected to TPIU.
STM	CombiProbe is connected to STM interface.
SWV	CombiProbe is connected to Serial Wire Viewer interface.

TRACE32 Preprocessor AutoFocus II:

TPIU	TRACE32 AutoFocus II Preprocessor is connected to TPIU (default). Also supported by LA-7991 PP-ARM-ETM-AF.
TPIUX2	TRACE32 AutoFocus II is connected to a trace port interface that provides 2 ETMv3 interfaces, multicore chip without TPIU (NEC Triton only).
TPIUX3	TRACE32 AutoFocus II is connected to a trace port interface that provides 3 ETMv3 interfaces, multicore chip without TPIU (NEC Triton only).
TPIUX4	TRACE32 AutoFocus II is connected to a trace port interface that provides 4 ETMv3 interfaces, multicore chip without TPIU (NEC Triton only).
STM	TRACE32 AutoFocus II Preprocessor is connected to STM interface. Also supported by LA-7991 PP-ARM-ETM-AF.
RTP	TRACE32 AutoFocus II Preprocessor is connected to Ram Trace Port interface.
TPIU+RTP	TRACE32 AutoFocus II Preprocessor is connected to a trace port interface that includes a TPIU and a Ram Trace Port interface.

TRACE32 Preprocessor Serial:

HSSTP	TRACE32 Preprocessor Serial is connected to a HSSTP interface (default).
SETM	TRACE32 Preprocessor Serial is connected to a SETM interface.

<trace>.PROfile — Rolling live plots of trace data

The **<trace>.PROfile** command group displays plots that are based on polling trace hardware and update in real time. See also **Count.PROfile**, **Data.PROfile** and **Var.PROfile** for similar plots of data polled directly from the target.

See also

- IProbe.state

<trace>.PROfile.channel — Display profile of signal probe channels

Format:	*<trace>*.**PROfile.channel** *[<items ...>]* [*/options*]
<options>:	**AutoInit** | **AutoArm**

Displays a rolling live plot of analog or digital channels of e. g. a Mixed-Signal Probe. If no *<items>* are given, the command plots all analog channels that are enabled (see **POD.ADC**).

With the options **AutoInit** and **AutoArm**, the window can be tied to the execution of the target program.

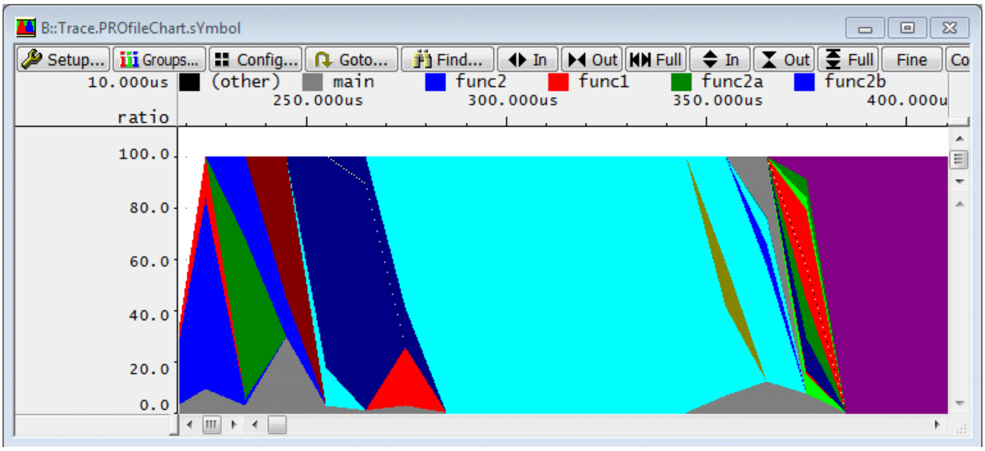
Channels are sampled approximately every 100 milliseconds and data is shown for the last 100 seconds. Sampling happens independently from the normal operation of the trace (i. e. **<trace>.Arm** or **<trace>.OFF**). This command is intended as a visual aid for slow or interactive changes of channels, not as a replacement for windows like **<trace>.DRAW.channel** or **<trace>.Timing**.

<trace>.PROfile.CTU — Display complex trigger unit counter profile

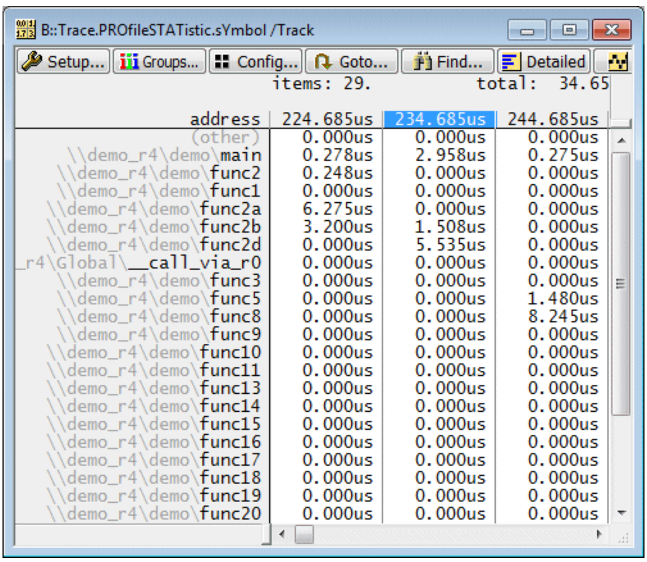
Format:	*<trace>*.**PROfile** *<counter>* [*<gate>*]
<gate>:	**0.1s** | **1.0s** | **10.0s**

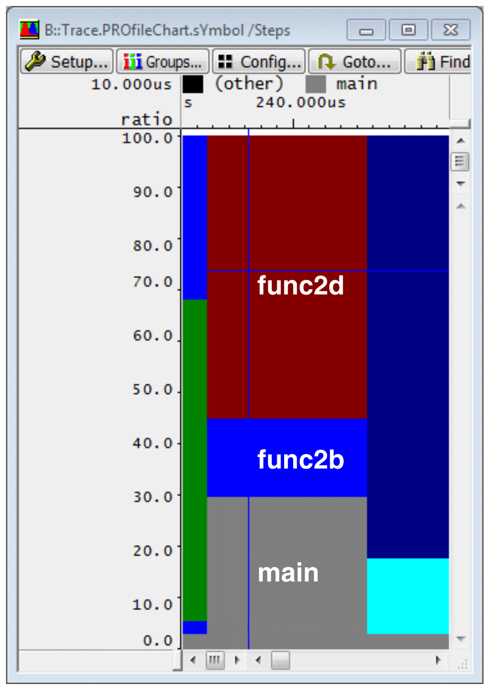
The contents of a trigger unit counter can be displayed as a function of time. Time counters are displayed in percent and event counters as events/s. Refer to **“Complex Trigger Unit for Nexus MPC5xxx”** (app_ctu_mpc5xxx.pdf) for more information.

The **<trace>.PROfileChart** command group displays distributions versus time graphically as color chart with fixed time intervals. The result is a stacked graph where the total ratio at a given time represent the sum of the ratios for all items at that time.

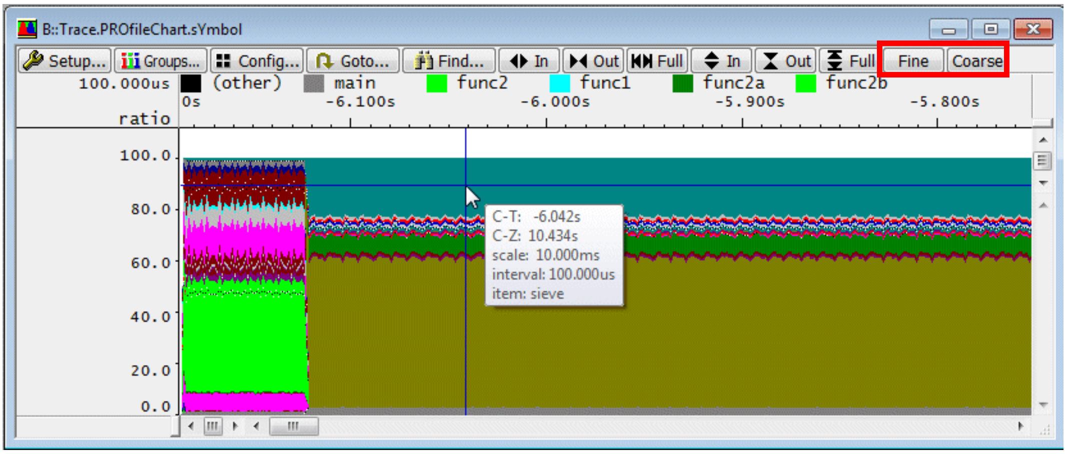


To draw the **Trace.PROfileChart** graphic, TRACE32 PowerView partitions the recorded instruction flow information into time intervals. The default interval size is 10 us. For each time interval rectangles are draw that represent the time ratio, events or time consumed within the time interval. For the final display this basic graph is smoothed.





The time interval size can be changed using the **Fine** and **Coarse** buttons.

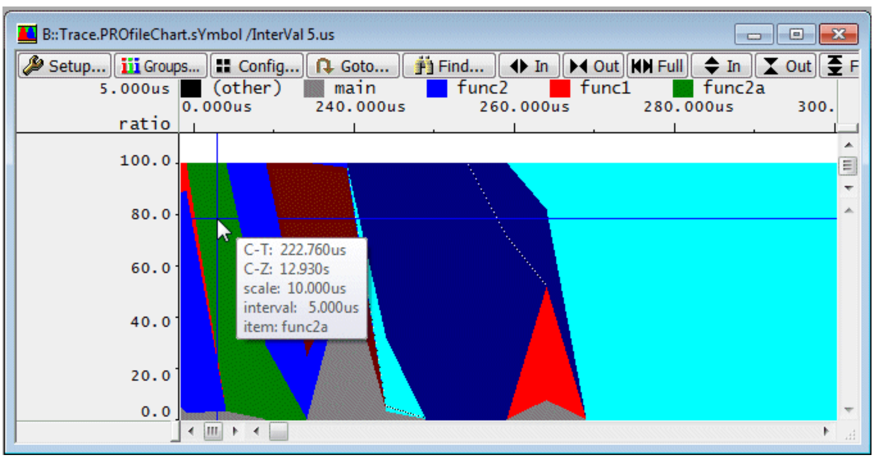


Fine	Decrease the time interval size by the factor 10
Coarse	Increase the time interval size by the factor 10

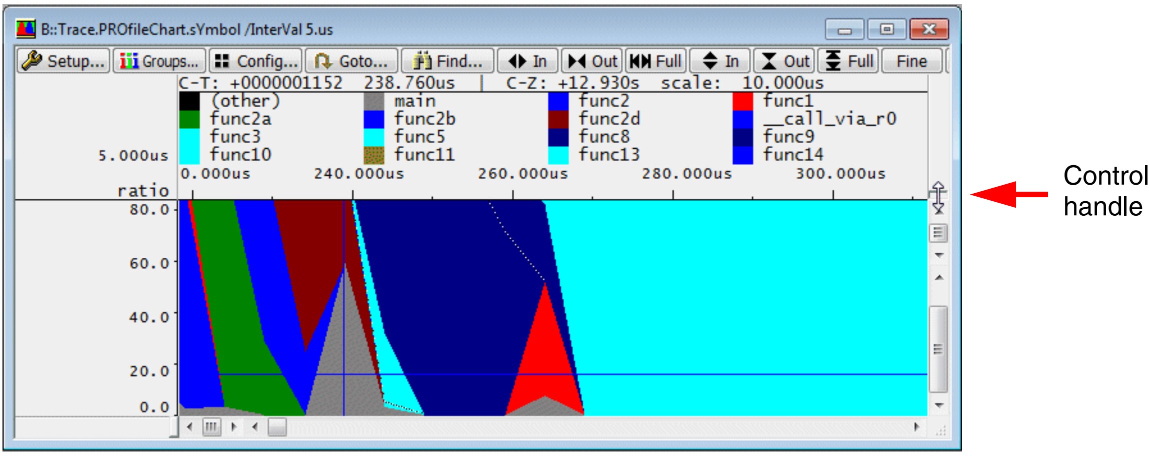
The time interval size can also be set manually using the **/InterVal** option:

```
Trace.PROfileChart.sYmbol /InterVal 5.ms        ; change the time
                                                ; interval size to 5.ms
```

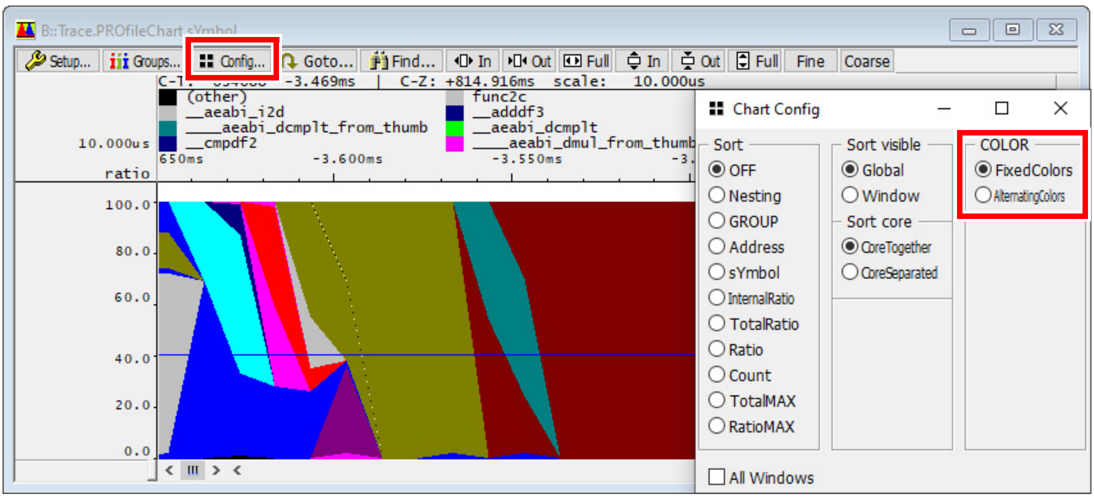
The tooltip at the cursor position shows the color assignment and the used interval size.



Use the control handle on the right upper corner of the **Trace.PROfileChart** window to get a color legend.



The color assignment is done per default statically (**FixedColors**), i.e. colors are assigned fixed to items. Fixed color assignment has the risk that two functions with the same color are drawn side by side and thus may convey a wrong impression of the dynamic behavior. Alternatively, a dynamic color assignment can be used instead (**AlternatingColors**), i.e. colors are assigned by the recording order of the items again and again for each measurement. The color assignment can be changed from the **Trace.PROfileChart** window using the **Config** button or using the **/Color** option in the command line.



Options

This section describes the options of the **<trace>.PROfileChart** command group. Not all options are supported by all **<trace>.PROfileChart** commands.

Track	The cursor in the **<trace>.PROfileChart** window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed. The zoom factor of the **<trace>.PROfileChart** window is retained, even if the trace content changes.
ZoomTrack	Same as option **Track**. If the tracking in performed with another **<trace>.PROfileChart** window the same zoom factor is used.
Sort [*<sort_visible>*] [*<sort_core>*] [*<sort>*]	Specify sorting criterion for analyzed items. For almost all commands the analyzed items are displayed in the order they are recorded by default. Details on the sorting criterion can be found at the description of the command **Trace.STATistic.Sort**.
InterVal *<time>*	Allows to divide the time period recorded by the trace (total) into time slices. Additional analysis details can be displayed for these time slices.
Address *<address* | *range>*	Display the results for the selected address or address range
FILE	Use the trace contents loaded with the command **<trace>.FILE**.
FlowTrace	Trace works as a program flow Trace. This option is usually not required.
BusTrace	Trace works as a bus trace. This option is usually not required.
INLINE	Treat inline functions as separate functions (default).
NoINLINE	Discard inline function from the results.
LABEL	Include all symbols in the results.
NoLABEL	Only include functions in the results.
RecScale	Display trace in fixed record raster. This is the default.
TimeScale	Display trace as true time display, time relative to the trigger point (respectively the last record in the trace).
TimeZero	Display trace as true time display, time relative to zero point. For more information about the zero point refer to **ZERO**.
TimeREF	Display trace as true time display, time relative to the reference point. For more information about the reference point refer to **<trace>.REF**.

Filter *<item>*	Filter the described item.
TASK *<task_magic>*, etc.	Operating system task in OS-aware debugging and tracing. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).
SplitTASK	Trace information is analyzed independently for each task. The time chart displays these individual results.
MergeTASK	Trace information is analyzed independently for each task. The time chart summarizes these results to a single result.
CORE *<n>*	Time chart is only displayed for the specified core. Only available for **SMP** multicore tracing.
SplitCORE	Trace information is analyzed independently for each core. The time chart displays these individual results. Only available for **SMP** multicore tracing.
MergeCORE	Trace information is analyzed independently for each core. The time chart summarizes these results to a single result. Only available for **SMP** multicore tracing.
JoinCORE	Core information is ignored for the time chart. Only available for **SMP** multicore tracing.

Draw Options:

Steps	Connect the dots for the data values by steps.
Vector	Connect the dots for the data values by vectors.
Color FixedColors	Colors are assigned fixed to items (default). Fixed color assignment has the risk that two functions with the same color are drawn side by side and thus may convey a wrong impression of the dynamic behavior.
Color AlternatingColors	Colors are assigned by the recording order of the items again and again for each measurement.

See also

- <trace>.Chart
- <trace>.PROfileChart.Address
- <trace>.PROfileChart.AddressGROUP
- <trace>.PROfileChart.AddressRate
- <trace>.PROfileChart.COUNTER
- <trace>.PROfileChart.DatasYmbol
- <trace>.PROfileChart.DIStance
- <trace>.PROfileChart.DistriB
- <trace>.PROfileChart.DURation
- <trace>.PROfileChart.GROUP
- <trace>.PROfileChart.INTERRUPT
- <trace>.PROfileChart.Line

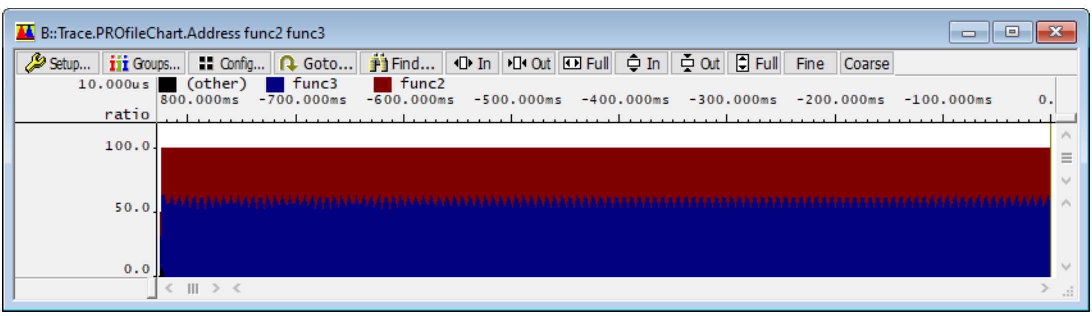
- <trace>.PROfileChart.MODULE
- <trace>.PROfileChart.PROGRAM
- <trace>.PROfileChart.Rate
- <trace>.PROfileChart.sYmbol
- <trace>.PROfileChart.TASKINFO
- <trace>.PROfileChart.TASKKernel
- <trace>.PROfileChart.TASKSRV
- <trace>.PROfileChart.TASKVSINTR
- <trace>.PROfileSTATistic
- BMC.PROfileChart
- IProbe.state
- RunTime.state
- <trace>.PROfileChart.PAddress
- <trace>.PROfileChart.PsYmbol
- <trace>.PROfileChart.RUNNABLE
- <trace>.PROfileChart.TASK
- <trace>.PROfileChart.TASKINTR
- <trace>.PROfileChart.TASKORINTERRUPT
- <trace>.PROfileChart.TASKVSINTERRUPT
- <trace>.PROfileChart.Var
- <trace>.STATistic
- EVENTS.PROfileChart
- RunTime

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**PROfileChart.Address** [*<trace_area>*] *<address1>* [*<address2>* ...] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Display the time interval between up to 8 program events as a profile chart.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.



Example:

```
Trace.PROfileChart.Address func2 func3
```

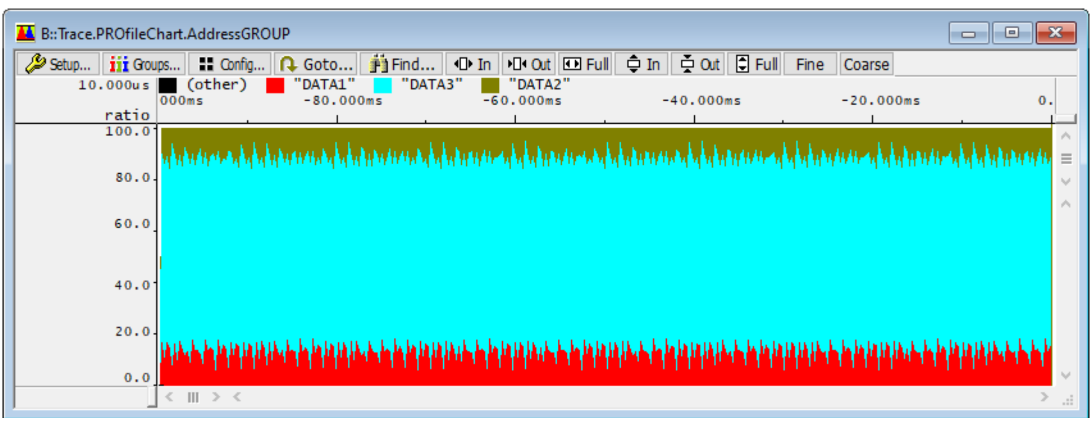
See also

■ <trace>.PROfileChart

Format:	*<trace>*.**PROfileChart.AddressGROUP** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<number>* **|** **SplitCORE** **|** **MergeCORE** | **JoinCORE** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Track** | **ZomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Address** *<item>* | *<range>* **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

The time for accessed address **groups** is displayed as time profile chart (flat statistic). The results include groups for both program and data addresses.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.



Example:

```
GROUP.Create "DATA1" 0x0000--0x1FFF /RED

GROUP.Create "DATA2" 0x2000--0x6FFF /OLIVE

GROUP.Create "DATA3" 0x7000--0x9fff /AQUA

Trace.PROfileChart.AddressGROUP
```

See also

- <trace>.PROfileChart
- <trace>.PROfileChart.GROUP
- BMC.PROfileChart.AddressGROUP

Format:	*<trace>*.**PROfileChart.AddressRate** [*<trace_area>*] *<address1>* [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Sort** *<item>* **Address** *<address* | *range>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Display the frequency of execution for the selected address.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

See also

- <trace>.PROfileChart

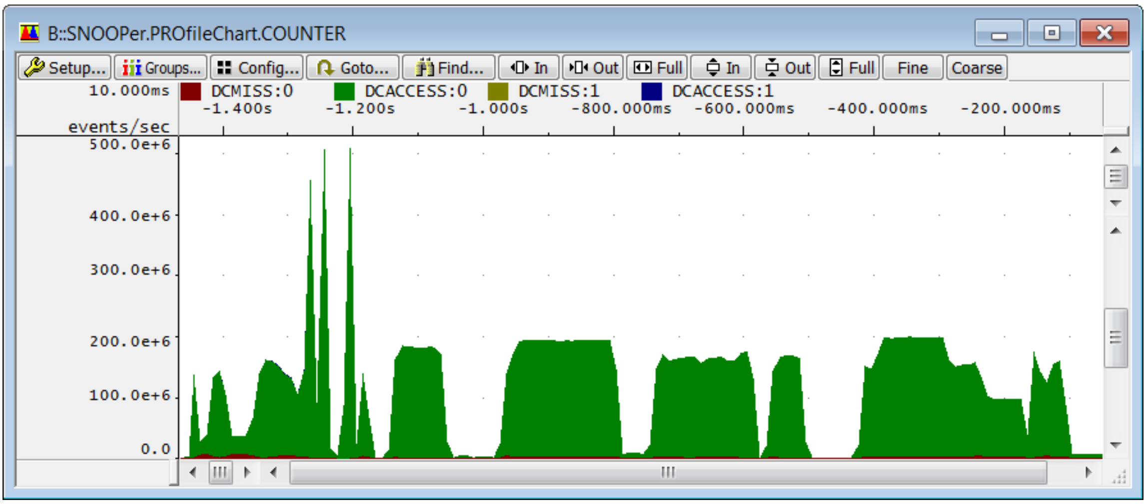
Format:	*<trace>*.**PROfileChart.COUNTER**[**%***<format>*][*<trace_area>*][*<items>*] [**/***<option>*]
<format>:	**ZeroUp.** [*<width>*] | **Up.** [*<width>*] | **Down.** [*<width>*] | **Frequency.** [*<width>*] | **POWER.** [*<width>*]
<width>:	**DEFault** | **Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **SByte**
<items>:	**DEFault** | **ALL** | *<cpu>* | *<signals>* | **Port**[**.***<subitem>*] | **MARK**[**.***<marker>*] | **ENERGY.Abs** | **POWER**[**.OFF**] | **SAMPLE**[**.OFF**] | **SPARE**[**.OFF**] | **LOW** | **HIGH** | **FINDINDEX**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **InterVal** *<time>* **Filter** *<filter_item>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Shows the time profiles of a counter that is traced as data value.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
ZeroUp	The counter does not increment steadily but starts counting from zero on each trace record.
Up	The counter starts counting from zero and increments steadily (default).
Down	The counter starts counting at its maximum value and decrements steadily.
Frequency	The trace records do not contain counter values but frequencies.
POWER	Used for **ETA** traces.

Example: sample data cache / data buffer hits and misses on a TriCore processor with **SNOOPer** trace and BenchMark Counter

```
BMC.RESet                                  ; reset BMC configuration

BMC.M1CNT DATA_X_HIT                       ; count data cache / data buffer
                                           ; hits

BMC.M2CNT DATA_X_CLEAN                     ; count data cache / data buffer
                                           ; misses

BMC.SnoopSet ON                            ; configure the SNOOPer trace for
                                           ; event counter recording

SNOOPer.PROfileChart.COUNTER
```



The result is a stacked graph i.e. the total number of events/s at a given time represent the sum of the events for all counters at that time.

See also

■ <trace>.PROfileChart

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.PROfileChart.DatasYmbol** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **LABEL** | **NoLABEL** | **INLINE** | **NoINLINE** **InterVal** *<time>* **Filter** *<filter_items>* **Sort** *<item>* **Track** | **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the contents of a pointer graphically.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

See also

- <trace>.PROfileChart
- BMC.PROfileChart.DatasYmbol

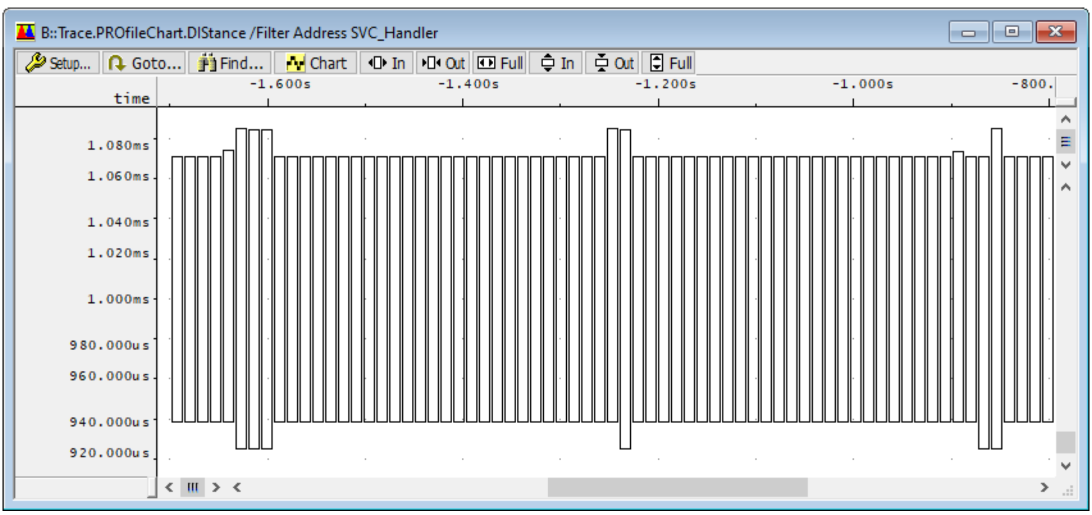
Format:	*<trace>***.PROfileChart.DIStance** [*<trace_area>*] [**/***<option>*] *<trace>***.Chart.DIStance** (deprecated)
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Steps** | **Vector**

Display the time interval for a single event graphically.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

Example: use the option /Filter to filter out the event of interest.

```
Trace.PROfileChart.DIStance /Filter Address SVC_Handler
```



See also

■ <trace>.PROfileChart

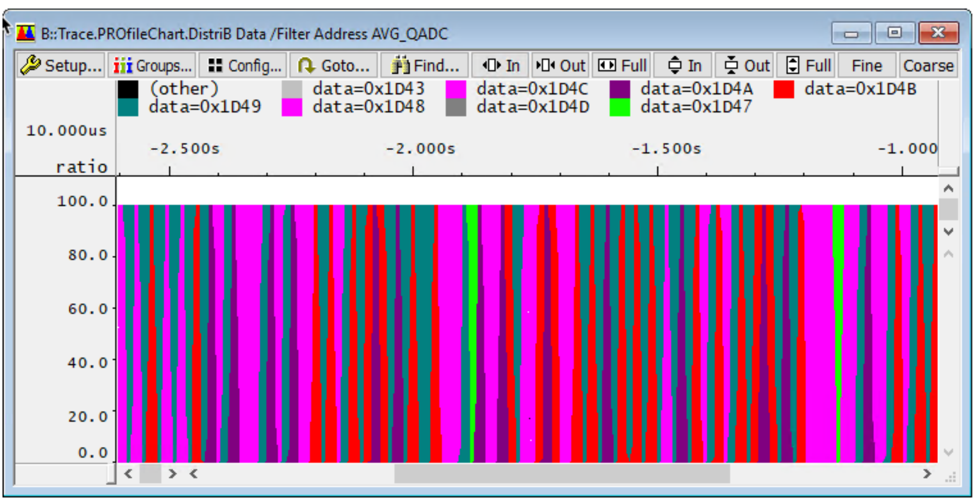
Format:	*<trace>*.**PROfileChart.DistriB** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Address** *<address* | *range>* | **InterVal** *<time>* **Filter** *<filter_items>* | **Sort** *<item>* **Track** | **ZoomTrack** **Vector** | **Steps** | **Color** [**FixedColors** | **AlternatingColors**]

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

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

Example: Display distribution of data value for AVG_QADC

```
Trace.PROfileChart.DistriB Data.L /Filter Address AVG_QADC
```



See also

■ <trace>.PROfileChart

Format:	*<trace>*.**PROfileChart.DURation** [*<trace_area>*] [**/***<option>*] *<trace>*.**Chart.DURation** (deprecated)
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**ATOA** | **ATOB** **BTOA** | **BTOB** **FilterA** *<filter>* | **FilterB** *<filter>* **FILE** **FlowTrace** | **BusTrace** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Track** | **ZoomTrack** **Vector** | **Steps**

Graphical display of time intervals between two events.

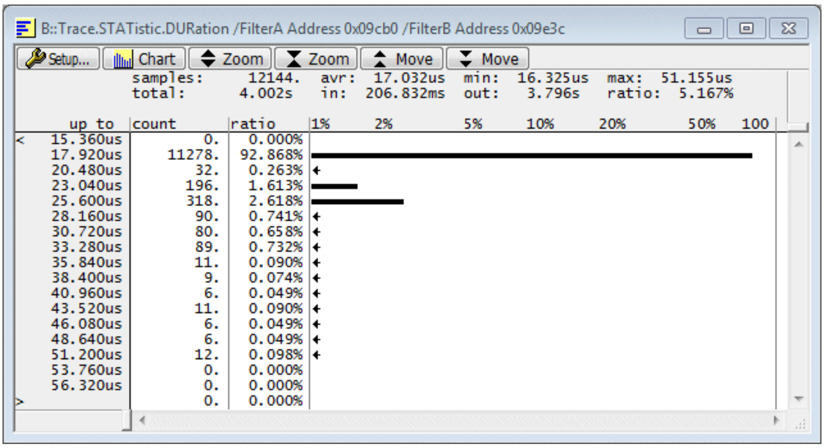
<option>	Refer to **<trace>.PROfileChart** for a description of the general options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
FilterA *<item>*	Specify the first event, see example below.
FilterB *<item>*	Specify the second event, see example below.
ATOA	Display the time interval from A to A, see example below.
ATOB	Display the time interval from A to B, see example below.
BTOA	Display the time interval from B to A, see example below.
BTOB	Display the time interval from B to B, see example below. If no selective tracing is possible and more specific events should be displayed it is also possible to use the options:

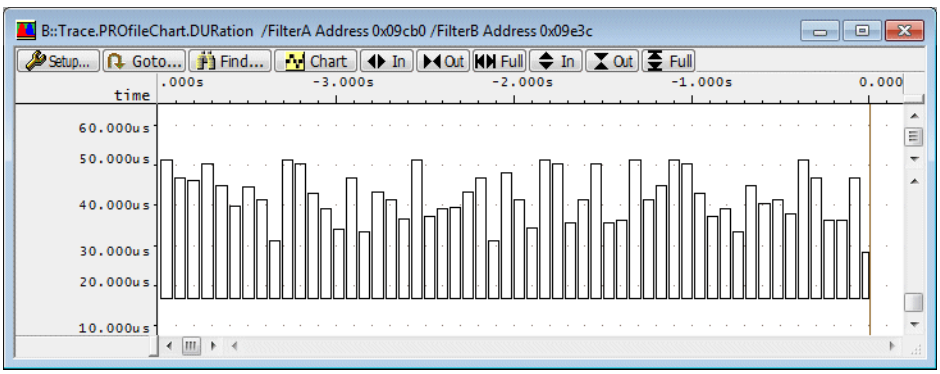
In order to use the command **Trace.STATistic.DURation**:

- Check if both events are exported by a trace packet. Information reconstructed by TRACE32 is not analyzed.
- Alternatively use a **TraceEnable** breakpoint export the event as a trace packet.

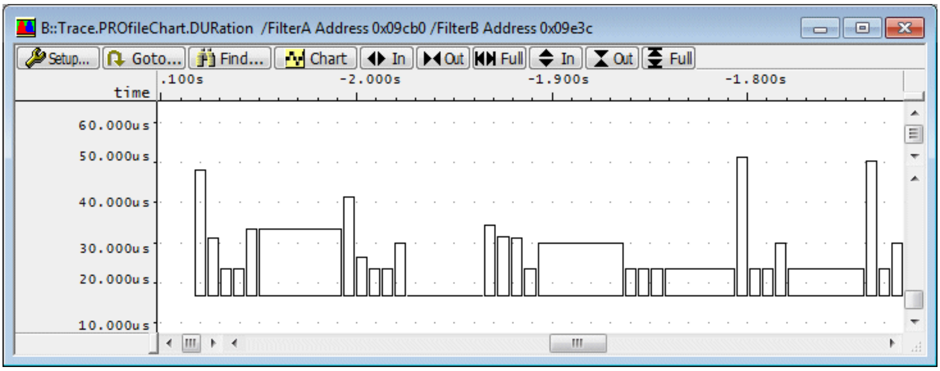
The options **FilterA** and **FilterB** provide you with the means to describe your event.

```
Trace.Mode Leash

Break.Set 0x9cb0 /Program /TraceEnable

Break.Set 0x9e3c /Program /TraceEnable

Go

WAIT !STATE.RUN()

Trace.STATistic.DURation /FilterA Address 0x9cb0 /FilterB Address 0x9e3c

Trace.PROfileChart.DURation /FilterA Address 0x9cb0
                            /FilterB Address 0x9e3c
```





Displays min and max duration per 10 pixels



Displays min and max duration per 10 pixels (with a higher resolution)

See also

■ <trace>.PROfileChart

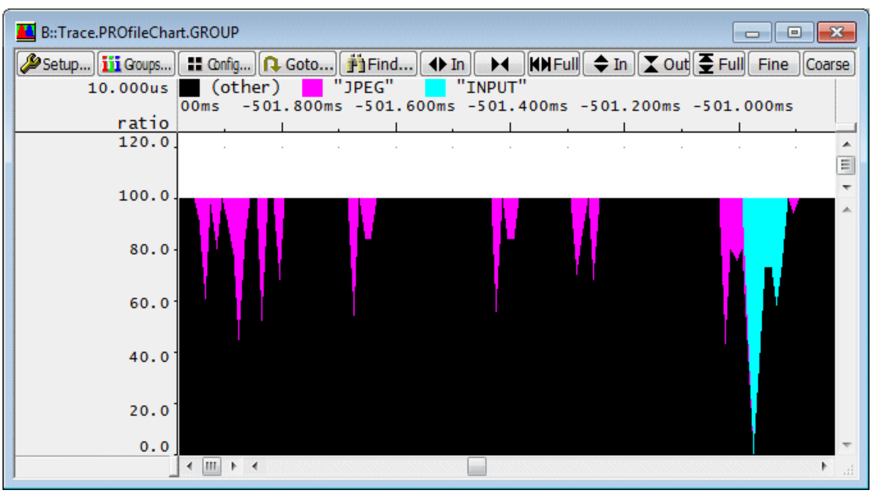
Format:	*<trace>*.**PROfileChart.GROUP** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Address** *<address* | *range>* | **InterVal** *<time>* **Filter** *<filter_items>* | **Sort** *<item>* | **Track** | **ZoomTrack** **Vector** | **Steps** | **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the **group** behavior and displays the result as a color chart with fixed time intervals. The results only include groups within the program range. Groups for data addresses are not included.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

Example:

```
GROUP.Create "INPUT" \jquant2 \jquant1 \jidctred \jdinput /AQUA
GROUP.Create "JPEG" \jdapimin \jdcolor \jddctmgr \jdcoefct /NAVY
Trace.PROfileChart.GROUP
```



See also

- ■ <trace>.PROfileChart
- ■ <trace>.PROfileChart.AddressGROUP
- ■ BMC.PROfileChart.GROUP

- ▲ 'Release Information' in 'Legacy Release History'

<trace>.PROfileChart.INTERRUPT — Display interrupt profile chart

Format:	*<trace>*.**PROfileChart.INTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

The time spent in different interrupts is displayed graphically as profile chart. This feature is only available if TRACE32 has been set for OS-aware debugging.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

See also

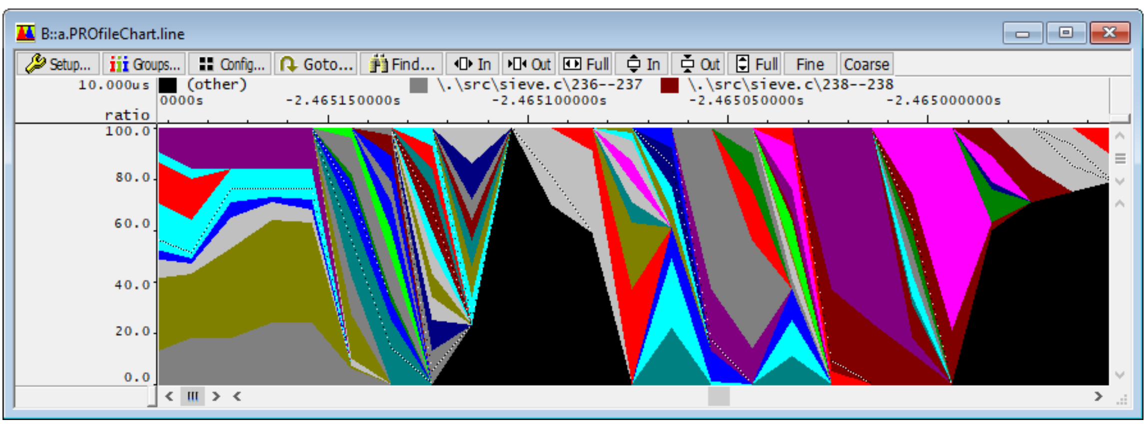
- ■ <trace>.PROfileChart

Format:	*<trace>*.**PROfileChart.Line** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the dynamic program behavior for high-level code lines and displays the result as a color chart with fixed time intervals.

Trace.PROfileChart.Line is based on a flat function run-time analysis.

<option>	Refer to **<trace>.PROfileChart** for a description of the options.
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.



See also

■ <trace>.PROfileChart ■ BMC.PROfileChart.Line

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

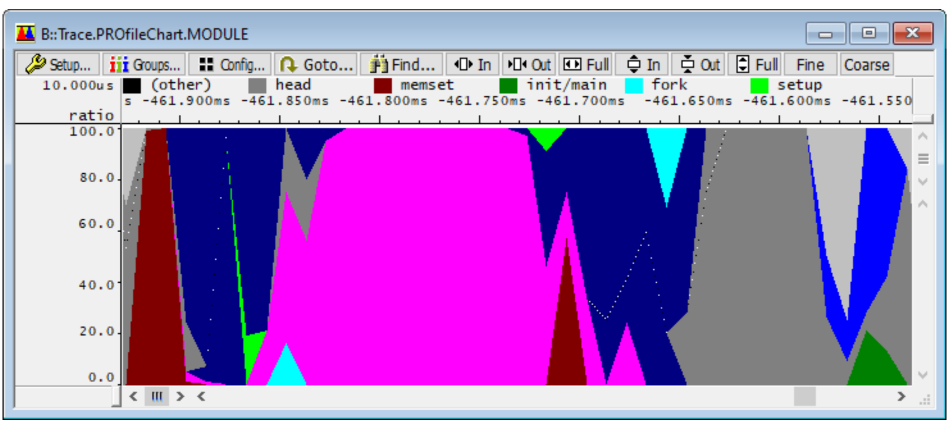
<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

<option>:
FILE
FlowTrace | **BusTrace**
TASK *<task>* | **SplitTASK** | **MergeTASK**
CORE *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE**
Track | **ZoomTrack**
RecScale | **TimeScale** | **TimeZero** | **TimeREF**
Address *<address* | *range>*
Filter *<item>*
Sort *<item>*
InterVal *<time>*
Vector | **Steps**
Color [**FixedColors** | **AlternatingColors**]

Analyzes the dynamic program behavior for symbol modules and displays the result as a color chart with fixed time intervals. The list of loaded modules can be displayed with **sYmbol.List.Module**.

Trace.PROfileChart.MODULE is based on a flat function run-time analysis.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.



See also

■ <trace>.PROfileChart ■ BMC.PROfileChart.MODULE

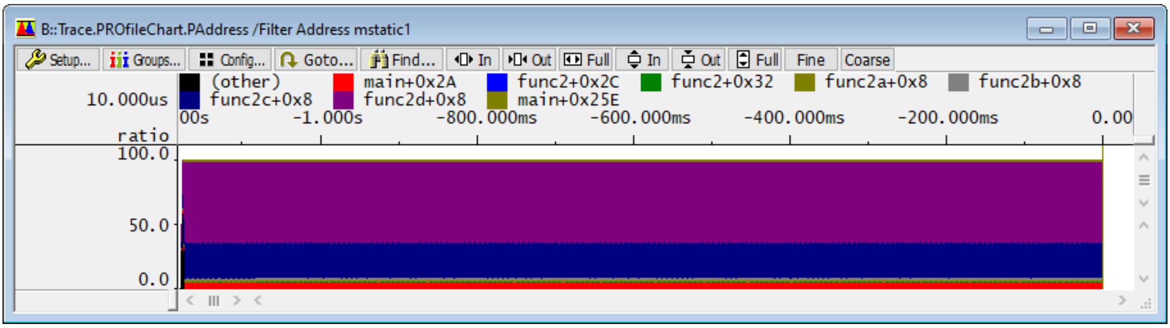
Format:	*<trace>***.PROfileChart.PAddress** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Address** *<address* | *range>* **Filter** *<item>* **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

The command provides a graphical profile chart of the instructions that accessed data addresses. You can select a specific address using the **/Filter** option.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

Example:

```
; display a profile chart of all addresses that accessed mstatic1
Trace.PROfileChart.PAddress /Filter sYmbol mstatic1
```



See also

■ <trace>.PROfileChart

<trace>.PROfileChart.PROGRAM — Program profile chart

Format:	*<trace>*.**PROfileChart.PROGRAM** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Address** *<address* | *range>* **Filter** *<item>* **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the dynamic execution behavior brocken down by loaded object files (program) and displays the result as a color chart with fixed time intervals. The loaded programs can be displayed with the command **sYmbol.Browse ***.

Trace.PROfileChart.PROGRAM is based on a flat function run-time analysis.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

■ <trace>.PROfileChart ■ BMC.PROfileChart.PROGRAM

<trace>.PROfileChart.PsYmbol Which functions accessed data address

Format: *<trace>*.**PROfileChart.PsYmbol** [*<trace_area>*] [**/***<option>*]

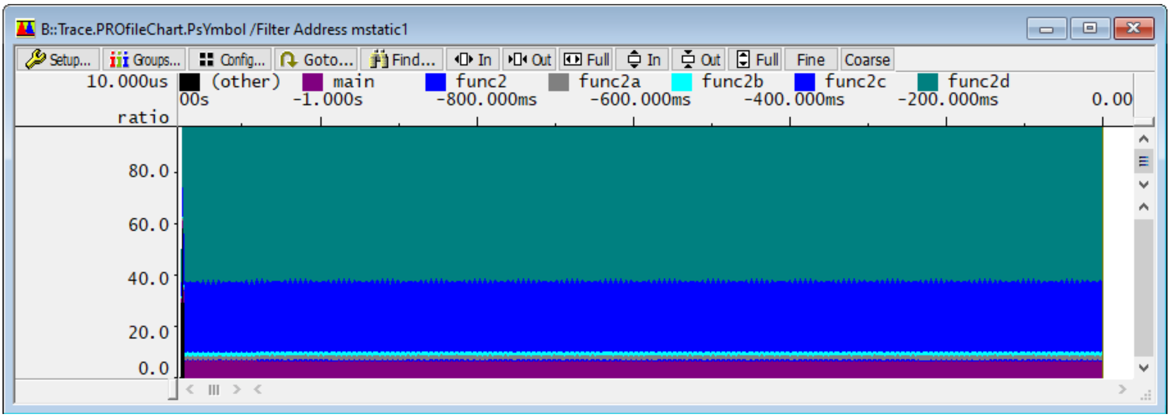
<option>:
FILE
FlowTrace | **BusTrace**
TASK *<task>* | **SplitTASK** | **MergeTASK**
CORE *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE**
INLINE | **NoINLINE** | **LABEL** | **NoLABLE**
Track | **ZoomTrack**
RecScale | **TimeScale** | **TimeZero** | **TimeREF**
Address *<address* | *range>*
Filter *<item>*
Sort *<item>*
InterVal *<time>*
Vector | **Steps**
Color [**FixedColors** | **AlternatingColors**]

The command provides a graphical profile chart of the functions that accessed data addresses. You can select a specific address using the **/Filter** option.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

Example:

```
; display a profile chart of all functions that accessed mstatic1
Trace.PROfileChart.PsYmbol /Filter sYmbol mstatic1
```



See also

■ <trace>.PROfileChart

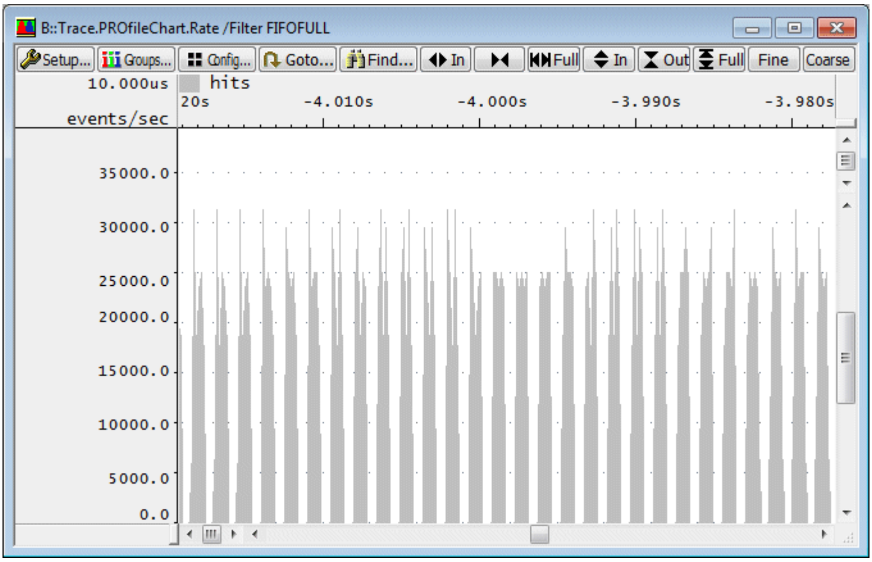
Format:	*<trace>*.**PROfileChart.Rate** [*<trace_area>*] [**/***<option>*] *<trace>*.**Chart.Rate** (deprecated)
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **INLINE** | **NoINLINE** | **LABEL** | **NoLABEL** **Address** *<address* | *range>* **Filter** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Graphical display of the event frequency over the time. Displays the rate of all cycles except dummy cycles.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

Example: Display the **TARGET FIFO OVERFLOW** (FIFOFULL) rate over the time.

```
Trace.PROfileChart.Rate /Filter FIFOFULL
```



See also

■ <trace>.PROfileChart

Format:	*<trace>*.**PROfileChart.RUNNABLE** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* **| SplitTASK | MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track | ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Address** *<address* | *range>* **Filter** *<item>* **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

The command provides a graphical profile chart of AUTOSAR runnables. 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>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

On TriCore AURIX there's a solution available for the Vector AUTOSAR tools that uses an automated instrumentation to trace runnables on all cores with minimum overhead. See *~~/demo/env/vector/rte_profiling*.

Otherwise, all functions that start an AUTOSAR "Runnable" have to be marked with the command **sYmbol.MARKER.Create RUNNABLESTARTPLUSSTOP**. Please refer to **"Trace Export for Third-Party Timing Tools"** (app_timing_tools.pdf) for more information.

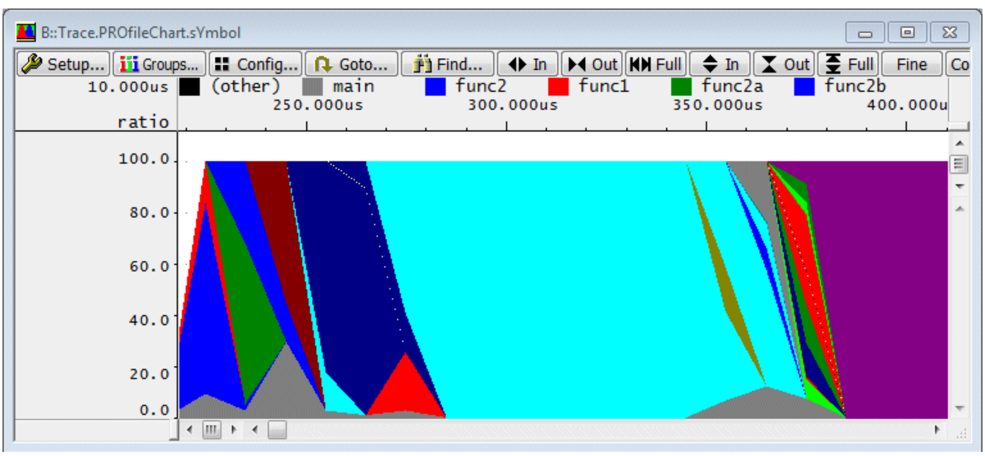
See also

- <trace>.PROfileChart

Format:	*<trace>*.**PROfileChart.sYmbol** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **LABEL** | **NoLABEL** | **INLINE** | **NoINLINE** **Address** *<address* | *range>* **InterVal** *<time>* **Filter** *<filter_items>* **Sort** *<item>* **Track** | **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the dynamic program behavior and displays the result as a color chart with fixed time intervals. **Trace.PROfileChart.sYmbol** is based on a flat function run-time analysis.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.



See also

- <trace>.PROfileChart
- BMC.PROfileChart.sYmbol
- CTS.PROfileChart.sYmbol

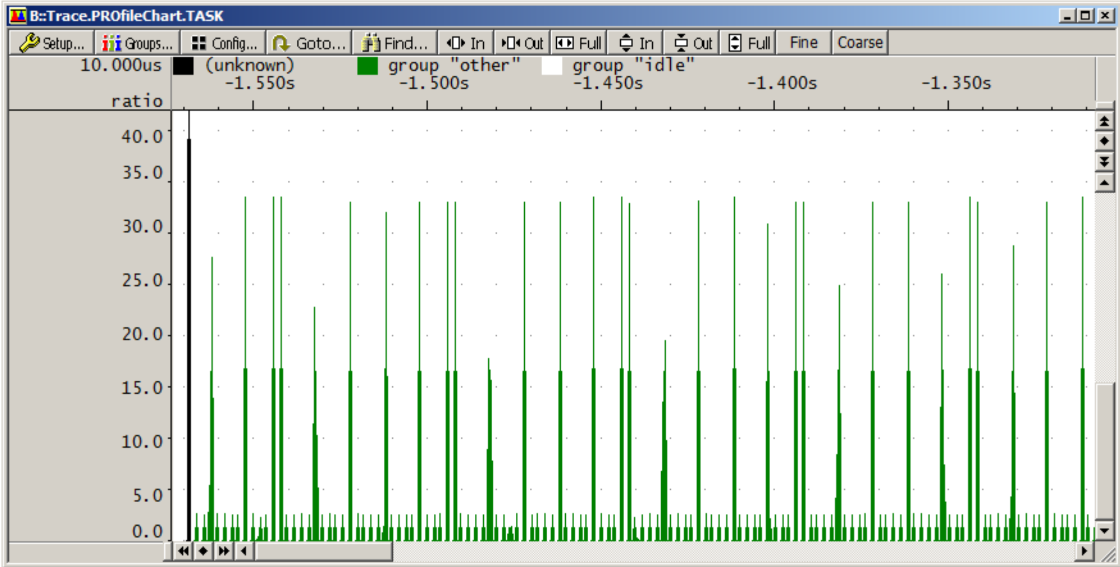
Format:	*<trace>*.**PROfileChart.TASK** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the dynamic task behavior and displays the result as a color chart with fixed time intervals. This command requires **OS-ware tracing**.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

Example to analyze CPU load:

```
; group all tasks that contain an idle loop to the group "Idle"
; all other tasks are members of the group "other"

; merge the result of all "Idle" group members and
; use white as "Idle" group color
GROUP.CreateTASK "Idle" "Idle_Task" /Merge /WHITE

; merge the result of all "other" group members
GROUP.Merge "other"

; use green as "other" group color
GROUP.COLOR "other" GREEN

; display the CPU load graphically
Trace.PROfileChart.TASK
```



(unknown) represents the time before the first task information was recorded to the trace.

See also

- ■ <trace>.PROfileChart
- ■ BMC.PROfileChart.TASK
- ■ CTS.PROfileChart.TASK
- ▲ 'CPU Load Measurement' in 'Application Note Profiling on AUTOSAR CP with ARTI'

<trace>.PROfileChart.TASKINFO — Context ID special messages

Format:	*<trace>*.**PROfileChart.TASKINFO** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Displays a graphical profile chart of special messages written to the Context ID register for ETM trace. The range of special values has to be reserved with the **ETM.ReserveContextID** command. These special values are then not interpreted for task switch or memory space switch detection.

This can be used for cores without data trace to pass data by the target application to the trace tool by writing to the ContextID register.

See also

- <trace>.PROfileChart
- BMC.PROfileChart.TASKINFO
- CTS.PROfileChart.TASKINFO

<trace>.PROfileChart.TASKINTR — ISR2 profile chart (ORTI)

Format:	*<trace>*.**PROfileChart.TASKINTR** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Displays graphical profile chart for ORTI based ISR2. This feature can only be used if the ISR2 can be traced based on the information provided by the ORTI file.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

- <trace>.PROfileChart
- BMC.PROfileChart.TASKINTR
- CTS.PROfileChart.TASKINTR

▲ 'Trace Features' in 'OS Awareness Manual OSEK/ORTI'

Format:	*<trace>*.**PROfileChart.TASKKernel** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Displays profile chart for results of **Trace.STATistic.TASKKernel**. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

- <trace>.PROfileChart
- CTS.PROfileChart.TASKKernel
- BMC.PROfileChart.TASKKernel

Format:	*<trace>*.**PROfileChart.TASKORINTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Analyzes the dynamic task and interrupt behavior and displays the result as a color chart with fixed time intervals. This command requires **OS-ware tracing**.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

- <trace>.PROfileChart
- CTS.PROfileChart.TASKORINTERRUPT
- BMC.PROfileChart.TASKORINTERRUPT

Format:	*<trace>*.**PROfileChart.TASKSRV** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

The time spent in OS service routines and different tasks is displayed as 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>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

- <trace>.PROfileChart
- CTS.PROfileChart.TASKSRV
- BMC.PROfileChart.TASKSRV

Format:	*<trace>*.**PROfileChart.TASKVSINTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Displays a graphical profile chart of tasks that were interrupted by interrupt service routines. This command requires **OS-ware tracing**.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

- <trace>.PROfileChart
- BMC.PROfileChart
- ETA.PROfileChart
- MIPS.PROfileChart

Format:	*<trace>***.PROfileChart.TASKVSINTR** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **InterVal** *<time>* **Sort** *<item>* **Track** **ZoomTrack** **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Displays a graphical 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>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

See also

- <trace>.PROfileChart
- BMC.PROfileChart.TASKVSINTR
- CTS.PROfileChart.TASKVSINTR

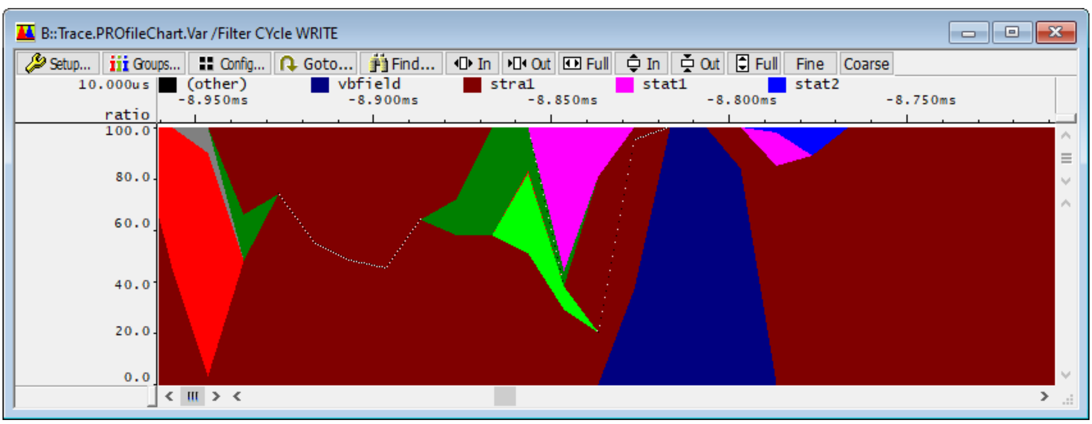
Format:	*<trace>*.**PROfileChart.Var** [*<record_range>*] [*<scale>*] [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **Track** | **ZoomTrack** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** **Filter** *<item>* **Address** *<address* | *range>* **Sort** *<item>* **InterVal** *<time>* **Vector** | **Steps** **Color** [**FixedColors** | **AlternatingColors**]

Displays a profile chart for variable accesses in the trace recording.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileChart** for a description of the options.

Example:

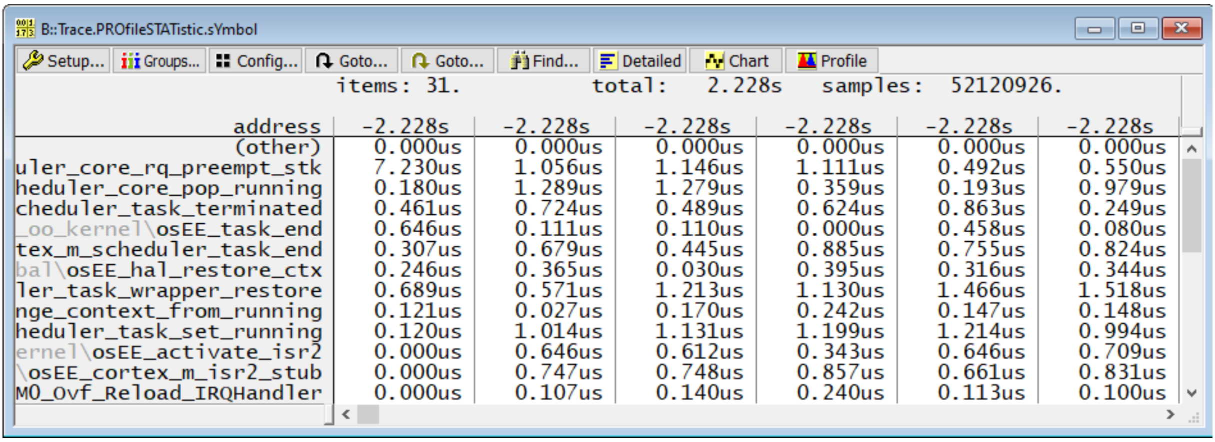
```
Trace.PROfileChart.Var /Filter CYcle WRITE
```



See also

■ <trace>.PROfileChart

The command group **<trace>.PROfileSTATistic** shows the results of numerical interval analysis in tabular format.



Options

This section describes the options of the **<trace>.PROfileSTATistic** command group. Not all options are supported by all **<trace>.PROfileChart** commands.

<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<core>* | **SplitCORE** | **MergeCORE** | **JoinCORE** **RecScale** | **IndexScale** | **TimeScale** | **TimeZero** | **TimeREF** **InterVal** *<time>* **Ratio** | **Compress** | **ROTATE** **Filter** *<item>* **INCremental** | **FULL** **Sort** *<item>* **Track**

FILE	Use the trace contents loaded with the command **<trace>.FILE**.
FlowTrace	Trace works as a program flow Trace. This option is usually not required.
BusTrace	Trace works as a bus trace. This option is usually not required.
TASK *<task_magic>*, etc.	Operating system task in OS-aware debugging and tracing. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).
SplitTASK	Trace information is analyzed independently for each task. The time chart displays these individual results.
MergeTASK	Trace information is analyzed independently for each task. The time chart summarizes these results to a single result.
CORE *<n>*	Time chart is only displayed for the specified core. Only available for **SMP** multicore tracing.
SplitCORE	Trace information is analyzed independently for each core. The time chart displays these individual results. Only available for **SMP** multicore tracing.
MergeCORE	Trace information is analyzed independently for each core. The time chart summarizes these results to a single result. Only available for **SMP** multicore tracing.
JoinCORE	Core information is ignored for the time chart. Only available for **SMP** multicore tracing.
RecScale	Display trace in fixed record raster. This is the default.
IndexScale	Results with index display.
TimeScale	Display trace as true time display, time relative to the trigger point (respectively the last record in the trace).
TimeZero	Display trace as true time display, time relative to zero point. For more information about the zero point refer to **ZERO**.
TimeREF	Display trace as true time display, time relative to the reference point. For more information about the reference point refer to **<trace>.REF**.
InterVal *<time>*	Allows to divide the time period recorded by the trace (total) into time slices. Additional analysis details can be displayed for these time slices.
Ratio	Ratio of time spent over the complete measurement is displayed instead of time.

ROTATE	Rotate x- and y-axis.
Filter *<item>*	Filter the described item.
INCremental	Intermediate results are displayed while the TRACE32 software analyzes the trace contents (default).
FULL	The result is displayed after the TRACE32 software finished the analysis.
Sort [*<sort_visible>*] [*<sort_core>*] [*<sort>*]	Specify sorting criterion for analyzed items. For almost all commands the analyzed items are displayed in the order they are recorded by default. Details on the sorting criterion can be found at the description of the command **Trace.STATistic.Sort**.
Track	The cursor in the **<trace>.PROfileChart** window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed. The zoom factor of the **<trace>.PROfileChart** window is retained, even if the trace content changes.

See also

■ <trace>.Chart ■ <trace>.PROfileChart ■ <trace>.STATistic ■ BMC.PROfileSTATistic
■ EVENTS.PROfileSTATistic

▲ 'Release Information' in 'Legacy Release History'

<trace>.PROfileSTATistic.Address Statistical analysis for addresses

Format:	*<trace>*.**PROfileSTATistic.Address** [*<trace_area>*] *<address1>* [*<address2>* ...] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for addresses.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.Address

<trace>.PROfileSTATistic.AddressGROUP Stat. for address groups

Format:	*<trace>*.**PROfileSTATistic.AddressGROUP** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for address **groups**. The results include groups for both program and data addresses.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- <trace>.PROfileSTATistic.GROUP
- BMC.PROfileSTATistic.AddressGROUP

<trace>.PROfileSTATistic.COUNTER — Statistical analysis for counter

Format:	*<trace>*.**PROfileSTATistic.COUNTER** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for counter traced as data value.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

<trace>.PROfileSTATistic.DatasYmbol — Statistic analysis for pointer content

Format:	*<trace>*.**PROfileSTATistic.DatasYmbol** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for pointer contents symbolically.

See also

- BMC.PROfileSTATistic.DatasYmbol

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

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

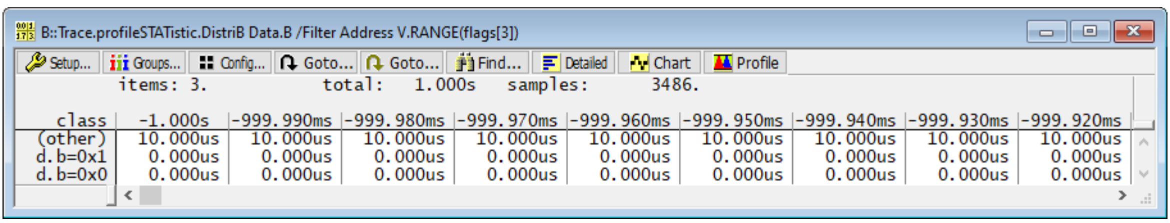
Shows the results of numerical interval analysis in tabular format for the statistical distribution of a selected item or based on the symbolic addresses if no item is specified.

<trace_area> For parameter descriptions and, see **Parameters** under **<trace>.Chart**.

<option> Refer to **<trace>.PROfileSTATistic** for information about the available options.

Example:

```
Trace.profileSTATistic.DistriB Data.B /Filter Address
Var.RANGE(flags[3])
```



See also

■ BMC.PROfileSTATistic.DistriB

Format:	*<trace>*.**PROfileSTATistic.GROUP** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for **groups**. The results only include groups within the program range. Groups for data addresses are not included.

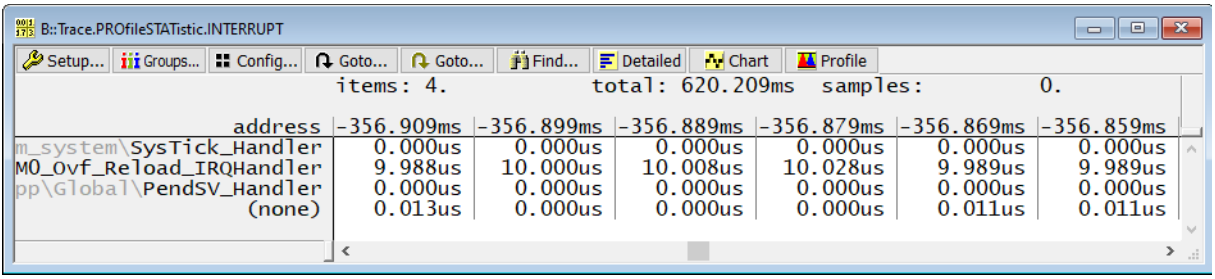
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- <trace>.PROfileSTATistic.AddressGROUP
- BMC.PROfileSTATistic.GROUP

Format:	*<trace>*.**PROfileSTATistic.INTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for interrupts. This command requires **OS-ware tracing**.



<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.INTERRUPT

Format:	*<trace>***.PROfileSTATistic.Line** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for HLL code lines.

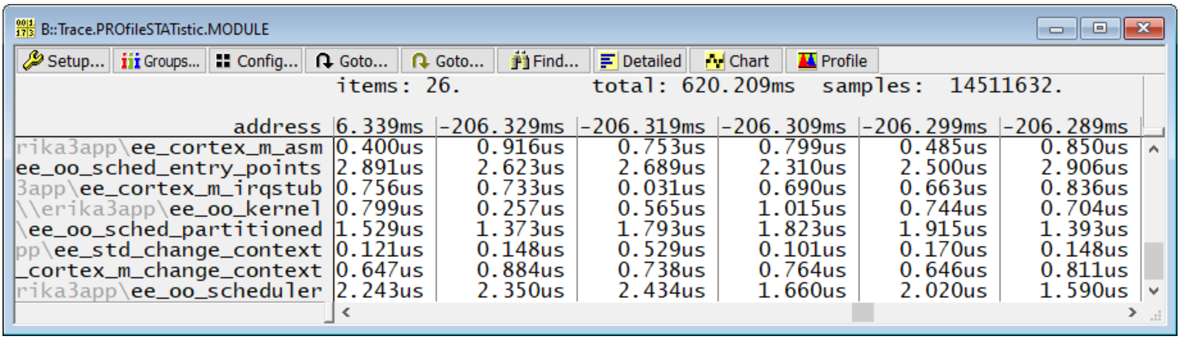
<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.Line

Format:	*<trace>*.**PROfileSTATistic.MODULE** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for the code execution broken down by symbol module. The list of loaded modules can be displayed with **sYmbol.List.Module**.



<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.MODULE

<trace>.PROfileSTATistic.PAddress — Which instr. accessed data address

Format:	*<trace>*.**PROfileSTATistic.PAddress** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format of the instructions that accessed data addresses. You can select a specific address using the **/Filter** option.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

Example:

```
; display a profile statistic of all addresses that accessed mstatic1
Trace.PROfileSTATistic.PAddress /Filter sYmbol mstatic1
```

<trace>.PROfileSTATistic.PROGRAM — Statistical analysis for programs

Format:	*<trace>*.**PROfileSTATistic.PROGRAM** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for the code execution broken down by loaded object file (program). The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.PROGRAM

<trace>.PROfileSTATistic.PsYmbol   Which functions accessed data address

Format:	*<trace>*.**PROfileSTATistic.PsYmbol /Filter Address** *<address>* [**/***<option>*]

Shows the results of numerical interval analysis in tabular format of the functions that accessed data addresses. You can select a specific address using the **/Filter** option.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

Example:

```
; display a profile statistic of all function that accessed mstatic1
Trace.PROfileSTATistic.PsYmbol /Filter sYmbol mstatic1
```

<trace>.PROfileSTATistic.RUNNABLE   Statistical analysis for runnables

Format:	*<trace>*.**PROfileSTATistic.RUNNABLE** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format 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>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

On TriCore AURIX there's a solution available for the Vector AUTOSAR tools that uses an automated instrumentation to trace runnables on all cores with minimum overhead. See *~~/demo/env/vector/rte_profiling*.

Otherwise, all functions that start an AUTOSAR "Runnable" have to be marked with the command **sYmbol.MARKER.Create RUNNABLESTARTPLUSSTOP**. Please refer to **"Trace Export for Third-Party Timing Tools"** (app_timing_tools.pdf) for more information.

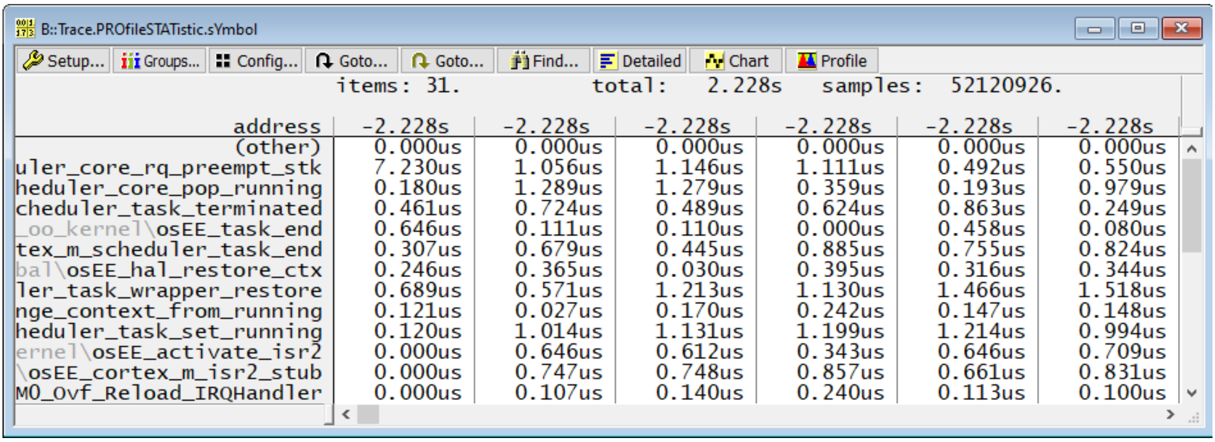
See also

- BMC.PROfileSTATistic.RUNNABLE

<trace>.PROfileSTATistic.sYmbol — Statistical analysis for symbols

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

Shows the results of numerical interval analysis in tabular format for different debug symbols.



address	-2.228s	-2.228s	-2.228s	-2.228s	-2.228s	-2.228s
(other)	0.000us	0.000us	0.000us	0.000us	0.000us	0.000us
uler_core_rq_preempt_stk	7.230us	1.056us	1.146us	1.111us	0.492us	0.550us
heduler_core_pop_running	0.180us	1.289us	1.279us	0.359us	0.193us	0.979us
cheduler_task_terminated	0.461us	0.724us	0.489us	0.624us	0.863us	0.249us
_oo_kernel\osEE_task_end	0.646us	0.111us	0.110us	0.000us	0.458us	0.080us
tex_m_scheduler_task_end	0.307us	0.679us	0.445us	0.885us	0.755us	0.824us
bal\osEE_hal_restore_ctx	0.246us	0.365us	0.030us	0.395us	0.316us	0.344us
ler_task_wrapper_restore	0.689us	0.571us	1.213us	1.130us	1.466us	1.518us
nge_context_from_running	0.121us	0.027us	0.170us	0.242us	0.147us	0.148us
heduler_task_set_running	0.120us	1.014us	1.131us	1.199us	1.214us	0.994us
ernel\osEE_activate_isr2	0.000us	0.646us	0.612us	0.343us	0.646us	0.709us
\osEE_cortex_m_isr2_stub	0.000us	0.747us	0.748us	0.857us	0.661us	0.831us
M0_Ovf_Reload_IRQHandler	0.000us	0.107us	0.140us	0.240us	0.113us	0.100us

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

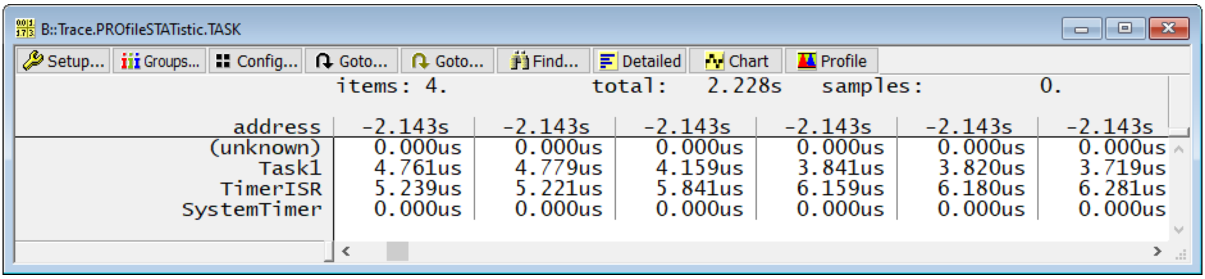
See also

- BMC.PROfileSTATistic.sYmbol

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

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

Shows the results of numerical interval analysis in tabular format for OS tasks. This command requires **OS-ware tracing**.



<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.TASK
- ▲ 'CPU Load Measurement' in 'Application Note Profiling on AUTOSAR CP with ARTI'

<trace>.PROfileSTATistic.TASKINFO — Context ID special messages

Format: *<trace>*.**PROfileSTATistic.TASKINFO** [*<trace_area>*] [**/***<option>*]

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

Displays a profile statistic of special messages written to the Context ID register for ETM trace. The range of special values has to be reserved with the **ETM.ReserveContextID** command. These special values are then not interpreted for task switch or memory space switch detection.

This can be used for cores without data trace to pass data by the target application to the trace tool by writing to the ContextID register.

See also

■ BMC.PROfileSTATistic.TASKINFO

<trace>.PROfileSTATistic.TASKINTR Statistical analysis for ISR2 (ORTI)

Format:	*<trace>*.**PROfileSTATistic.TASKINTR** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows the results of numerical interval analysis in tabular format for ORTI based ISR2. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

■ BMC.PROfileSTATistic.TASKINTR

▲ 'Trace Features' in 'OS Awareness Manual OSEK/ORTI'

Format:	*<trace>***.PROfileSTATistic.TASKKernel** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Numerical interval analysis in tabular format for results of **Trace.STATistic.TASKKernel**. This command requires **OS-ware tracing**.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.TASKKernel

<trace>.PROfileSTATistic.TASKORINTERRUPT Interrupts and tasks

Format:	*<trace>***.PROfileSTATistic.TASKORNTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Numerical interval analysis in tabular format for tasks and interrupts. This command requires **OS-ware tracing**.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

- BMC.PROfileSTATistic.TASKORINTERRUPT

<trace>.PROfileSTATistic.TASKSRV Analysis of OS service routines

Format:	*<trace>*.**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 in tabular format. 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>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

■ BMC.PROfileSTATistic.TASKSRV

<trace>.PROfileSTATistic.TASKVSINTERRUPT Interrupted tasks

Format:	*<trace>*.**PROfileSTATistic.TASKVSINTERRUPT** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Numerical interval analysis in tabular format for tasks that were interrupted by interrupt service routines. This command requires **OS-ware tracing**.

<trace_area>	For parameter descriptions and, see **Parameters** under **<trace>.Chart**.
<option>	Refer to **<trace>.PROfileSTATistic** for information about the available options.

See also

■ BMC.PROfileSTATistic ■ ETA.PROfileSTATistic ■ MIPS.PROfileSTATistic

See also

- <trace>.PROTOcol.Chart
- <trace>.PROTOcol.EXPORT
- <trace>.PROTOcol.list
- <trace>.PROTOcol.PROfileSTATistic
- <trace>.PROTOcol.Chart
- <trace>.PROTOcol.EXPORT
- <trace>.PROTOcol.list
- <trace>.PROTOcol.PROfileSTATistic
- IProbe.state
- <trace>.PROTOcol.Draw
- <trace>.PROTOcol.Find
- <trace>.PROTOcol.PROfileChart
- <trace>.PROTOcol.STATistic
- <trace>.PROTOcol.Draw
- <trace>.PROTOcol.Find
- <trace>.PROTOcol.PROfileChart
- <trace>.PROTOcol.STATistic

<trace>.PROTOcol.Chart — Graphic display for user-defined protocol

Format:	*<trace>***.PROTOcol.Chart** *<protocol>* *<parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **Track** **RecScale** **TimeScale** **TimeZero** **TimeREF** **Filter**
<items>:	**%***<format>* *<line>* **DEFault** | **ALL** **TIme.Back** | **TIme.Fore** | **TIme.REF** **TIme.Zero** | **TIme.Trigger** **SyncClock** **SPARE**

<parlist> Refer to **“Protocol specific Options”**, page 351.

Options:

FILE Display trace memory contents loaded with the command **Trace.FILE.**

Track Track other trace list window (tracks to record number or time)

RecScale	Record Scaling
TimeScale	Timed Scaling
Filter	

Items:

TIme.REF	Time marker, relative to reference point
TIme.Trigger	Time marker, relative to trigger point
TIme.Zero	Time marker, relative to global reference
SyncClock	Synchronous clock event

Example: Display the user-defined protocol “proto1” on line x.0

```
PP::Analyzer.PROTOcol.Chart proto1 x.0
```

See also

- <trace>.PROTOcol

Format:	*<trace>*.**PROTOcol.Draw** *<protocol> <parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **Track** **RecScale** **TimeScale** **TimeZero** **TimeREF** **Filter**
<items>:	**%***<format>* *<line>* **DEFault** | **ALL** **TIme.Back** | **TIme.Fore** | **TIme.REF** **TIme.Zero** | **TIme.Trigger** **SyncClock** **SPARE**

<parlist> Refer to **"Protocol specific Options"**, page 351.

Options:

FILE Display trace memory contents loaded with the command **Trace.FILE.**

Track Track other trace list window
(tracks to record number or time)

RecScale Record Scaling

TimeScale Timed Scaling

Filter Filter the described item.

Items:

TIme.REF Time marker, relative to reference point

TIme.Trigger Time marker, relative to trigger point

TIme.Zero	Time marker, relative to global reference
SyncClock	Synchronous clock event

Example: Display the user-defined protocol “proto1” on line x.0

```
PP::Analyzer.PROTOcol.Draw proto1 x.0
```

See also

■ <trace>.PROTOcol

<trace>.PROTOcol.EXPORT Export trace buffer for user-defined protocol

Format:	*<trace>*.**PROTOcol.EXPORT** *<protocol>* *<parlist>* *<file>* [*<range>* …]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**

<parlist>	Refer to **“Protocol specific Options”**, page 351.

Example: Export the user-defined protocol “proto1” on line x.0 to test.lst

```
PP::Analyzer.PROTOcol.EXPORT proto1 x.0 test.lst
```

See also

■ <trace>.PROTOcol

Format:	*<trace>*.**PROTOcol.Find** *<protocol>* *<parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **Back** **NoFind**

<parlist> Refer to **"Protocol specific Options"**, page 351.

Options:

FILE Display trace memory contents loaded with the command **Trace.FILE.**

Back Search back

NoFind

Example: Find in the user-defined protocol "proto1" on line x.0

```
PP::Analyzer.PROTOcol.Find proto1 x.0
```

See also

- <trace>.PROTOcol
- <trace>.PROTOcol.list
- <trace>.PROTOcol

Format:	*<trace>*.**PROTOcol.list** *<protocol> <parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **Track**
<items>:	**%***<format>* *<line>* **DEFault** | **ALL** **TIme.Back** | **TIme.Fore** | **TIme.REF** **TIme.Zero** | **TIme.Trigger** **SyncClock** **SPARE**
<format>:	**Hex** | **Decimal** | **BINary** | **Ascii** **Timing** | **HighLow** **LEN** *<size>* **TimeAuto** | **TimeFixed**

<parlist> Refer to **"Protocol specific Options"**, page 351.

Options:

FILE	Display trace memory contents loaded with the command **Trace.FILE**.
Track	Track other trace list window (tracks to record number or time)

Formats:

Timing	Display single bits as vertical timing
HighLow	Display single bits as HIGH/LOW value
Hex	Display single bytes in hex values
Decimal	Display single bytes in decimal values
BINary	Display single bytes in binary values
Ascii	Display single bytes as ascii characters

LEN *<size>*	Specify the width of non numeric fields (e.g. symbols)
TimeAuto	The unit of time is selected automatically.
TimeFixed	The displayed unit of time is fixed.

Items:

DEFault	Default selections (see list below)
ALL	Select all recorded data (superset of DEFault)
TIme	Time marker (default Time.Fore)
TIme.Back	Time marker, relative time to previous record
TIme.Fore	Time marker, relative time to next record
TIme.REF	Time marker, relative to reference point
TIme.Trigger	Time marker, relative to trigger point
TIme.Zero	Time marker, relative to global reference
SyncClock	Synchronous clock event
SPARE	Displays an empty block

Example: Displays the user-defined protocol “proto1” on line x.0

```
PP::Analyzer.PROTOcol.list proto1 x.0
```

```
; JTAG <tck> <tms> <tdi> <tdo> <trst> <initstate>
; when the sampling is started the JTAG state machine is in state
; run-test/idle

Trace.PROTOcol.list JTAG X.8 X.9 X.4 X.12 X.14 run-test/idle

; CAN <canline> <...> (see "Options for CAN" for details)

Trace.PROTOcol.list CAN X.7 NominalFrequency 1.0MHz DEFault

; USB <+signal> <-signal>

Trace.PROTOcol.list USB X.17 X.18

; I2C <scl> <sda>

Trace.PROTOcol.list I2C X.22 X.23

; asynchronous communication interface
; ASYNC <asyline> <frequency> +|- <parity> <length> <stopbit>

Trace.PROTOcol.list ASYNC X.6 3600. + EVEN 7 1STOP STRING
Trace.PROTOcol.list ASYNC X.5 2400. - NONE 5 2STOP CHAR

; special protocols
; TRACE32 offers a API that allows to use special, customer specific
; protocols

Trace.PROTOcol.list protojtag.dll X.4 X.12 X.14

; examples for special protocols are provided under ~~/demo/proto
```

See also

- <trace>.PROTOcol
- <trace>.PROTOcol.STATistic
- <trace>.PROTOcol
- <trace>.PROTOcol.Find
- <trace>.REF

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**PROTOcol.PROfileChart** *<protocol> <parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **Track** **RecScale** **TimeScale** **TimeZero** **TimeREF** **Filter**
<items>:	**%***<format>* *<line>* **DEFault** | **ALL** **TIme.Back** | **TIme.Fore** | **TIme.REF** **TIme.Zero** | **TIme.Trigger** **SyncClock** **SPARE**

<parlist>	Refer to **"Protocol specific Options"**, page 351.

Options:

FILE	Display trace memory contents loaded with the command **Trace.FILE.**
Track	Track other trace list window (tracks to record number or time)
RecScale	Record Scaling
TimeScale	Timed Scaling
Filter	

Items:

TIme.REF	Time marker, relative to reference point
TIme.Trigger	Time marker, relative to trigger point
TIme.Zero	Time marker, relative to global reference
SyncClock	Synchronous clock event

See also

- <trace>.PROTOcol

<trace>.PROTOcol.PROfileSTATistic — Profile chart for user-defined protocol

Format:	*<trace>*.**PROTOcol.PROfileSTATistic** *<protocol>* *<parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **Track** **RecScale** **TimeScale** **TimeZero** **TimeREF** **Filter**
<items>:	**%***<format>* *<line>* **DEFault** | **ALL** **TIme.Back** | **TIme.Fore** | **TIme.REF** **TIme.Zero** | **TIme.Trigger** **SyncClock** **SPARE**

<parlist>	Refer to **"Protocol specific Options"**, page 351.

Options:

FILE	Display trace memory contents loaded with the command **Trace.FILE.**

Track	Track other trace list window (tracks to record number or time)
RecScale	Record Scaling
TimeScale	Timed Scaling
Filter	

Items:

TIme.REF	Time marker, relative to reference point
TIme.Trigger	Time marker, relative to trigger point
TIme.Zero	Time marker, relative to global reference
SyncClock	Synchronous clock event

See also

■ <trace>.PROTOcol

Format:	*<trace>*.**PROTOcol.STATistic** *<protocol>* *<parlist>* [*<items>* …] [**/***<option>*]
<protocol>:	**JTAG** | **CAN** | **USB** | **I2C** | **I2S** | **ASYNC** | **SWDP** | **SPI** | **PROBEUSB**
<option>:	**FILE** **BEFORE** **AFTER** **List** **Filter** **Accumulate** **INCremental** **FULL**
<items>:	**%***<format>* *<line>* **DEFault** | **ALL** **TIme.Back** | **TIme.Fore** | **TIme.REF** **TIme.Zero** | **TIme.Trigger** **SyncClock** **SPARE**
<format>:	**Hex** | **Decimal** | **BINary** | **Ascii** **Timing** | **HighLow** **LEN** *<size>* **TimeAuto** | **TimeFixed**

<parlist>	Refer to **"Protocol specific Options"**, page 351.

Options:

FILE	Display trace memory contents loaded with the command **Trace.FILE.**
Track	Track other trace list window (tracks to record number or time)

Example: Display statistics in the user-defined protocol "proto1" on line x.0

```
PP::Analyzer.PROTOcol.STATistic proto1 x.0
```

See also

- <trace>.PROTOcol
- <trace>.PROTOcol.list
- <trace>.PROTOcol

Protocol specific Options

Options for ASYNC

<parlist>:	*<signal>* [*<baudrate>* [*<polarity>* [*<parity>* [*<bits>* [*<stopbits>* [*<disp>*]]]]]]
<baudrate>:	**1.** … **1000000.**
<polarity>:	**+** | **-**
<parity>:	**NONE** | **ODD** | **EVEN**
<bits>:	**5** | **6** | **7** | **8**
<stopbits>:	**1STOP** | **2STOP**
<disp>:	**CHAR** | **STRING**

<parlist>:	*<signal>* [*<setting>* …]
<setting>:	**NominalFrequency** *<frequency>* **DataFrequency** *<frequency>* **NominalTiming** *<tq> <PropSeg> <PhaseSeg1> <PhaseSeg2> <SJW>* **DataTiming** *<tq> <PropSeg> <PhaseSeg1> <PhaseSeg2> <SJW>* **Filter** *<name> <frameType> <id> <byteOffset> <size> <endian> <format>* **DRAWRange** *<min> <max>* **Level** *<level>*
<frequency>:	*<frequency, e. g. 1MHz>*
<tq>:	*<time, e. g. 0.1us>*
<PropSeg>: *<PhaseSeg1>* *<PhaseSeg2>* *<SJW>*: *<id>*: *<byteOffset>*: *<min>*: *<max>*:	*<integer>* *<integer>* *<integer>* *<integer>* *<integer>* *<integer>* *<integer>* *<integer>*
<id>:	*<string>*
<frameType>:	**Base** | **Extended**
<size>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **PByte** | **HByte** | **SByte**
<endian>:	**LE** | **BE**
<format>:	**Decimal** | **DecimalU** | **Hex**
<level>:	**PCS** | **BITS** | **FIELDS** | **FRAMES** | **FILTERS**

CAN decoder for buses compliant with ISO 11898-1:2015 (CAN-FD). To record CAN, the probe must be connected to the RX line of a CAN transceiver. Do not connect the probe directly to the CAN lines.

Most *<setting>*s are optional with the following constraints:

- Either **NominalFrequency** or **NominalTiming** must always be specified.
- For the **<trace>.PROTOcol.Draw** command, at least one **Filter** must be specified.

NominalFrequency **DataFrequency**	Specify the baud rate. This sets up a sample point at 70 % of a period; use these commands if you do not know or care about the exact timing parameters of the CAN bus.
NominalTiming **DataTiming**	Specify the exact timing parameters as defined in the CAN specification. For *<tq>*, specify the effective time of a time quantum derived from the base clock and the baud rate prescaler. Prefer to use this family of options over the Frequency options, at least for CAN-FD buses.
Filter	Add a filter to decode fixed-format frames. For every data frame with matching type (i. e. base or extended) and ID, extract the specified data. The number of filters is limited only by the maximum length of a PRACTICE command. The *<name>* is only used to identify the filter in the List and EXPORT commands.
DrawRange	Specify upper/lower bounds for Draw window. This setting is only relevant for the **<trace>.PROTOcol.Draw** command. If not given, this option is automatically chosen to encompass the full range of all defined filters. Note that the draw range is limited to either signed or unsigned 32-bit integers.
Level	Specify the display level. For the **<trace>.PROTOcol.list** command, this is the default level and can be changed with the More and Less buttons. For the **<trace>.PROTOcol.EXPORT** command, only **FRAMES** and **FILTERS** are valid. The available levels are as follows: • **PCS**: Phases of the Physical Coding Sub-layer • **BITS**: Decoded bits of the protocol • **FIELDS**: Decoded fields of the protocol • **FRAMES**: Complete CAN frames • **FILTERS**: Data extracted with the **Filter** setting

NOTE:	Prior to R.2020.09, TRACE32 included a CAN decoder that was not compatible with CAN-FD. This old decoder had a different command line syntax. To use the old decoder, type *<trace>*.PROTOcol.*<sub_cmd>* CANLEGACY *<...>*.

Options for I2C

<parlist>:	*<scl> <sda>* [*<glitch_ns>*] (All commands except <trace>.PROTOcol.FIND) *<scl> <sda> <data>* [*<glitch_ns>*] (<trace>.PROTOcol.FIND)
<sck>:	*<signal>*
<sda>:	*<signal>*
<glitch_ns>:	*<integer>*
<data>:	*<integer>*

If not specified, the default glitch filter is 50 ns.

Options for I2S

<parlist>:	*<sck> <sd> <ws>* [*<glitch_ns>*] (All commands except <trace>.PROTOcol.FIND) *<sck> <sd> <ws> <data>* [*<glitch_ns>*] (<trace>.PROTOcol.FIND)
<sck>:	*<signal>*
<sd>:	*<signal>*
<ws>:	*<signal>*
<glitch_ns>:	*<integer>*
<data>:	*<integer>*

<parlist>:	*<tck> <tms> <tdi> <tdo>* [*<trst>*] [*<initial_state>*]
<tck>: *<tms>*: *<tdi>*: *<tdo>*: *<trst>*:	*<signal>*
<initial_ state>:	**Exit2-DR** **Exit1-DR** **Shift-DR** **Pause-DR** **Select-IR-scan** **Update-DR** **Capture-DR** **Select-DR-scan** **Exit2-IR** **Exit1-IR** **Shift-IR** **Pause-IR** **Run-Test/Idle** **Update-IR** **Capture-IR** **Test-Logic-Reset**

<parlist>:	*<+signal> <-signal>* [*<state>*]
<+signal>:	*<signal>*
<-signal>:	*<signal>*
<state>:	**BUSRESET** **GAP** **SYNC** **EOP** **ERRORS** **SOF** **#SETUP** **#PRE** **#IN** **#OUT** **#ACK** **#NACK** **#STALL** **#DATA0** **#DATA1** **#OTHER**

Format:	*<trace>*.**REF** [*<time>* | *<record>* | "*<trace_bookmark>*"]
<option>:	**FILE**

Sets the reference point for time measurements using the **TIme.REF** column of the **Trace.List** window. The default reference point is always the last record in trace memory. The reference point can also be set via context menu entry **Set Ref** in **Trace.List**, **Trace.Chart**, **Trace.Timing** etc. windows.

<time>	Sets the reference point to the global ZERO point. If the time for each trace event is calculated based on timestamps generated by the processor, the global ZERO point is at the start of the trace recording currently stored in the trace buffer. If the time for each trace event is based on timestamps generated by the trace module, the global ZERO point is set to the start of the first debug session after the start of TRACE32 PowerView.
<record>	Sets the reference point to the time index of the specified record number.
<trace_bookmark>	Sets the reference point to the time index of the specified bookmark location. You can create trace bookmarks with the **<trace>.BookMark** command.

Examples:

```
Trace.REF +2000.              ; set reference to record +2000

Trace.REF 100us               ; set ref. point to absolute time

Trace.REF "MyRef"             ; set ref. point to bookmark "MyRef"
```

See also

- <trace>.GOTO
- <trace>.PROTOcol.list
- <trace>.Timing
- <trace>.View
- IProbe.state
- RunTime
- RunTime.state
- Analyzer.REF()

<trace>.RESet Reset command

Format:	*<trace>*.**RESet**

Resets the trace unit to its default settings.

See also

- IProbe.state
- RunTime
- RunTime.state

Format:	*<trace>*.**SAVE** *<file>* [*<trace_area>*] [{**/***<options>*}]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FlowTrace** **BusTrace** **ZIP** **PACK** **NoCompress**

The trace memory contents are stored to the selected file. What is actually saved to the file depends on the **Trace.Mode**:

- **Trace.Mode FlowTrace**: Trace raw data plus decompressed addresses, data and op-codes are saved.

  If the program and data flow is output by the CPU in a compressed format, it is decompressed before saving it to a file for postprocessing. By reading the source code information the addresses, data value and op-codes are decompressed.

- All other settings for **Trace.Mode**: Only the trace raw data are saved into *<file>* if the information from the external busses, ports etc. are recorded to the trace buffer.

Parameters

<file>	The default extension of the file name is ***.ad**. For some TRACE32 devices additional setting are saved to *<file>*: • PowerProbe (**Trace.METHOD Probe**) All Probe settings and all **NAME** settings are saved to *<file>*. • PowerIntegrator (**Trace.METHOD Integrator**) All **Integrator** settings and all **NAME** settings are saved to *<file>*.

<trace_area>	
<trace_bookmark>	Specify two *<trace_bookmarks>* to define the *<trace_area>* you want to save to the file. • If you specify only one *<trace_bookmark>*, then the *<trace_area>* ranges from that bookmark up to the end of the trace recording. See example.
<record_range>	You need to specify two record numbers to define the *<trace_area>* you want to save to the file. See example.
<record>	Specify two record numbers to define the *<trace_area>* that you want to save to the file. • If you specify only one *<record>*, then the *<trace_area>* ranges from that record number up to the end of the trace recording. See example.
<time_range>	You need to specify two absolute timestamps that are based on the zero time to define the *<trace_area>* you want to save to file. See example.
<time>	Specify two absolute timestamps that are based on the zero time to define the *<trace_area>* you want to save to the file. • If you specify only one absolute timestamp, then the *<trace_area>* ranges from that timestamp up to the end of the trace recording.

Options

FlowTrace	Obsolete.
BusTrace	Save only the trace raw data, if a flow trace is used. This option is helping if a decompression of the program and data trace information is not possible.
PACK	Save the trace contents is a compact way. **PACK** is less effective and slower then **ZIP**. It is only recommended if the option **ZIP** is not available.
ZIP	Save the trace contents is a compact way.
NoCompress	Obsolete.

Example for <trace_bookmark>

[Parameter Descr.]

```
Trace.List                              ; display trace listing

Trace.BookMark "First" -123366.         ; mark the trace record -123366.
                                        ; with the bookmark "First"

Trace.BookMark "Last" -36675.           ; mark the trace record -36675.
                                        ; with the bookmark "Last"

BookMark.List                           ; list all bookmarks

Trace.SAVE testb "First" "Last"         ; save trace contents between
                                        ; bookmarks "First" and "Last"
                                        ; to the file testb
```

Example for <record_range>

[Parameter Descr.]

```
; display trace listing
Trace.List

; save trace contents between record -107032. and record -21243.
; to the file testr
Trace.SAVE testr (-107032.)--(-21243.)
```

Example for <record> <record2>

[Parameter Descr.]

```
Trace.List                              ; display trace listing

Trace.SAVE testv -107032. -21243.       ; save trace contents between
                                        ; record -107032. and record
                                        ; -21243.to the file testv
```

Example for <time_range>

[Parameter Descr.]

```
; display trace listing
Trace.List TIme.ZERO DEFault

; save trace contents between the point of time 4.us and the point of
; time 1.952ms to the file testt
Trace.SAVE testt 4.us--1.952ms
```

More Examples

```
Trace.SAVE test4                     ; save trace contents to the file
                                     ; test4

QUIT                                 ; end TRACE32

                                     ; off-line postprocessing of the
                                     ; trace contents e.g. with a
                                     ; TRACE32 Instruction Set Simulator

Trace.LOAD test4                     ; load the saved trace contents

Data.LOAD.Elf demo.elf /NoCODE       ; load the symbol information if
                                     ; you like to have HLL information
                                     ; for the trace analysis

Trace.List                           ; display the loaded trace contents
                                     ; as trace listing

Trace.STATistic.Func                 ; perform a function run-time
                                     ; analysis on the loaded trace
                                     ; contents
                                     ;
                                     ; HLL information is required
```

```
; save trace contents to the file test4
Trace.SAVE test4

;…

; use saved trace contents as reference

; load the saved trace contents
Trace.FILE test4

; display the trace contents loaded from the file test4.ad as trace
; listing
Trace.List /FILE

; compare the current trace contents with the trace contents loaded from
; test4.ad with regards to the addresses
Trace.ComPare (-27093.)--(-8986.) Address /FILE
```

See also

- <trace>.EXPORT
- <trace>.LOAD
- <trace>.STREAMSAVE
- IProbe.state
- RunTime
- RunTime.state

▲ 'Release Information' in 'Legacy Release History'

Format: *<trace>*.**SelfArm** [**ON** | **OFF**]
<trace>.**AutoTEST** [**ON** | **OFF**] (deprecated)

Trace.SelfArm ON automatically restarts the trace recording. There are mainly two use cases for this command.

Snapshot without stopping the program execution

If stopping the program execution it not advisable, but you are interested in the target state at a specific point of the program execution, proceed as follows:

1. Display the information of interest on the screen.

   Please make sure to display only information that can be updated while the program execution is running.

2. Use a trigger to specify your point of interest.

3. Activate the self-arm mode.

Whenever the trace recording is stopped by the trigger, all information displayed by TRACE32 is updated before the trace recording is automatically restarted.

Example for ARM11:

```
Data.LOAD.Elf armla.Elf /PlusVM        ; load source code to virtual
                                       ; memory within TRACE32 in order to
                                       ; enable the trace display while
                                       ; program execution is running

Trace.List                             ; display the information of
                                       ; interest

Break.Set sieve /TraceTrigger          ; specify the trigger point

Trace.Mode AutoInit ON                 ; clear the trace buffer and
                                       ; re-activate the trigger
                                       ; before the trace recording
                                       ; is automatically restarted

Trace.Mode SelfArm ON                  ; activate the self-arm mode

Go

;…

Trace.Mode SelfArm OFF                 ; stop automatic restarting of
                                       ; trace recording
```

Automated run-time analysis

To automate an incremental run-time analysis, proceed as follows:

1. Prepare the run-time analysis and open a run-time analysis window.
2. Switch the trace to **Leash** mode.
3. Activate the self-arm mode.

Whenever the trace recording/program execution is stopped because the trace buffer is full, the current trace contents is analyzed and the analysis window is updated correspondingly. Afterwards the program execution is restarted.

Example:

```
Trace.STATistic.Func /ACCUMULATE      ; open a window that performs a
                                      ; continuous nested function
                                      ; run-time analysis

Trace.Mode Leash                      ; switch the trace to Leash mode

Trace.Mode AutoInit ON                ; clear the trace buffer
                                      ; before the trace recording
                                      ; is automatically restarted

Trace.Mode SelfArm ON                 ; activate the self-arm mode

Go

;…

Trace.Mode SelfArm OFF                ; stop automatic restarting of
                                      ; trace recording
```

See also

- <trace>.SnapShot
- IProbe.state

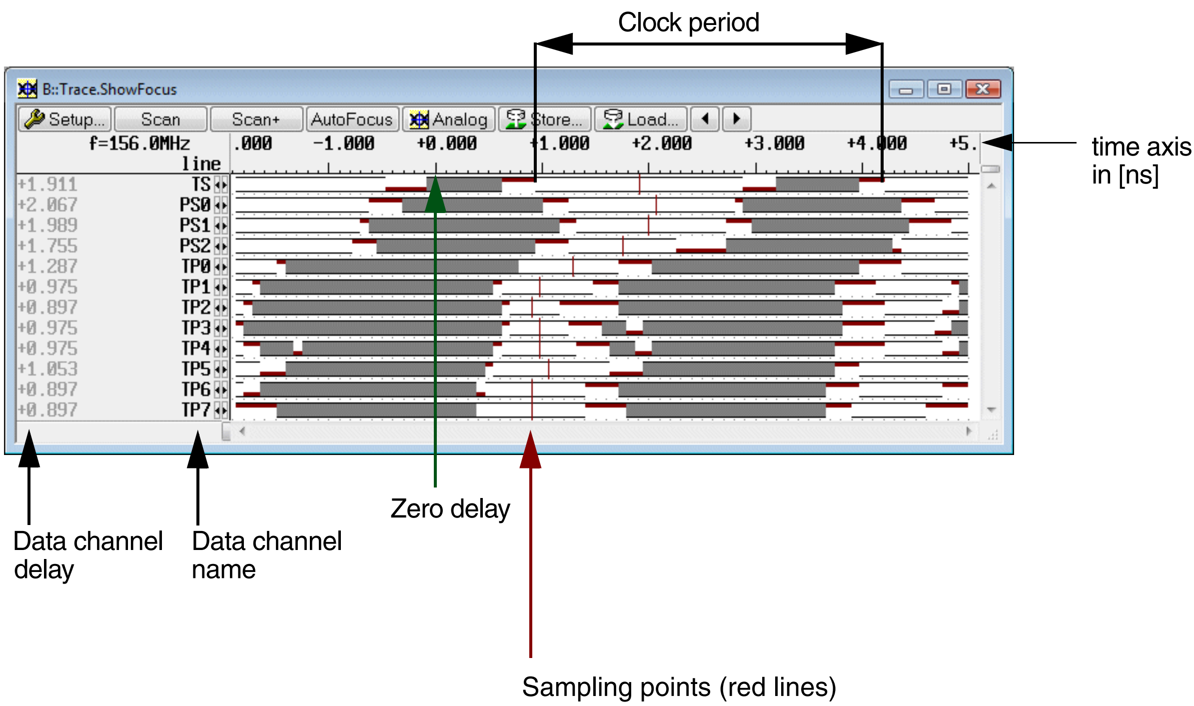
Format:	*<trace>*.**ShowFocus**

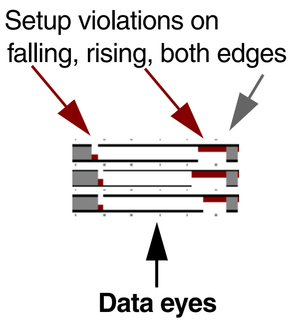
The **Trace.ShowFocus** command displays the data eyes as they are “seen” by a preprocessor with AUTOFOCUS technology resulting from the following commands:

- **Trace.AutoFocus**
- **Trace.TestFocus**

Description of Buttons in the Trace.ShowFocus Window

Setup …	Open **Trace.state** window to configure the trace.
Scan	Perform a **Trace.TestFocus** scan.
Scan+	Perform a **Trace.TestFocus /Accumulate** scan.
AutoFocus	Perform a **Trace.AutoFocus** scan.
Data	Open a **Trace.ShowFocusEye** window.
Clock	Open a **Trace.ShowFocusClockEye** window.
Store …	Save the current AUTOFOCUS configuration to a file (**STOre** *<file>* **AnalyzerFocus**).
Load …	Load an AUTOFOCUS configuration from a file (**DO** *<file>*).
	Move all sampling points 1 * *<time_clock>* to the left.
	Move all sampling points 1 * *<time_clock>* to the right.





In the **Trace.ShowFocus** window the data eyes are white, whereas setup violations are marked as follows:

Setup violation on the rising edge	High red line
Setup violation on the falling edge	Low red line
Setup violation on both edges	Grey bar

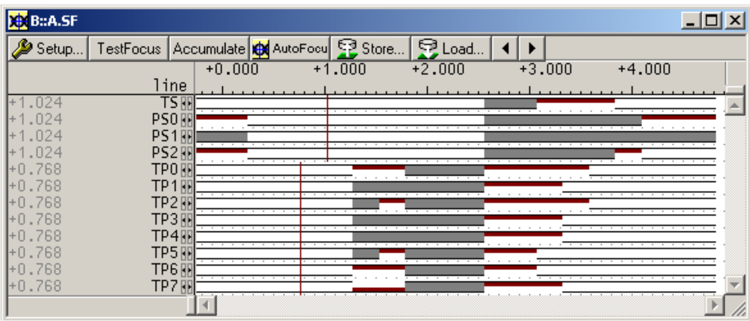
The x-axis of the **Trace.ShowFocus** window corresponds to the time axis, whereas the y-axis corresponds to the data channels of the ETM trace port. In the example above we have an 8-bit ETMv1.x architecture with the channels TRACESYNC (TS), PIPESTAT (PS[3:0]) and TRACEPKT (TP[7:0]).

A preprocessor with AUTOFOCUS technology has programmable delays for the clock channel as well as all data channels. With that in mind the x-axis (time-axis) has the following meaning:

Data delay greater than clock delay	Negative value
Both clock and data delay are zero	Zero
Clock delay greater than data delay	Positive value

In the example above there is a channel to channel skew of more than 1 ns that has been compensated by choosing individual sampling points for each data channel. The time resolution for clock and data channel adjustment is not necessarily the same. In the example the time resolution for data channel adjustment is relatively coarse (500-600 ps), whereas the clock channel can be adjusted in fine delay steps (78 ps). The actual values are functions of voltage, temperature and process. However they are measured for each **Trace.AutoFocus** or **Trace.TestFocus** execution, so the numbers displayed in the **Trace.ShowFocus** window do have a physical meaning (time unit is ns).

The example shows the **Trace.ShowFocus** window as it might appear when using the LA-7991 OTP (see **Preprocessor for ARM-ETM AutoFocus** for details). For the re-programmable version both clock and data delays are typically 270 ps and the **Trace.ShowFocus** window for the same application might look like this:



Trace.ShowFocus as it appears for a re-programmable LA-7991

NOTE:	The NEXUS AutoFocus adapter does not support this feature.

See also

- <trace>.TestFocus
- <trace>.ShowFocusClockEye
- <trace>.TestFocus
- <trace>.TestFocusClockEye
- <trace>.ShowFocusEye
- <trace>.TestFocusEye

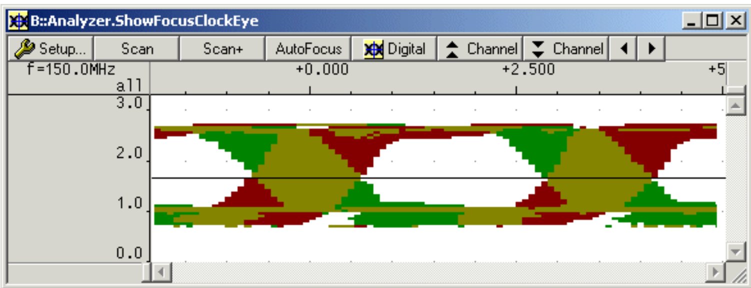
▲ 'Installation' in 'AutoFocus User's Guide'
▲ 'Installation' in 'Arm ETM Trace'

Format:	*<trace>*.**ShowFocusClockEye** [*<channel>*]
<channel>:	**TS** | **PS**[**0**…**2**] | **TP**[**0**…**32**] (ETM V1.x) **TCTL** | **TP**[**0**…**32**] (ETM V3.x)

The command **Trace.ShowFocusClockEye** displays a graph reflecting the clock waveform. Basically it shows data eyes from a different point of view.

The result of the command **Trace.ShowFocusClockEye** shows a single ETM channel or all ETM channels superimposed. Further are:

- X-axis: time range [ns]
- Y-axis: voltage range [V]



Color Legend

White area	Data eye.
Green area	Setup violation on the rising edge.
Red area	Setup violation on the falling edge.
Superimposed area (green and red)	Setup violation on both edges.

Setup ...	Open **Trace.state** window to configure the trace.
Scan	Perform a **Trace.TestFocusEye** scan.
Scan+	Perform a **Trace.TestFocusEye /Accumulate** scan.
AutoFocus	Perform a **Trace.AutoFocus**.
Digital	Open a **Trace.ShowFocus** window scan.
Channel (previous)	Display **Trace.ShowFocusClockEye** for a single trace line (previous).
Channel (next)	Display **Trace.ShowFocusClockEye** for a single trace line (next).
	Move all sampling points 1 * *<time_clock>* to the left.
	Move all sampling points 1 * *<time_clock>* to the right.

Examples

```
Trace.ShowFocusEye                   ; Display data eye with
                                     ; all trace channels superimposed

Trace.ShowFocusEye PS2               ; Display data eye for the
                                     ; trace channel PS2
```

NOTE:	The NEXUS AutoFocus adapter does not support this feature.

See also

■ <trace>.ShowFocus ■ <trace>.ShowFocusEye ■ <trace>.TestFocusEye ■ <trace>.TestFocus
■ <trace>.TestFocusClockEye

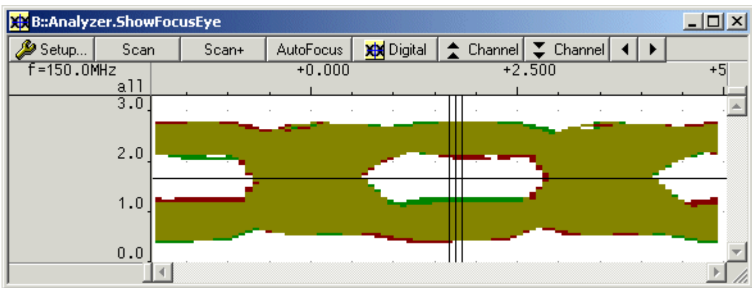
▲ 'Diagnosis' in 'AutoFocus User's Guide'
▲ 'Diagnosis' in 'Arm ETM Trace'

Format:	*<trace>*.**ShowFocusEye** [*<channel>*]
<channel>:	**TS** | **PS**[**0**…**2**] | **TP**[**0**…**32**] (ETM V1.x) **TCTL** | **TP**[**0**…**32**] (ETM V3.x) **LANE[0...n]** (only with serial preprocessor or PowerTrace Serial)

The command **Trace.ShowFocusEye** displays the data eye as it is 'seen' by a preprocessor with AUTOFOCUS technology or PowerTrace Serial resulting from the command **Trace.TestFocusEye**.

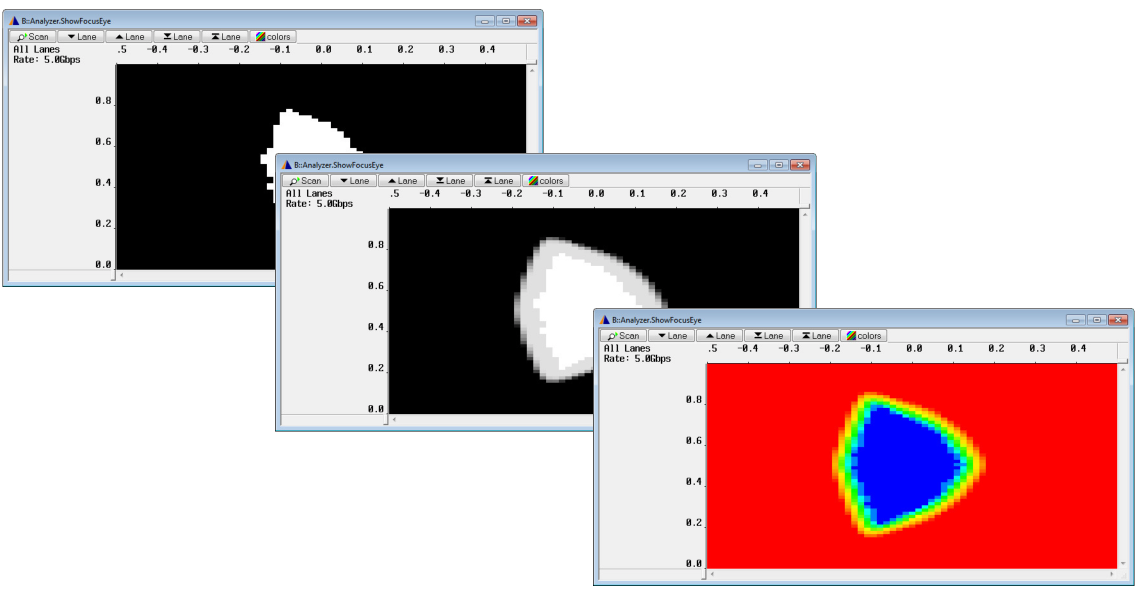
The result of the command **Trace.ShowFocusEye** shows a single trace channel or all trace channels superimposed. The unit of the axis differs for AUTOFOCUS:

- X-axis: time range [ns] or [UI]
- Y-axis: voltage range [V] or [percentage]



and PowerTrace Serial technology:

- X-axis: time range [UI]
- Y-axis: voltage range [percentage of eye height]



Color Legend for AUTOFOCUS

[Back to Top]

White area	Stable data.
Green area	Setup violation on the rising edge.
Red area	Setup violation on the falling edge.
Superimposed area (green and red)	Setup violation on both edges.

Color Legend for PowerTrace Serial

[Back to Top]

White area	Stable data eye.
Black areas	Unstable data.

White area	Stable data eye.
Grey/Black areas	Unstable data.

Dark Blue area	Stable data eye.
Blue/Green/Orange/ Yellow/Red areas	Unstable data.

Descriptions of Buttons in the <trace>.ShowFocusEye Window:

[Back to Top]

Setup …	Open **Trace.state** window to configure the trace.
Scan	Perform a **Trace.TestFocusEye** scan.
Scan+	Perform a **Trace.TestFocusEye /Accumulate** scan.
AutoFocus	Perform a **Trace.AutoFocus** scan.
Digital	Open a **Trace.ShowFocus** window.
Channel (previous)	Display **Trace.ShowFocusEye** for a single trace line (previous).
Channel (next)	Display **Trace.ShowFocusEye** for a single trace line (next).
	Move all sampling points 1 * *<time_clock>* to the left.
	Move all sampling points 1 * *<time_clock>* to the right.

Examples

[Back to Top]

```
Trace.ShowFocusEye                    ; Display data eye with
                                      ; all trace channels superimposed

Trace.ShowFocusEye PS2                ; Display data eye for the
                                      ; trace channel PS2
```

NOTE:	The NEXUS AutoFocus preprocessor does not support this feature.

See also

- <trace>.TestFocus
- <trace>.ShowFocus
- <trace>.TestFocusEye
- <trace>.TestFocusClockEye
- <trace>.ShowFocusClockEye

<trace>.SIZE — Define buffer size

Format:	*<trace>*.**SIZE** [*<size>*]

Sets the *<size>* of trace memory which is used for trace recording. If the command is called with size zero, the trace size will be set to its maximum size. If the configured trace size is larger that the maximum size allowed by the used trace method, then the maximum size is set or an error message is returned.

Reducing the size used for trace recording helps to reduce time needed for trace data download and trace analysis (statistical analysis, trace chart display etc, searching for an event in the trace), because of the smaller amount of recorded data. There is no other benefit besides that.

See also

■ RunTime ■ RunTime.state ❏ Analyzer.RECORDS() ❏ Analyzer.SIZE()
▲ 'Release Information' in 'Legacy Release History'

<trace>.SnapShot — Restart trace capturing once

Format:	*<trace>*.**SnapShot** *<trace>*.**TEST** (deprecated)

Restart the trace capturing. Effectively the same as executing the commands **Trace.OFF**, **Trace.Init** and **Trace.Arm**.

Most often used to restart the trace recording:

- After the trace capturing was stopped by a trigger (e.g. by a TraceTrigger breakpoint).
- When the trace works in **Stack** mode and trace capturing was stopped because the trace buffer was full.

See also

■ IProbe.state ■ <trace>.SelfArm

Format:	*<trace>*.**SPY**

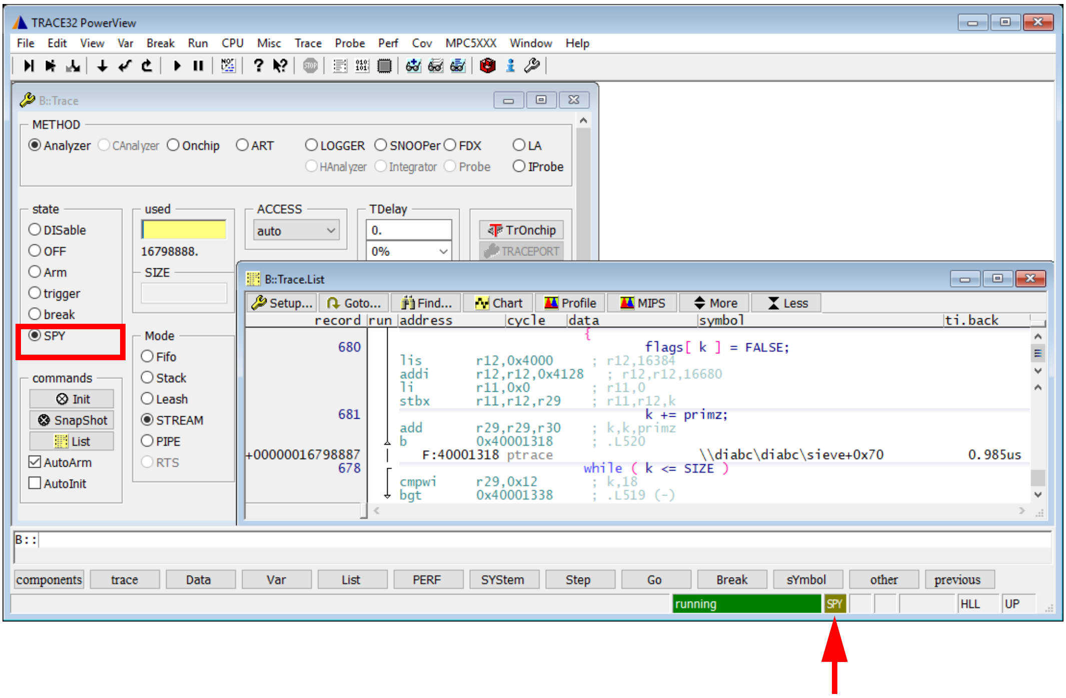
Trace.SPY allows display intermediate trace analysis results while streaming (see **<trace>.Mode STREAM**). If the average data rate at the trace port is high, the analysis time is reduced, and vise versa.

The **Trace.SPY** command requires that trace decoding is possible while the program execution is running. This is possible, if the core architecture in use provides run-time access to memory or if the object code is loaded to the TRACE32 virtual memory. It is recommended to load the object code to the TRACE32 Virtual Memory in any case, because the trace analysis is then faster.

The **Trace.SPY** command only works if the trace is currently in Arm mode.

Example:

```
;…

Data.LOAD.Elf demo.elf /VM              ; copy object code to TRACE32
                                        ; Virtual Memory

Analyzer.Mode STREAM                    ; select trace mode STREAM

Go                                      ; start the program execution

;…

IF Analyzer.STATE()!=1
   PRINT "No switch to SPY mode possible"

Trace.SPY                               ; enable analysis of streaming file

Trace.List

Trace.Arm                               ; switch back to standard recording

;…
```

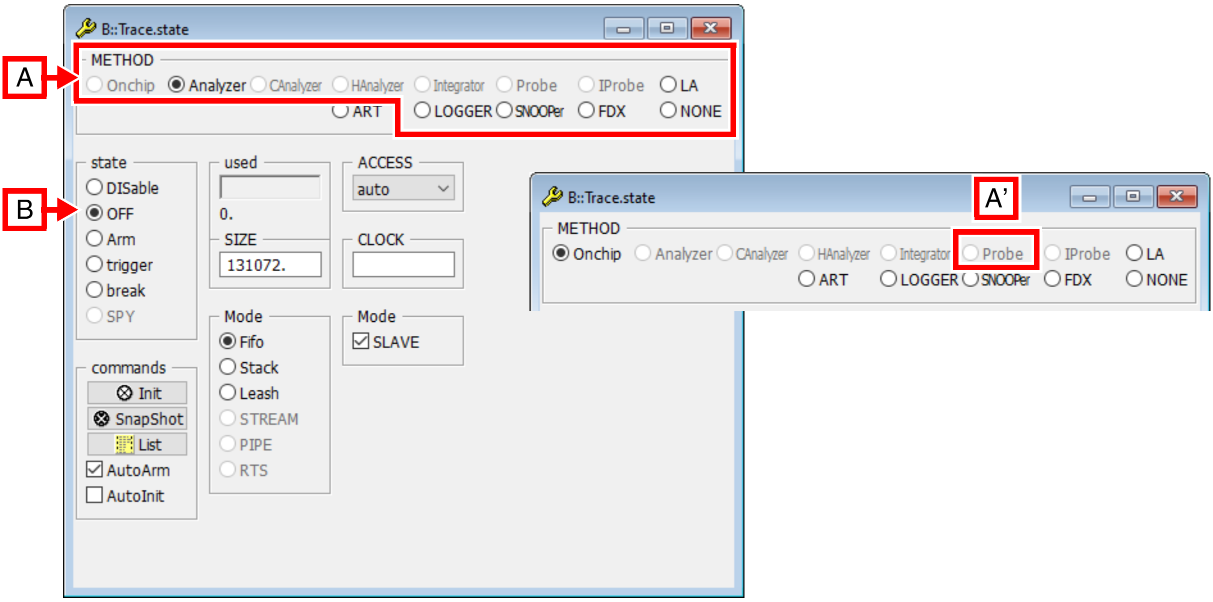


See also

■ IProbe.state ■ Trace

Format:	*<trace>*.**state**

Displays the trace configuration window. The trace methods are displayed at the top of the window. The configuration options below the trace methods adjust to the currently selected trace method, compare screenshot on the left with the screenshot on the right.



A After you have selected the desired trace method (**Trace.METHOD**), you can work with the commands that start with **Trace**. This principle is illustrated in the two PRACTICE script snippets below.

B For descriptions of the commands in the **Trace.state** window, please refer to the **<trace>.*** commands in this chapter. **Example**: For information about **OFF**, see **<trace>.OFF**.

```
Trace.METHOD Analyzer ;Select the trace method, here Analyzer

Trace.List            ;The trace listing now refers to the
                      ;method Analyzer
```

```
Trace.METHOD SNOOPer  ;Select the trace method, here SNOOPer

Trace.List            ;The trace listing now refers to the method SNOOPer
```

See also

- Trace.METHOD
- FDX.CLEAR
- FDX.CLOSE
- FDX.DISableChannel
- FDX.ENableChannel
- FDX.InChannel
- FDX.METHOD
- FDX.Out
- FDX.OutChannel
- FDX.PipeREAD
- FDX.PipeWRITE
- FDX.Rate
- FDX.READ
- FDX.TImestamp
- FDX.TraceChannel
- FDX.WRITE

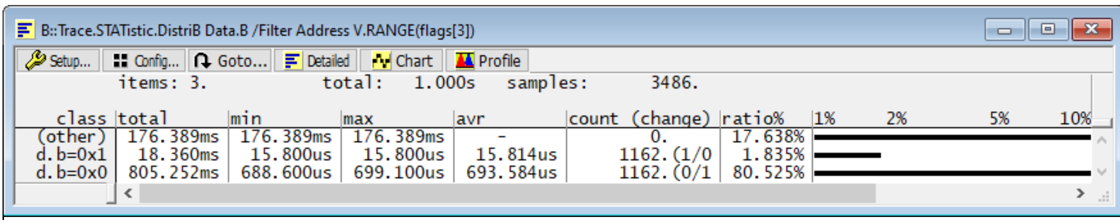
▲ 'Trace Functions' in 'General Function Reference'
▲ 'Release Information' in 'Legacy Release History'

The **<trace>.STATistic** commands can be used for statistical analysis based on the information sampled to the trace buffer.

In contrast to the performance analyzer (**PERF** commands), the statistical analysis commands provide a higher precision and much more information about the analyzed item, but since the size of the trace buffer is limited, the observation time is limited. Statistic evaluations can be made after the trace memory stops sampling.

Example for TRACE32-ICD and TRACE32-PowerTrace:
IIf no selective tracing is possible, use the option /**Filter** to filter out the event of interest.

```
Go
Break
Trace.STATistic.DistriB Data.B /Filter Address Var.RANGE(flags[3])
```



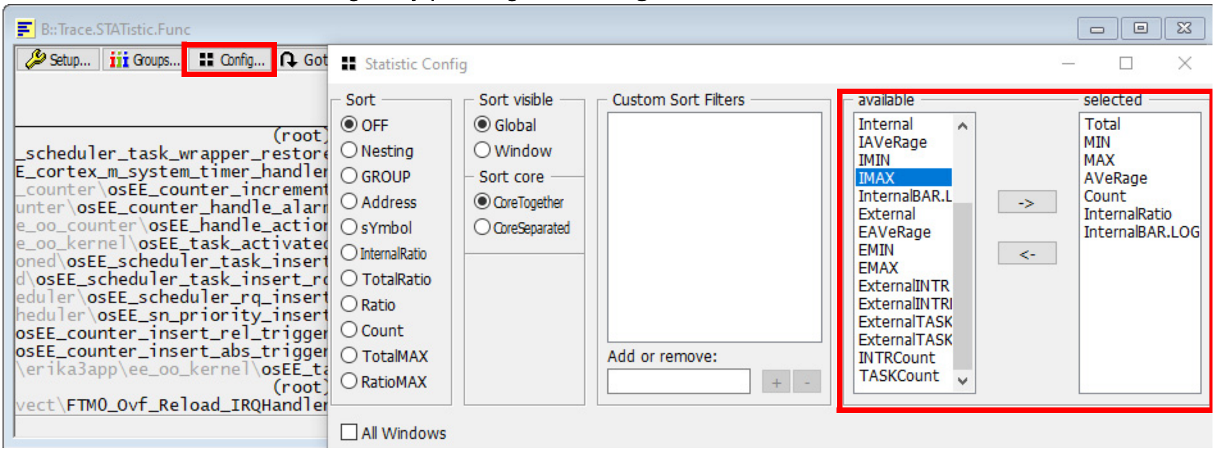
If selective tracing is possible, use the **/TraceEnable** filter to extend the observation time:

```
Break.Set InterruptEntry /Program /TraceEnable
Go
Break
Trace.STATistic.DIStance /Filter Address InterruptEntry
```

Parameters

List items

The <*list_items*> can be arranged by pushing the **Config** button in the **<trace>.STATistic** window.



The table below include a description of the **List items**. Please note that not all **List items** are available for all **<trace>.STATistic** commands.

DEFault	Default trace statistics display.
NAME	Event name.
GROUP	Group name assigned by **GROUP** commands.
TASK	Task name for event.
Total	Total time within the event.
TIme	Total time the event was true. For function nesting analysis, the time spent in interrupt routines is by default taken out of the measurement.
TotalRatio	Ratio of time spent within the event over the complete measurement time. If nesting analysis is used (e.g. **<trace>.STATistic.Func**), then called subroutines are included.
TotalBAR.log, TotalBAR.LINear	Graphical display of the ratio (linear or logarithmic).
Count	Number of occurrences of the event.
MIN, MAX	Minimum and maximum time the event was true.
AVeRage	Average time the event was true.

Internal	Time spent within the event.
IAVeRage	Average time spent in the event.
IMIN, IMAX	Shortest/longest time spent in the event.
InternalRatio	Ratio of time spent in the event.
InternalBAR.log, Internal.LINear	Graphical display of the ratio (linear or logarithmic) spent in the event.
External	Time spent within sub functions.
EAVeRage	Average time spent within sub functions.
EMIN, EMAX	Shortest/longest time spent within sub functions.
ExternalTASK	Total time in other tasks.
ExternalTASKMAX	Max time one function pass was interrupted by other tasks.
INTRCount	Number of interrupts that occurred during the function run-time.
TASKCount	Number of other tasks that interrupted the function.
Ratio, BAR.log, BAR.LINear	Ratio of time spent in events to total measurement time in percent and as graphical bars.
CountRatio, CountBAR.log, CountBAR.LINear	Ratio of count to total count in percent and as graphical bars.
TotalMIN, TotalMAX	Shortest/longest time period in the task.
RatioMIN, RatioMAX	Shortest/highest ratio in the task.
CountMIN, CountMAX	Shortest/highest ratio in the task.
CORE	Core number.

DEFault	Default format.
LEN *<size>*	Specifies the width of non numeric fields (e.g. symbols)
TimeAuto	Adapt the time display. (default)
TImeFixed	Display all time information in seconds.

Options

This section describes the options of the **<trace>.STATistic** command group. Not all options are supported by all **<trace>.STATistic** commands.

JoinCORE (default)	Analysis is performed for all cores. The core information is discarded.
SplitCORE	Same as JoinCORE.
MergeCORE	Same as JoinCORE.
CORE *<number>*	Analysis is performed for the specified core.

TASK [!] *<task>*	Analysis is performed for the specified task only or excluding the task. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).
SplitTASK	Splits up the results for different tasks.
MergeTASK	Trace information is analyzed independently for each task. The trace statistics summarizes these results to a single result.

INLINE	Treat inline functions as separate functions (default).
NoINLINE	Discard inline functions from the results.
LABEL	Include all symbols in the results.
NoLABEL	Only include functions in the results.

FILE	Displays trace memory contents loaded with **Trace.FILE**.
FlowTrace	The trace works as flow trace. This option is usually not required.
BusTrace	The trace works as a bus trace. This option is for diagnosis only and usually not required.
ACCUMULATE	By default only the current trace contents is analyzed by the statistic functions. The option **/ACCUMULATE** allows to add the current trace contents to the already displayed results.

INCremental	Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).
FULL	The result is displayed after the TRACE32 software finished the analysis.
Track	Track the **Trace.STATistic** window with other trace list windows (tracking to record number or time possible).
NoMerge	(For diagnosis purpose only).
Address *…*	Perform statistic on specified addresses, assign statistic information for all other functions to (other).
Filter *<item>*	Filter the described item. The recorded trace information is first filtered and then analyzed.
InterVal	Divide the time period recorded by the trace (total) into time slices and analyze the time slices, see **Interval Analysis**.
Number	Define the number of classes.
LOG	Display the bars in the result display in a logarithmic format (default).
LINear	Display the bars in the result display in a linear format.
BEFORE	Display the time before the event. That means how long the previous state lasted until the listed state was reached.
AFTER	Display the time after the event. That means how long the state lasted after it was reached (default).
List *<list_items>*	Specify the result that should be displayed in the window. Refer to **List items**.
/Sort *<item>*	Sorting the analysis result. Refer to **Sorting**.
INTR	The time spent in interrupts is included to the measurement like a function call.
CLOCKS	The measurement results display the number of clocks instead of time information.

CountFirst (default)	Count the occurrence of the start address of a program symbol region or of a function.
CountChange	Count how often the address range of a program symbol region or of a function was entered.
CountALL	Count all executed instructions.

IncludeOWN, IncludeTASK, IncludeINTR	Refer to the **Analysis Options**.

ARTIAP	Option for AUTOSAR Real-Time Interface on Adaptive Platform trace decoding. Decode MIPI STP (System Trace Protocol) format trace which is defined in ARTI Trace Driver on AUTOSAR Adaptive Platform.

InterVal Analysis

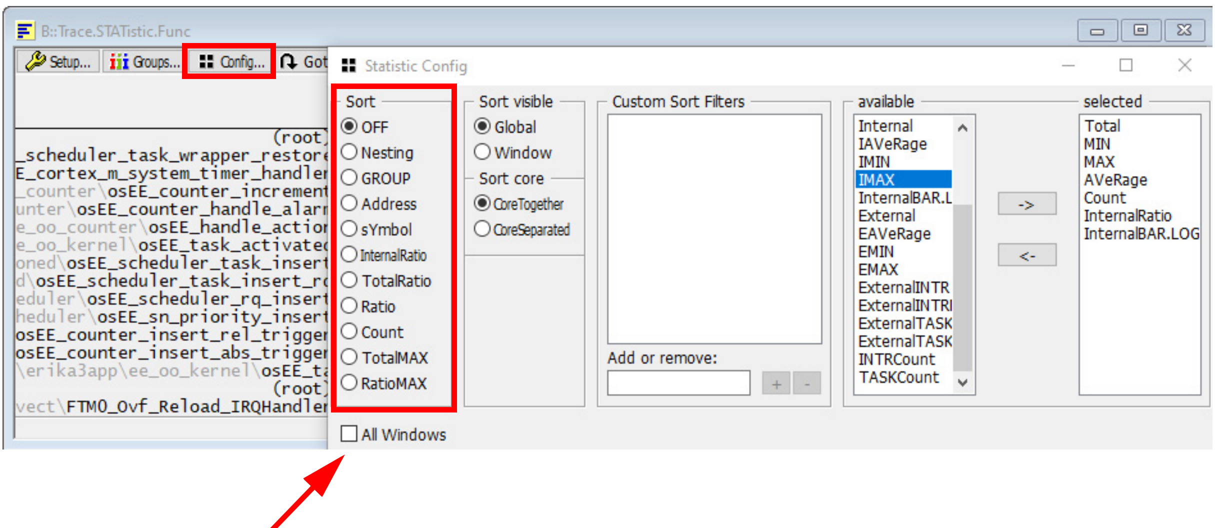
The **InterVal** option allows to divide the time period recorded by the trace (total) into time slices. Additional analysis details can be displayed for these time slices.

Trace.STATistic.TASK /InterVal *<time>* | *<event>*

```
; divide trace into 10.ms time slices
Trace.STATistic.TASK /InterVal 10.ms

; divide trace in time slices, a new time slice is started when the
; function Funccpu0_generateData is entered
Trace.STATistic.TASK /InterVal sYmbol Funccpu0_generateData
```

/Sort *<item>*	Sorting the Analysis Result
OFF	Sorting by program flow (default)
Nesting	Sorting by nesting
Address	Sorting by addresses
sYmbol	Sorting by names
TotalRatio/Ratio	Sorting by TotalRatio
Count	Sorting by Count
Window **Global**	(ineffectual)

The sorting can also be arranged by pushing the **Config** button in the **Trace.STATistic.Func** window.



If **All Windows** is selected, the selected sorting method is applied to all **Trace.STATistic** and **Trace.Chart** windows.

See also **Trace.STATistic.Sort**.

See also

- <trace>.Chart
- <trace>.PROfileSTATistic
- <trace>.STATistic.AddressDIStance
- <trace>.STATistic.AddressGROUP
- <trace>.STATistic.COLOR
- <trace>.STATistic.DatasYmbol
- <trace>.STATistic.DistriB
- <trace>.STATistic.FIRST
- <trace>.PROfileChart
- <trace>.STATistic.Address
- <trace>.STATistic.AddressDURation
- <trace>.STATistic.ChildTREE
- <trace>.STATistic.CYcle
- <trace>.STATistic.DIStance
- <trace>.STATistic.DURation
- <trace>.STATistic.Func

- <trace>.STATistic.FuncDURation
- <trace>.STATistic.FuncDURationInternal
- <trace>.STATistic.GROUP
- <trace>.STATistic.Ignore
- <trace>.STATistic.INTERRUPT
- <trace>.STATistic.InterruptIsFunction
- <trace>.STATistic.InterruptIsKernel
- <trace>.STATistic.InterruptIsKernelFunction
- <trace>.STATistic.InterruptIsTaskswitch
- <trace>.STATistic.INTERRUPTTREE
- <trace>.STATistic.LAST
- <trace>.STATistic.Line
- <trace>.STATistic.LINKage
- <trace>.STATistic.Measure
- <trace>.STATistic.MODULE
- <trace>.STATistic.PAddress
- <trace>.STATistic.ParentTREE
- <trace>.STATistic.PROCESS
- <trace>.STATistic.PROGRAM
- <trace>.STATistic.PsYmbol
- <trace>.STATistic.RUNNABLE
- <trace>.STATistic.RUNNABLEDURation
- <trace>.STATistic.Sort
- <trace>.STATistic.sYmbol
- <trace>.STATistic.TASK
- <trace>.STATistic.TASKFunc
- <trace>.STATistic.TASKINFO
- <trace>.STATistic.TASKINTR
- <trace>.STATistic.TASKKernel
- <trace>.STATistic.TASKLOCK
- <trace>.STATistic.TASKORINTERRUPT
- <trace>.STATistic.TASKORINTRState
- <trace>.STATistic.TASKSRV
- <trace>.STATistic.TASKState
- <trace>.STATistic.TASKStateDURation
- <trace>.STATistic.TASKTREE
- <trace>.STATistic.TASKVSINTERRUPT
- <trace>.STATistic.TASKVSINTR
- <trace>.STATistic.TREE
- <trace>.STATistic.Use
- <trace>.STATistic.Var
- BMC.STATistic
- EVENTS.STATistic
- IProbe.state
- RunTime
- RunTime.state

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.Address** *<address1>* [*<address2>* …] [**/***<option>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **List** *<item>* **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

Displays the time interval between up to 8 program events.

<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

Analysis background:

address1
address2
address2
address3
address1
address1
address2

Examples:

```
Trace.STATistic.Address sieve func1 func2

Trace.STATistic.Address 0x125c 0x1264 0x1274 0x1290 0x12ac 0x12b8 0x12d8
```

See also

■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.AddressDIStance** *<address>* [*<timemin>*] [*<increment>*] [**/***<option>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **ACCUMULATE** **INCremental** | **FULL** **Number** *<record>* **LOG** | **LINear**

Displays the time interval for a single program event. Without parameter the assignment of classes (16) is done automatically. With arguments the classes can be set up manually.

<address>	Program event.
<timemin>	Allows to specify the time for the first result class.
<increment>	Allows to specify the increment for the next result class.
<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

The following 2 commands are equivalents:

```
Trace.STATistic.AddressDIStance InterruptEntry

Trace.STATistic.DIStance /Filter Address InterruptEntry
```

The parameter *<timemin>* allows to specify the time for the first result class, the parameter *<increment>* allows to specify the increment for the next result class.

```
Trace.STATistic.AddressDIStance InterruptEntry 15.0us 1.0us
```

See also

■ <trace>.STATistic   ■ <trace>.STATistic.DIStance

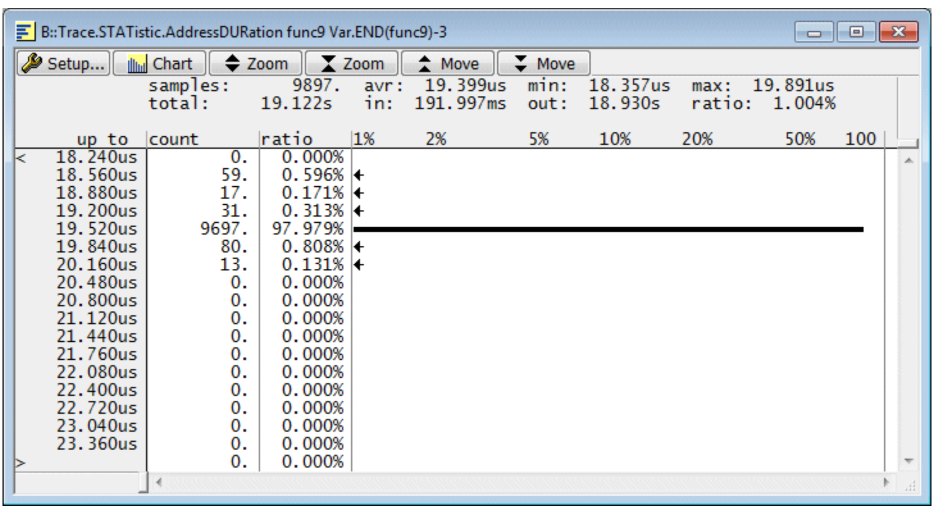
▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.AddressDURation** *<address1> <address2>* [*<timemin>*] [*<increment>*] [**/***<option>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **ACCUMULATE** **INCremental** | **FULL** **Number** *<record>* **LOG** | **LINear**

The statistic distribution between two program events is analyzed. This command can be used to analyze the run-time of a single function or interrupt response times.

<address1> *<address2>*	Program events.
<timemin>	Allows to specify the time for the first result class.
<increment>	Allows to specify the increment for the next result class.
<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

```
; Analyze the run-time of a single function
; func9: start address of the function
; sYmbol.EXIT(func9): Exit address of the given function
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9)
```



By default TRACE32 PowerView builds 16 result classes. For a graphical display of the results, use the command **Trace.PROfileChart.DURation**.

The *<option>* **Number** allows a user-defined number of result classes.

```
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9) /Number 6.
```

The parameter *<timemin>* allows to specify the time for the first result class, the parameter *<increment>* allows to specify the increment for the next result class.

```
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9) 15.us 1.us
```

Trace filter allow a more effective usage of the trace memory:

```
Trace.Mode Leash

Break.Set func9 /Program /TraceEnable

Break.Set sYmbol.EXIT(func9) /Program /TraceEnable

Go

WAIT !STATE.RUN()

Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9)
```

See also

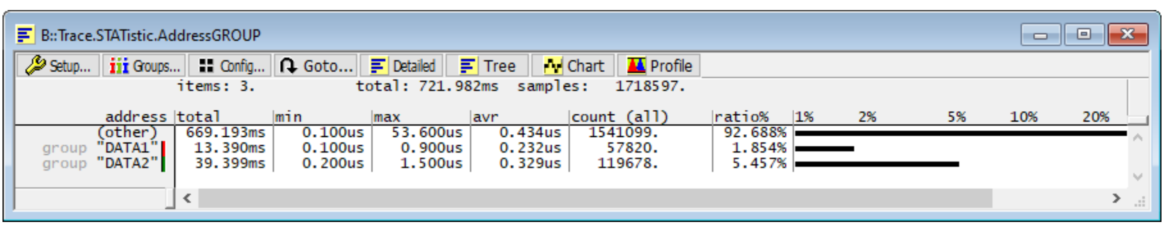
■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.AddressGROUP** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **NAME** | **GROUP** | **CORE** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

The time for accessed address **groups** and the number of accesses is calculated (flat statistic). The results include groups for both program and data addresses.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.



Example:

```
GROUP.Create "DATA1" 0x6800--0x68FF /RED

GROUP.Create "DATA2" 0x6700--0x67FF /GREEN

Trace.STATistic.AddressGROUP
```

See also

■ <trace>.STATistic ■ <trace>.STATistic.GROUP

▲ 'Release Information' in 'Legacy Release History'

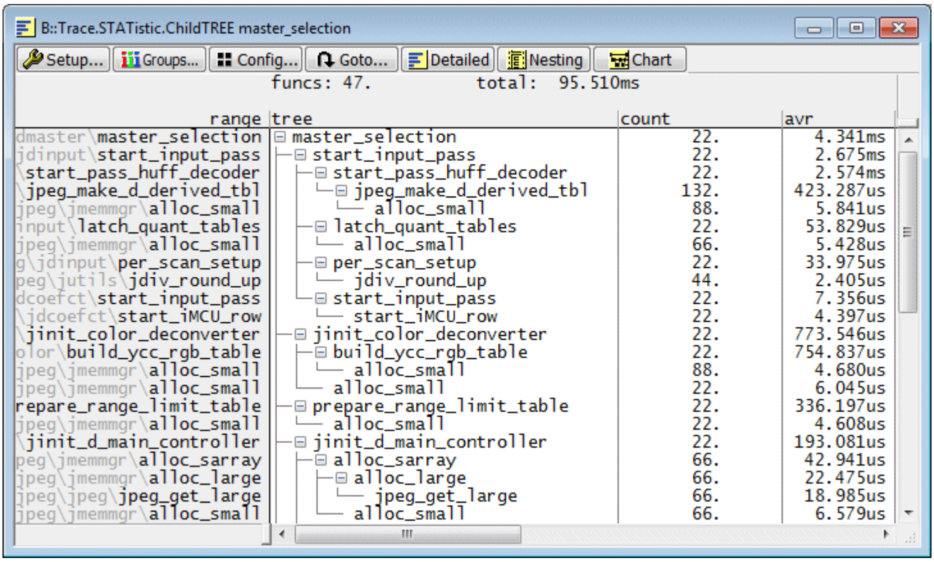
Format:	*<trace>*.**STATistic.ChildTREE** *<address>* [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **TREE** | **LEVEL** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **INTRROOT** | **INTRTASK** **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Show call tree and run-time of all functions called by the specified function. The function is specified by its start *<address>*.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Example:

```
Trace.STATistic.ChildTREE master_selection
```



See also

- <trace>.STATistic
- <trace>.STATistic.ParentTREE
- BMC.STATistic.ChildTREE

▲ 'Release Information' in 'Legacy Release History'

<trace>.STATistic.COLOR Assign colors to function for colored graphics

Format:	*<trace>*.**STATistic.COLOR FixedColors** | **AlternatingColors**

FixedColors (default)	Colors are assigned fixed to functions.
AlternatingColors	Colors are assigned by the recording order of the functions for each measurement.

See also

- <trace>.STATistic

▲ 'PowerView - Screen Display' in 'PowerView User's Guide'

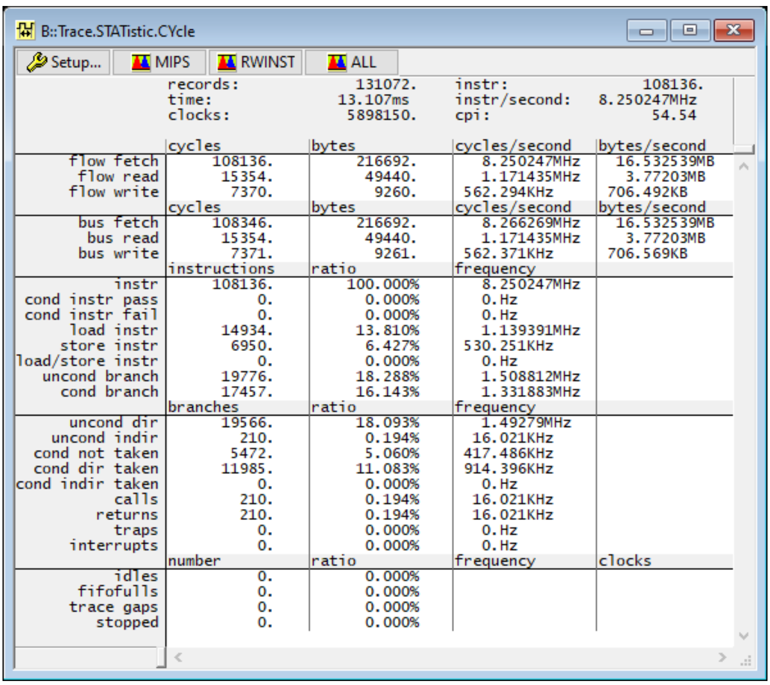
Format:	*<trace>*.**STATistic.CYcle** [*<time_range>*] [**/***<option>*]
<option>:	**FILE** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **INCremental** | **FULL** **IdleThreshold** *<clocks>* **TASK** *<task>* **ACCUMULATE**

Performs a statistical analysis of the cycle types.

<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

Example based on CoreSight ETMv3 for a Cortex-R4:

```
ETM.DataTrace ON                    ; full data trace
ETM.CycleAccurate ON                ; cycle accurate tracing
Trace.CLOCK 450.MHz                 ; inform TRACE32 about the core
                                    ; clock
```



survey	
records	Number of records in the trace
time	Time period recorded by the trace
clocks	Number of clock cycles recorded by the trace
instr	Number of instructions
instr/second	Instructions executed per second
cpi	Average clocks per instruction (clocks/instr)

details	
flow fetch	Number of cycles for instruction fetching
flow read	Number of cycles that performed a read access
flow write	Number of cycles the performed a write access
bus fetch	Number of fetch cycles
bus read	Number of data read cycles
bus write	Number of data write cycles
instr	Number of executed/not executed instruction
slot instr	Number of instructions executed in a branch delay slot
cond instr pass	Number of conditional instructions that passed (taken branch instructions not included)
cond instr fail	Number of conditional instructions that failed (failed branch instructions not included)
load instr	Number of load instructions
store instr	Number of store instructions
load/store instr	Number of instructions that do a load and a store
uncond branch	Unconditional branch instructions

details	
cond branch	Conditional branch instructions
uncond dir	Number of unconditional direct branches taken
uncond indir	Number of unconditional indirect branches taken
cond not taken	Number of failed conditional branch instructions
cond dir taken	Number of taken direct conditional branches
cond indir taken	Number of taken indirect conditional branches
traps	Number of traps
interrupts	Number of interrupts
idles	Number of “wait for interrupt” (coprocessor instruction or WFI instruction) or number of times that 1000. clock cycles passed without a broadcast of trace information. The option **IdleThreshold** allows to modify the number of clock cycles that need to pass for a idle detection.
fifofulls	Number of trace FIFO overflows (**FIFOFULL**)
trace gaps	Number of trace gaps (filtered trace information)
stopped	Number of debug stops
event ...	Number of trace events (architecture specific)

See also

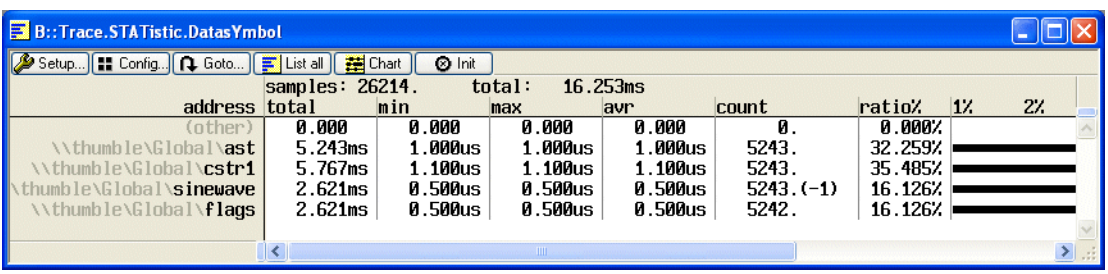
■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.DatasYmbol** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **NAME** | **GROUP** | **CORE** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **LABEL** | **NoLABEL** | **INLINE** | **NoINLINE** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* **Filter** *<filter>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

The command **Trace.STATistic.DatasYmbol** analyzes the contents of a pointer numerically.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.



If a full program and data trace is analyzed, the following command is recommended:

```
; analyze the contents of the pointer vpchar numerically
Trace.STATistic.DatasYmbol /Filter Address vpchar
```

A more effective usage of the trace memory is possible, if only write accesses to the pointer are recorded in the trace.

```
; set a filter to record only write cycles to the pointer vpchar to the
; trace
Var.Break.Set vpchar /Write /TraceEnable

;…

; analyze the contents of the pointer
Trace.STATistic.DatasYmbol

; analyze the contents of the pointer, sort the result by symbol names
Trace.STATistic.DatasYmbol /Sort sYmbol
```

See also

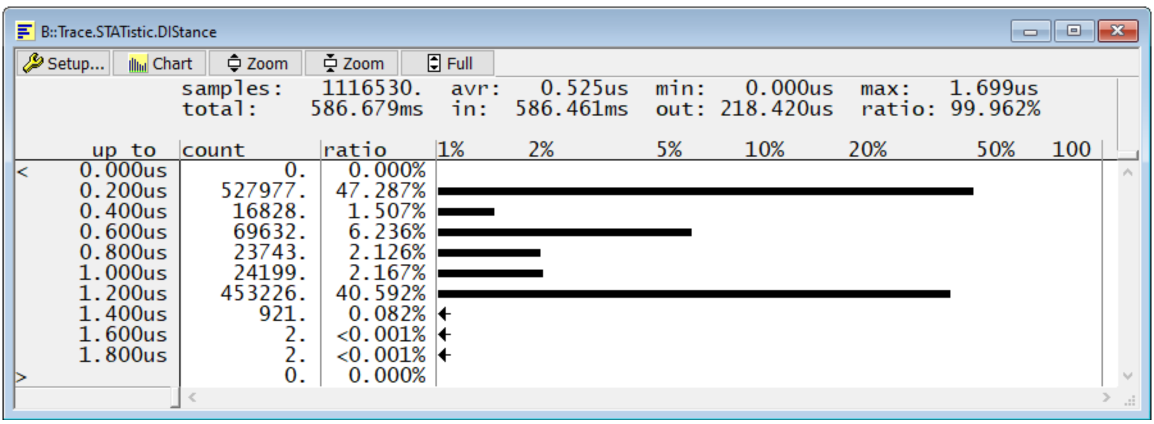
■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.DIStance** [*<timemin>*] [*<increment>*] [**/***<option>*]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **Filter** *<filter>* **ACCUMULATE** | **INCremental** | **FULL** | **LOG** | **LINear**

Displays the time interval for a single event. Without parameter the assignment of classes (16) is done automatically. With arguments the classes can be set up manually.

<timemin>	Allows to specify the time for the first result class.
<increment>	Allows to specify the increment for the next result class.
<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.




```
Trace.SAVE measure1
Trace.FILE measure1
Trace.STATistic.DIStance /FILE

; add the current trace contents to already displayed results
Trace.STATistic.DIStance /ACCUMULATE

; Define 10 classes
Trace.STATistic.DIStance /Number 10.
```

See also

- <trace>.STATistic
- <trace>.STATistic.AddressDIStance

▲ 'Jitter Measurement' in 'Application Note Profiling on AUTOSAR CP with ARTI'
▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.DistriB** [**%***<format>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | *<cpu>* | *<signals>* | **Port**[**.***<subitem>*] | **MARK**[**.***<marker>*] | **ENERGY.Abs** | **POWER**[**.OFF**] | **SAMPLE**[**.OFF**] | **SPARE**[**.OFF**] | **LOW** | **HIGH** | **FINDINDEX**
<options>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **LABEL** | **NoLABEL** | **INLINE** | **NoINLINE** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **List** [*<list_item>* …] **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

The statistic distribution of any data is displayed if *<item>* is specified. Displayed are the number of occurrences and the time after the events, i.e. the time an event is assumed to be valid. Without *<item>* the statistic is based on symbolic addresses.

<format>	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

- ■ <trace>.STATistic ■ BMC.STATistic.DistriB
- ▲ 'Release Information' in 'Legacy Release History'

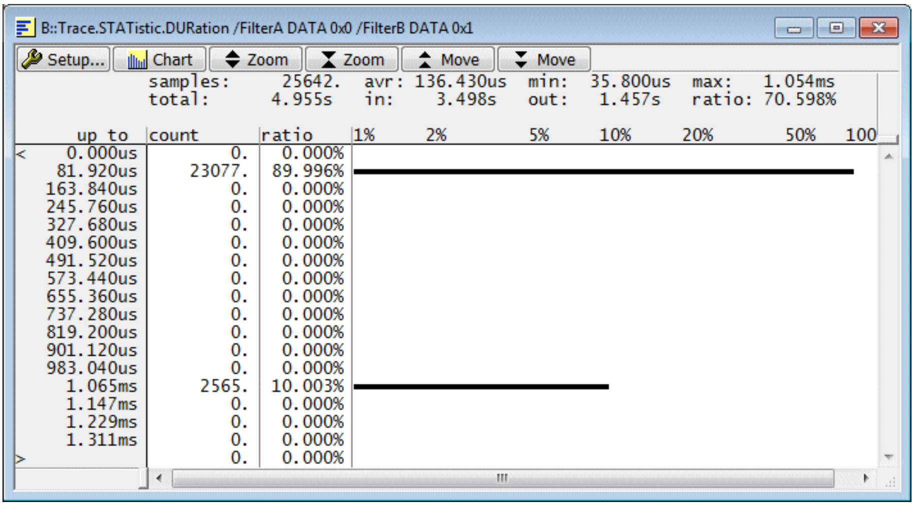
Format:	*<trace>*.**STATistic.DURation** [*<timemin>*] [*<increment>*] [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **ATOA** | **ATOB** | **ATOC** | **ATOD** **BTOA** | **BTOB** | **BTOC** | **BTOD** **CTOA** | **CTOB** | **CTOC** | **CTOD** **DTOA** | **DTOB** | **DTOC** | **DTOD** **FilterA** *<filter>* | **FilterB** *<filter>* **ACCUMULATE** **INCremental** | **FULL** **Number** | **LOG** | **LINear**

Analyzes the statistic distribution between two events. To determine the time interval between two instructions (addresses) **Trace.STATistic.AddressDURation** is more suitable.

<timemin>	Allows to specify the time for the first result class.
<increment>	Allows to specify the increment for the next result class.
ATOA	Display the time interval from A to A.
BTOA	Display the time interval from B to A.
BTOB	Display the time interval from B to B.
…	
FilterA *<item>*	Specify the first event.
FilterB *<item>*	Specify the second event.
Other options	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

Example: This example analyzes how long it takes when the contents of a variable changes from 0x0 to 0x1.

```
Var.Break.Set flags /Write /TraceEnable

Trace.STATistic.DURation /FilterA Data 0x0 /FilterB Data 0x1
```



In order to use the command **Trace.STATistic.DURation**:

- Check if both events are exported by a trace packet. Information reconstructed by TRACE32 is not analyzed.
- Alternatively use a **TraceEnable** breakpoint export the event as a trace packet.

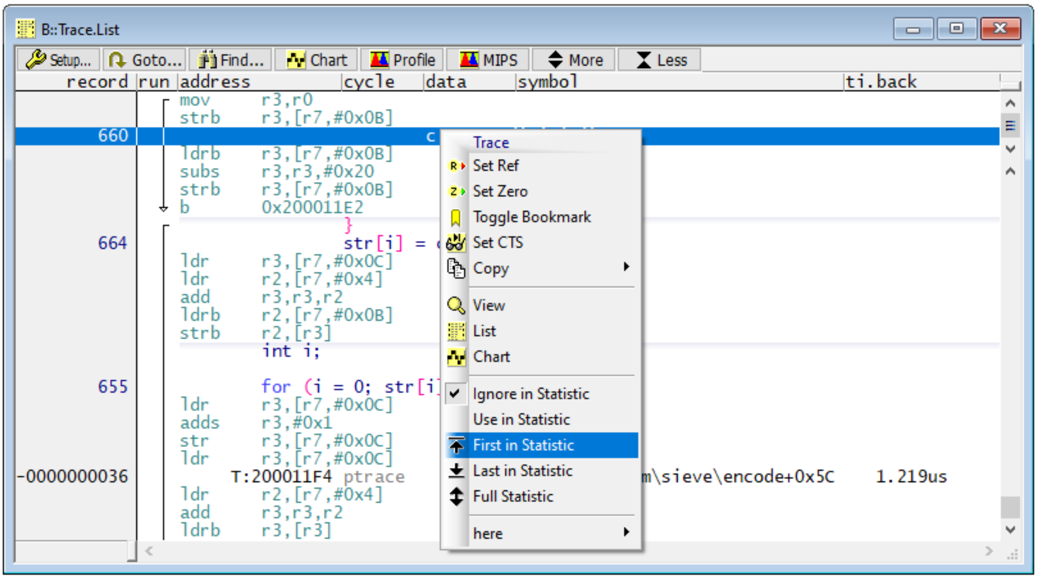
The options **FilterA** and **FilterB** provide you with the means to describe your event.

See also

■ <trace>.STATistic

Format: *<trace>*.**STATistic.FIRST** *<value>* | *<time>* | *<string>*

The **Trace.STATistic** commands analyze the complete trace contents by default. The command **Trace.STATistic.FIRST** allows to freely select a start point for the statistic analysis.



Example for <value>:

```
Trace.List                          ; display trace listing

Trace.STATistic.FIRST -123366.      ; select trace record -123366.
                                    ; as start point for the trace
                                    ; analysis

Trace.STATistic.LAST -36675.        ; select trace record -36675.
                                    ; as end point for the trace
                                    ; analysis

Trace.STATistic.Func                ; perform a function run-time
                                    ; analysis
```

Example for <time>:

```
Trace.List TIme.ZERO DEFault          ; display trace listing

Trace.STATistic.FIRST 0.3us           ; select trace record with time
                                      ; stamp 0.3 µs (zero time)
                                      ; as start point for the trace
                                      ; analysis

Trace.STATistic.Func                  ; perform a function run-time
                                      ; analysis between the specified
                                      ; start point and the end of the
                                      ; trace buffer
```

See also

- ■ <trace>.STATistic ■ <trace>.STATistic.LAST
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.Func** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **NAME** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **INTRROOT** | **INTRTASK** **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Analyzes the function nesting and calculates the time spent in functions and the number of function calls.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Considerations

	Please be aware that any gap in the trace recording (**FIFOFULL**) might result in a incorrect analysis results.

The trace can be tested for FIFOFULLs as follows:

```
; Process the complete trace contents
Trace.FLOWPROCESS

IF Analyzer.FLOW.FIFOFULL()!=0
     PRINT "Trace.STATistic.Func not possible due to FIFOFULL errors."
```

If it is not possible to eliminate the FIFOFULLs, it is recommended to use the command **Trace.STATistic.sYmbol**.

Analysis of the Function Nesting

In order to prepare the results for the command **Trace.STATistic.Func**, TRACE32 post-processes the program flow recorded by the PowerTrace to find:

- **Function entries**

  The execution of the first instruction of an HLL function is regarded as function entry.

  Additional identifications for function entries are implemented depending on the processor architecture and the used compiler.

- **Function exits**

  A RETURN instruction within an HLL function is regarded as function exit.

  Additional identifications for function exits are implemented depending on the processor architecture and the used compiler.

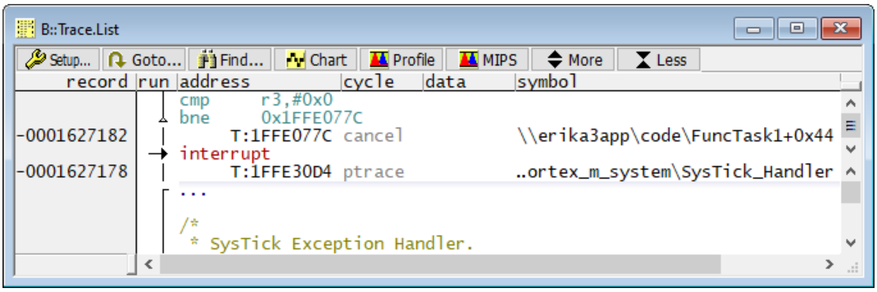
- **Entries to interrupt service routines (asynchronous)**

  If an interrupt was identified, the following entry to an HLL function is regarded as entry to the interrupt service routine.

  Interrupts are identified as follows:

  - The trace port broadcasts the occurance of an interrupt (e.g. PPC4xx).
  - An entry to the vector table is detected and the vector address indicates an asynchronous/hardware interrupt (e.g. ARM9).
  - If the vector table base address is configurable the usage of the command **SYStem.Option.VECTORS** might be necessary (e.g. MPC5xxx).

  If an interrupt is detected in the trace, it is marked as in the screenshot below.



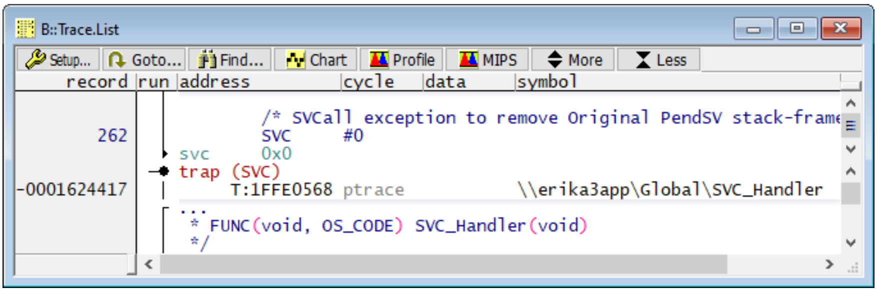
- **Exits of interrupt service routines**

  A RETURN / RETURN FROM INTERRUPT within the HLL interrupt service routine is regarded as exit of the interrupt service routine.

- **Entries to TRAP handlers (synchronous)**

  If an entry to the vector table was identified and if the vector address indicates a synchronous interrupt/trap the following entry to an HLL function is regarded as entry to the trap handler.

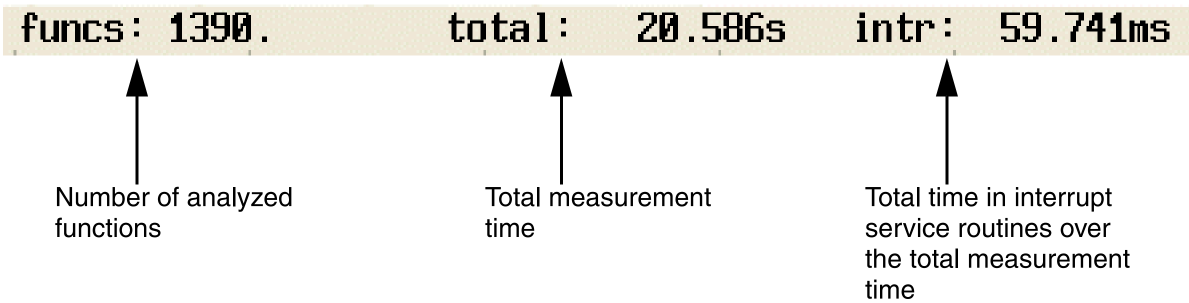
  If a TRAP is detected in the trace, it is marked as in the screenshot below.

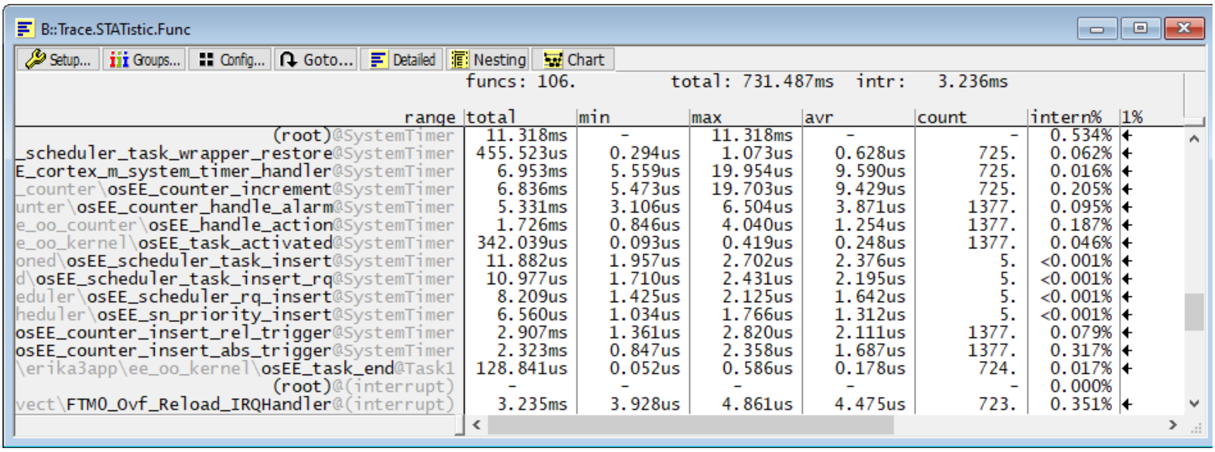


- **Exits of TRAP handlers**

  A RETURN / RETURN FROM INTERRUPT within the HLL trap handler is regarded as exit of the trap handler.

Interpretation of the Result





range	total	min	max	avr	count	intern%	1%
(root)@SystemTimer	11.318ms	-	11.318ms	-	-	0.534%	
_scheduler_task_wrapper_restore@SystemTimer	455.523us	0.294us	1.073us	0.628us	725.	0.062%	
E_cortex_m_system_timer_handler@SystemTimer	6.953ms	5.559us	19.954us	9.590us	725.	0.016%	
_counter\osEE_counter_increment@SystemTimer	6.836ms	5.473us	19.703us	9.429us	725.	0.205%	
unter\osEE_counter_handle_alarm@SystemTimer	5.331ms	3.106us	6.504us	3.871us	1377.	0.095%	
e_oo_counter\osEE_handle_action@SystemTimer	1.726ms	0.846us	4.040us	1.254us	1377.	0.187%	
e_oo_kernel\osEE_task_activated@SystemTimer	342.039us	0.093us	0.419us	0.248us	1377.	0.046%	
oned\osEE_scheduler_task_insert@SystemTimer	11.882us	1.957us	2.702us	2.376us	5.	<0.001%	
d\osEE_scheduler_task_insert_rq@SystemTimer	10.977us	1.710us	2.431us	2.195us	5.	<0.001%	
eduler\osEE_scheduler_rq_insert@SystemTimer	8.209us	1.425us	2.125us	1.642us	5.	<0.001%	
heduler\osEE_sn_priority_insert@SystemTimer	6.560us	1.034us	1.766us	1.312us	5.	<0.001%	
osEE_counter_insert_rel_trigger@SystemTimer	2.907ms	1.361us	2.820us	2.111us	1377.	0.079%	
osEE_counter_insert_abs_trigger@SystemTimer	2.323ms	0.847us	2.358us	1.687us	1377.	0.317%	
\erika3app\ee_oo_kernel\osEE_task_end@Task1	128.841us	0.052us	0.586us	0.178us	724.	0.017%	
(root)@(interrupt)	-	-	-	-	-	0.000%	
vect\FTM0_Ovf_Reload_IRQHandler@(interrupt)	3.235ms	3.928us	4.861us	4.475us	723.	0.351%	

Some additional explanations with regards to the function name (column **range**):

- **(root)**: is the root of the analyzed function nesting.
- **HLL interrupt service routines:** HLL interrupt service routines are indicated in the analysis as shown below:

- **HLL trap handler:** HLL trap handler are indicated in the analysis as shown below:

If **Trace.STATistic.TASKFunc** was performed instead of **Trace.STATistic.Func**, because TRACE32 detected an RTOS, the following function names will appear:

- **<function>@<task_name>:** The name of the task in which the function is called is appended to the function name.

- **(root)@<task_name>:** is the root of the analyzed function nesting for the task *<task_name>*.
- **(root)@(root):** program section where no task-assignment is possible (e.g. measurement started within a task) are summarized here.

The following description of the *<list_item>* that provide the analysis results is kept quite general. An accurate description is given together with the **Analysis Options**.

<list_item>	**Default Display**
Total	The total time within the function.
MIN	The shortest measured time it took to execute the function. The time includes the execution times of all sub-function calls. The time used for interrupt requests is not included, unless the window is opened with option **IncludeINTR**. If the function was never executed completely, the MIN time is not displayed.
MAX	The longest measured time it took to execute the function. The time includes the execution times of all subfunction calls. The time used for interrupt requests is not included, unless the window is opened with option **IncludeINTR**.
AVeRage	The average time it took to execute the function. The time includes the execution times of all subfunction calls. The time used for interrupt requests is not included, unless the window is opened with option **IncludeINTR**
Count	Number of calls of the function. If a function is never completely executed, no number of calls is displayed.

If function entries or exits are missing, this is display in the following format:

<times within the function>. (*<number of missing function entries>*/*<number of missing function exits>*).

Interpretation examples:

1. 950. (0/1): 950. times within the function, 1 function exit is missing.
2. 9. (1/0): 9. times within the function, 1 function entry is missing.
3. 11. (1/1): 11. times within the function, 1 function entry and 1 function exit is missing.
4. 9. (0/3): 9. times within the function, 3 function exits missing.

	If the number of missing function entries or exits is higher the 1. the analysis performed by the command **Trace.STATistic.Func** might fail due to nesting problems. A detailed view to the trace contents is recommended. In some cases a further treatment of the trace contents might help. For more information refer to **Adjusting the Measurement**.

<list_item>	Time only in Function
Internal	Total time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …
IAVeRage	Average time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …
IMIN	Shortest between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …
IMAX	Longest time spent in the function between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …
InternalRatio	*<internal_time_of_function>/<total_measurement_time>* as a numeric value.
InternalBAR	*<internal_time_of_function>*/*<total_measurement_time>* graphically.

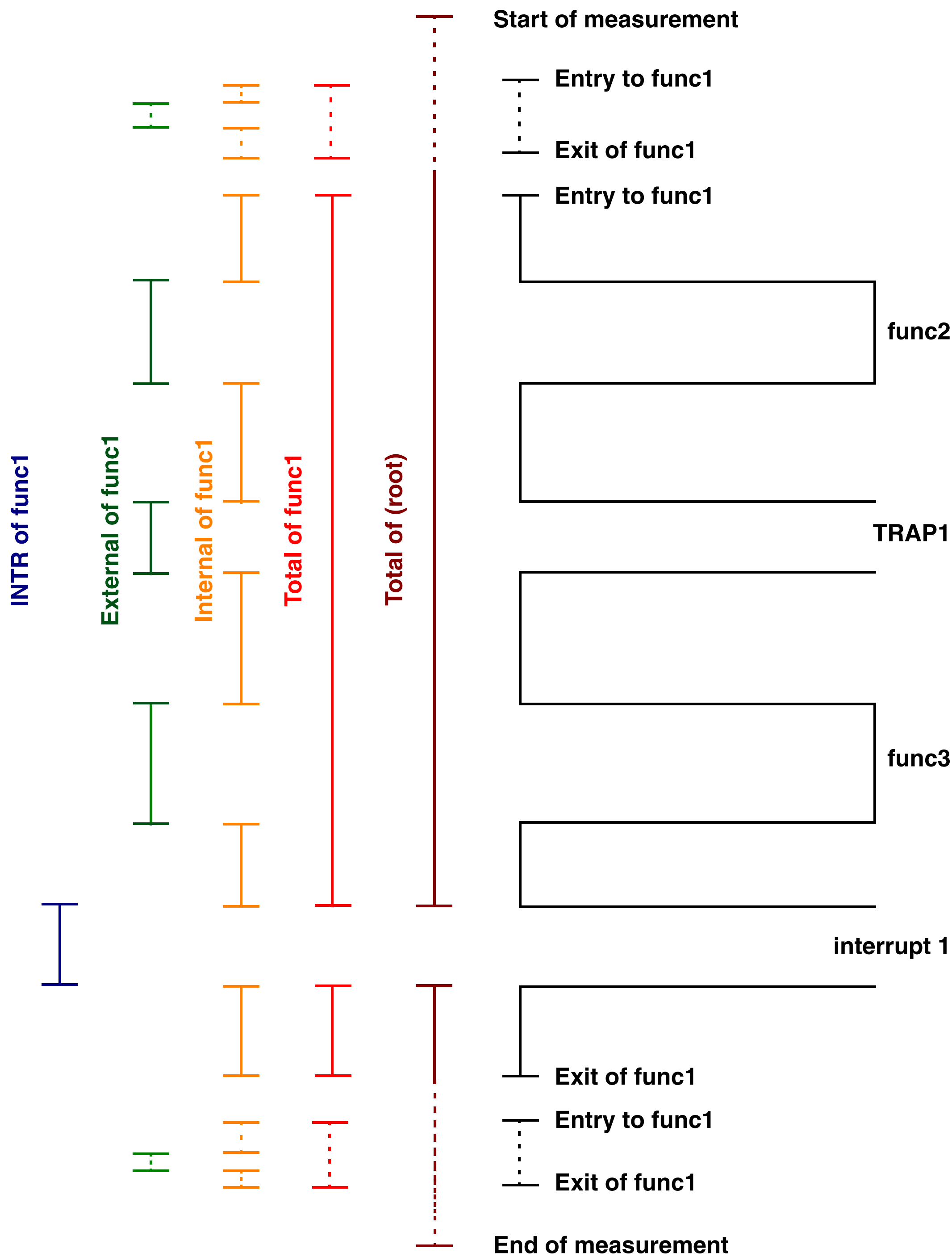
<list_item>	Time in Sub-Functions
External	Total time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …
EAVeRage	Average time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …
EMIN	Shortest time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …
EMAX	Longest time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …

<list_item>	Interrupt Times
INTR	Total time the function was interrupted.
INTRMAX	Max. time 1 function pass was interrupted.
INTRCount	Number of interrupts that occurred during the function run-time.

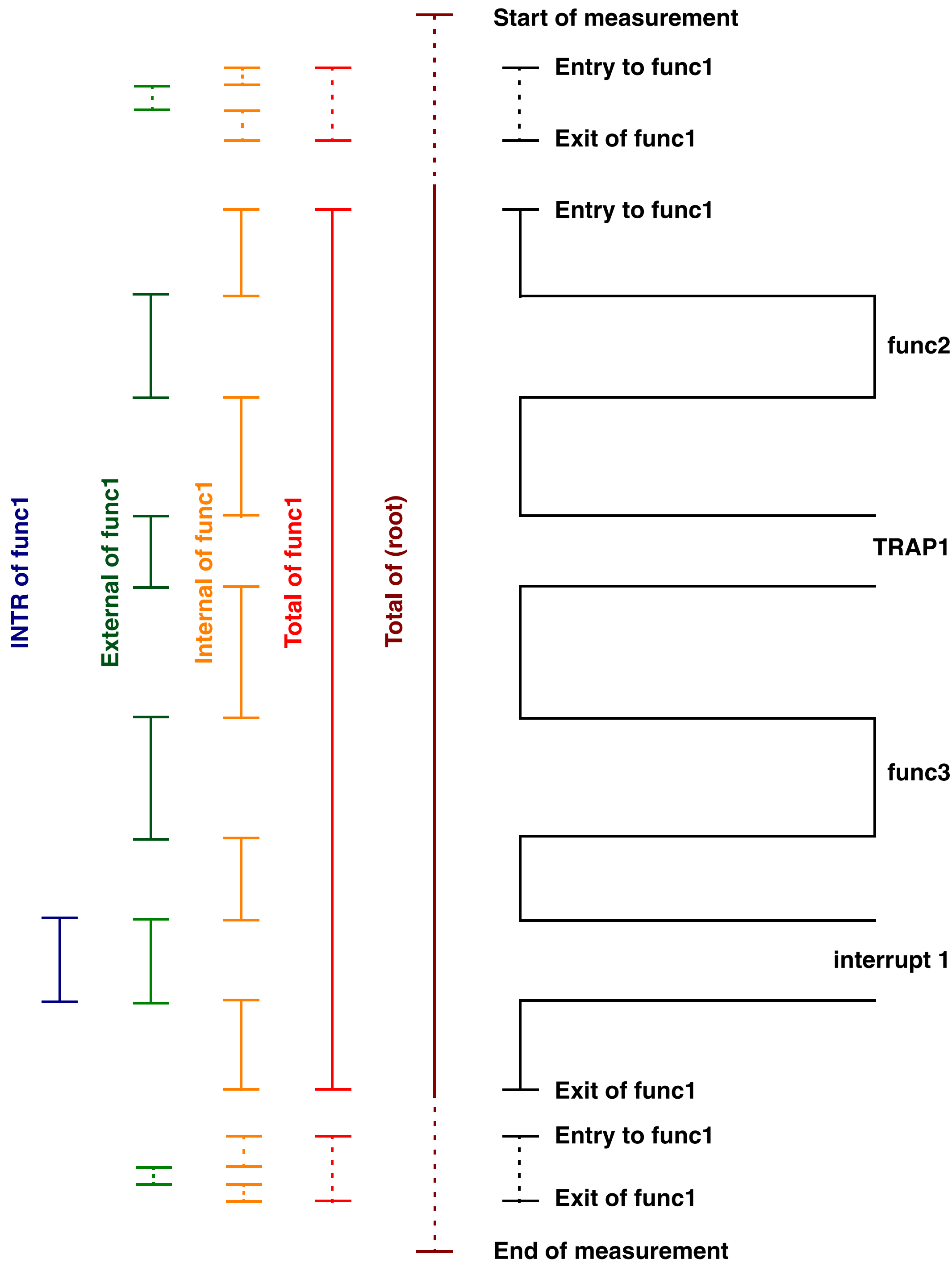
<list_item>	Time in Other Tasks (Trace.STATistic.TASKFunc only)
ExternalTASK	Total time in other tasks.
ExternalTASKMAX	Max. time 1 function pass was interrupted by other tasks.
TASKCount	Number of other tasks that interrupted the function.

<list_item>	Total Time Ratio
TOTALRatio	*<total_time_of_function>/<total_measurement_time>* as a numeric value.
InternalBar	*<total_time_of_function>/<total_measurement_time>* graphically.

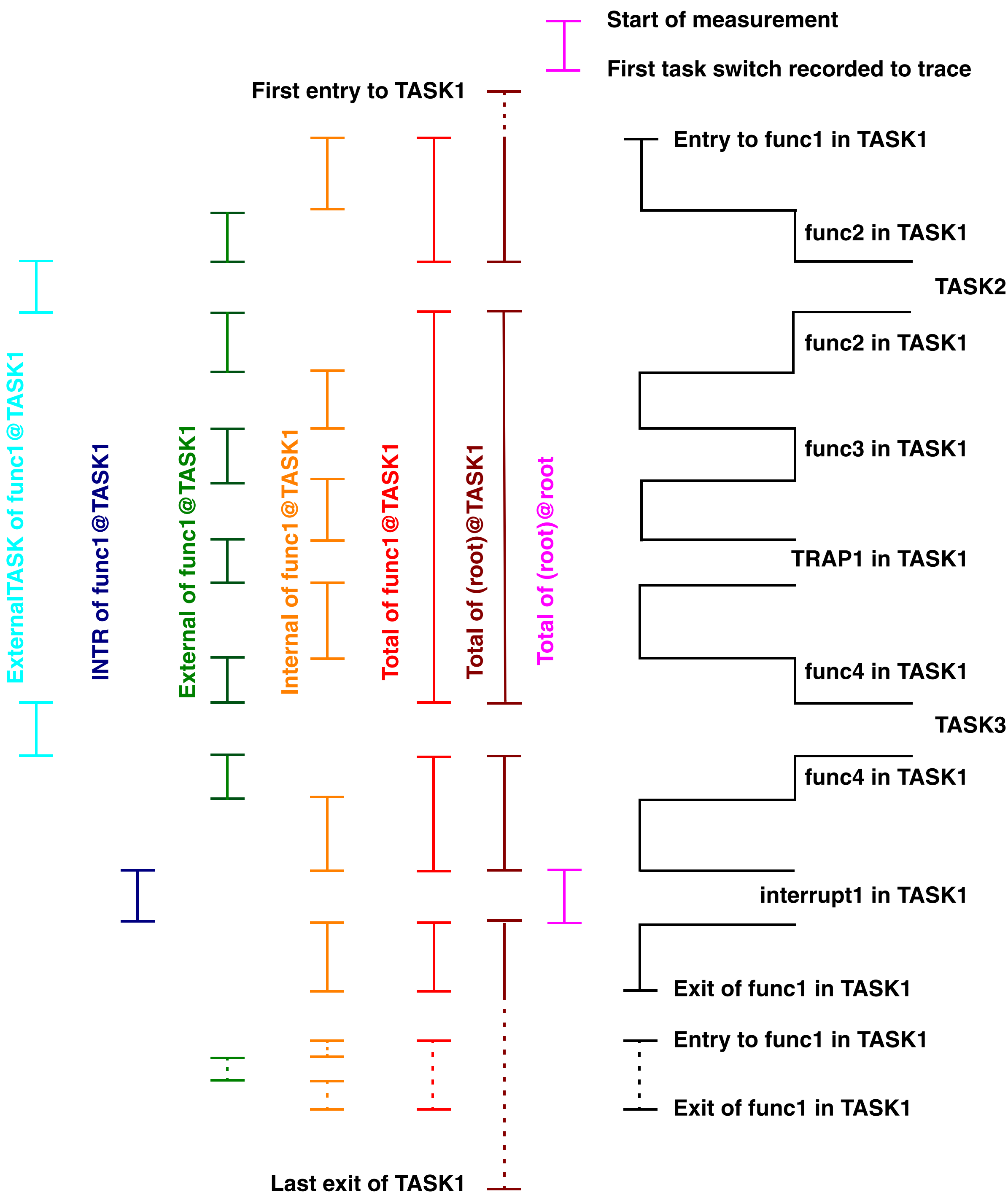
<option>	Configuration of the Analysis
(default)	Function run-times are calculated without interrupts.



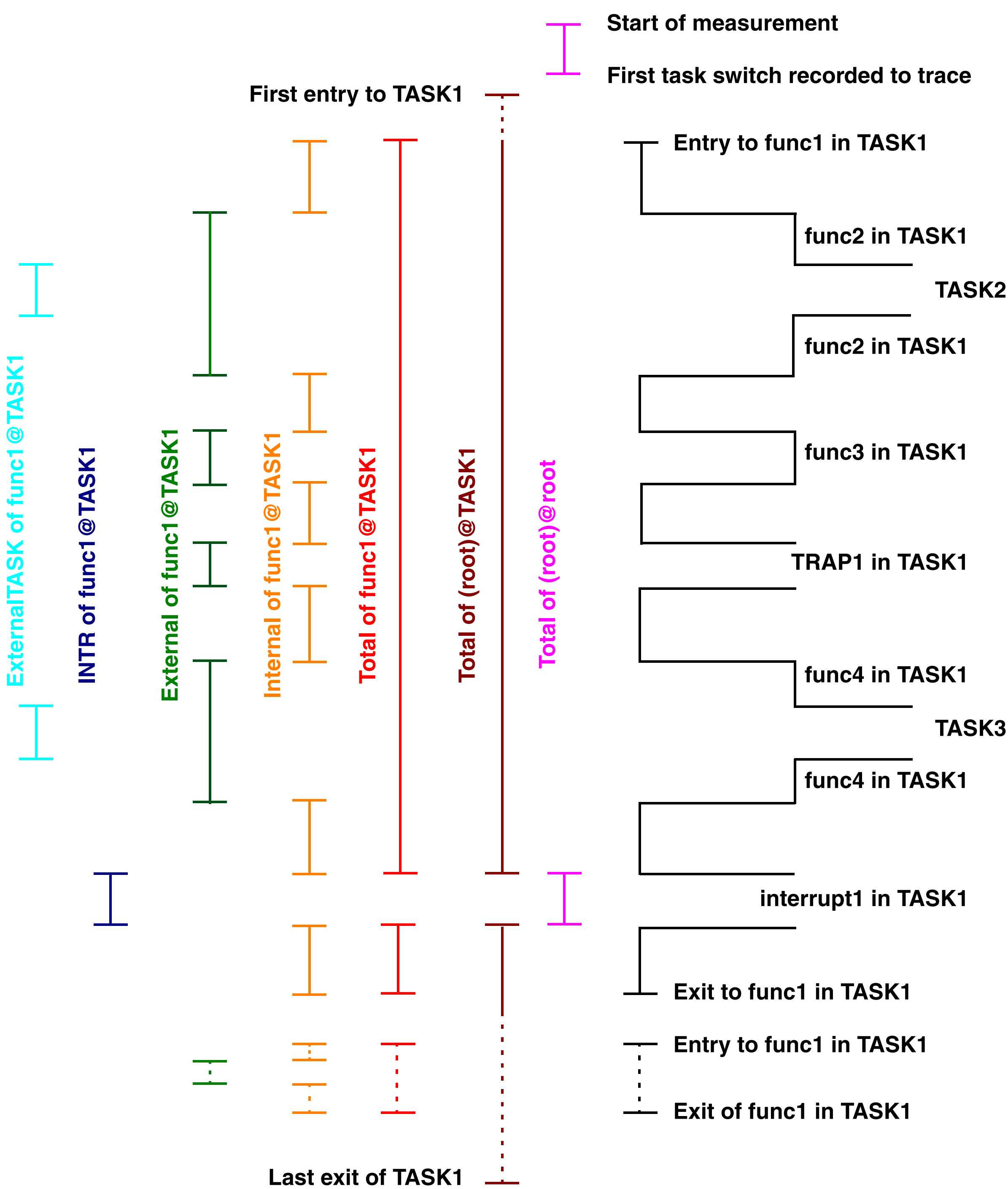
<option>	Configuration of the Analysis
IncludeINTR	Function run-times include times in interrupts. In other words, interrupts are treated as sub-functions.



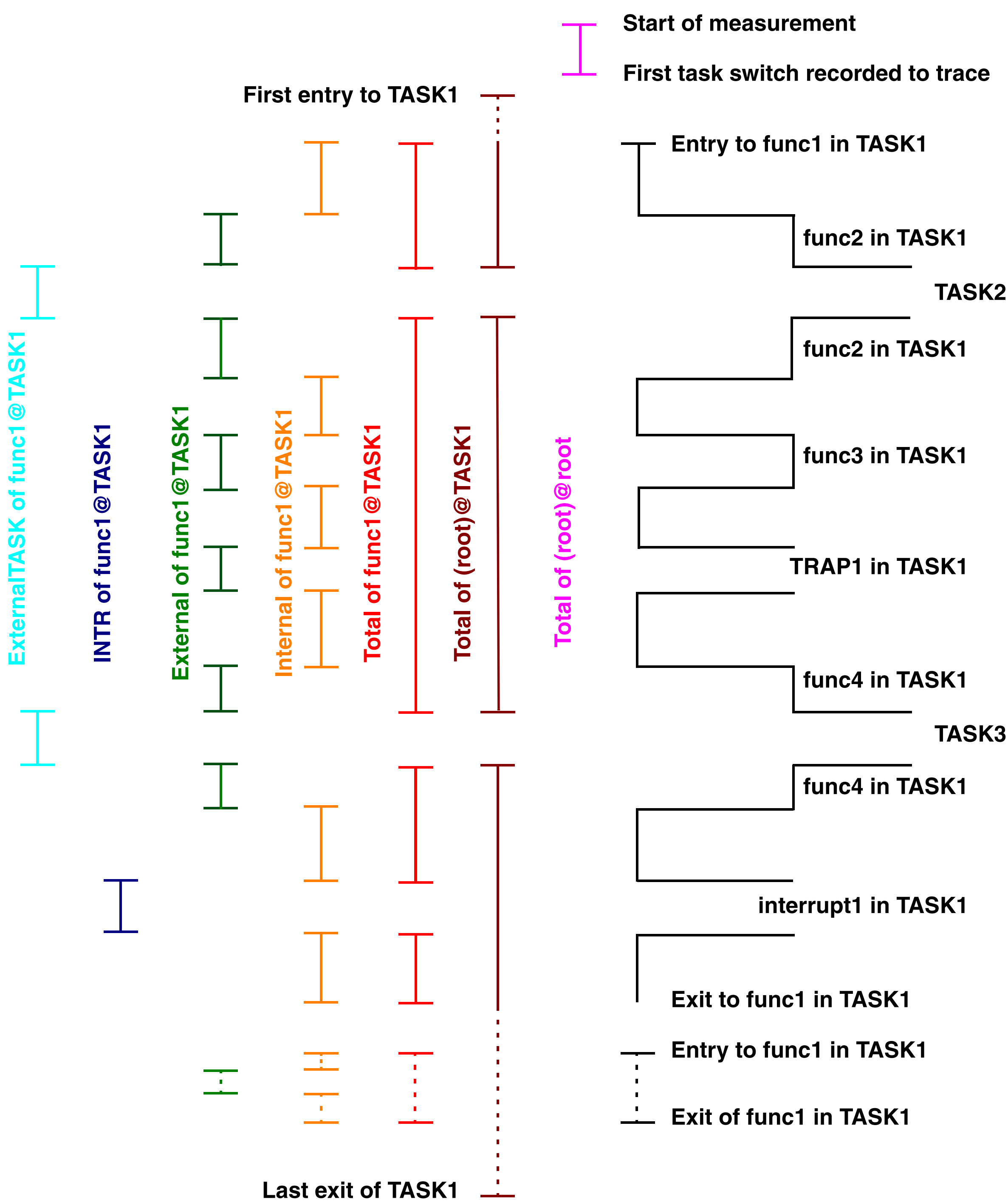
<option>	Configuration of the Analysis (RTOS)
IncludeOWN + **INTRROOT**	Function run-times without interrupts and without times in other tasks (default). Interrupts are assigned to (root)@(root)



<option>	Configuration of the Analysis (RTOS)
IncludeTASK + **INTRROOT**	Function run-times without interrupts but with times in other tasks. Interrupts are assigned to (root)@(root)



<option>	Configuration of the Analysis (RTOS)
IncludeOWN + **INTRTASK**	Function run-times without interrupts and without times in other tasks (default). Interrupts are assigned to (root)@*<task_name>*



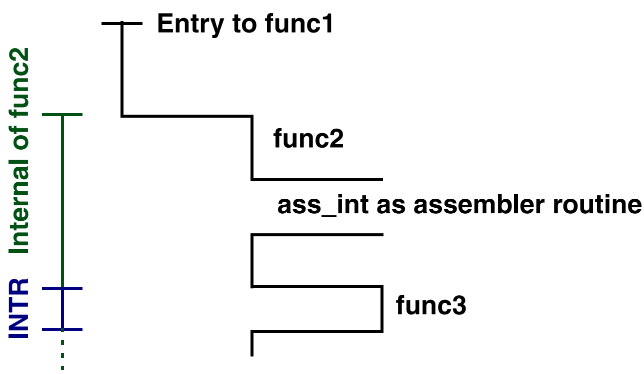
- **Trace.STATistic.FIRST/ Trace.STATistic.LAST**

  The **Trace.STATistic** commands analyze the complete trace contents by default. The command **Trace.STATistic.FIRST** allows to freely select the start point for the statistic analysis; the command **Trace.STATistic.LAST** allows to freely select the end point for the statistic analysis.

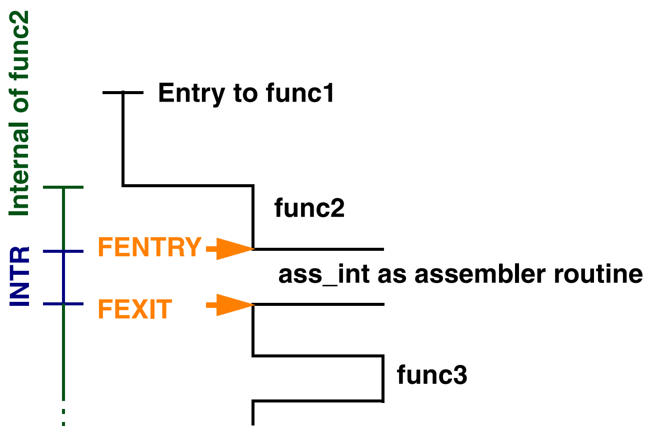
- **sYmbol.MARKER.Create FENTRY / FEXIT**

  If the function nesting analysis can't identify code sections as HLL functions (e.g. assembler function, unusual function exits) these code sections can be marked manually as functions by using the marker FENTRY and FEXIT.

  Example 1:



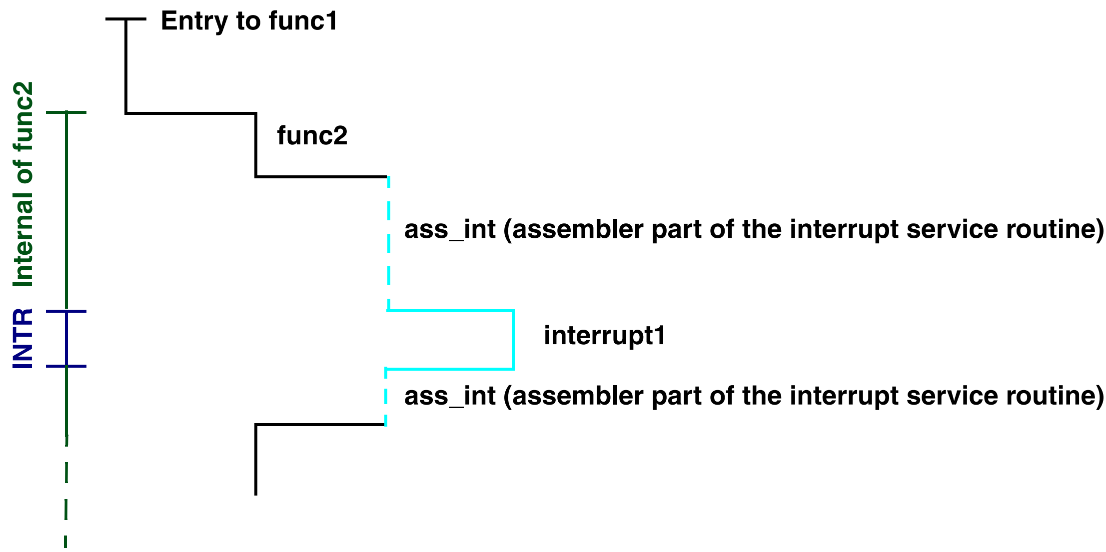
Since func3 is the HLL function executed after an interrupt occurred, it is regarded as interrupt service routine.



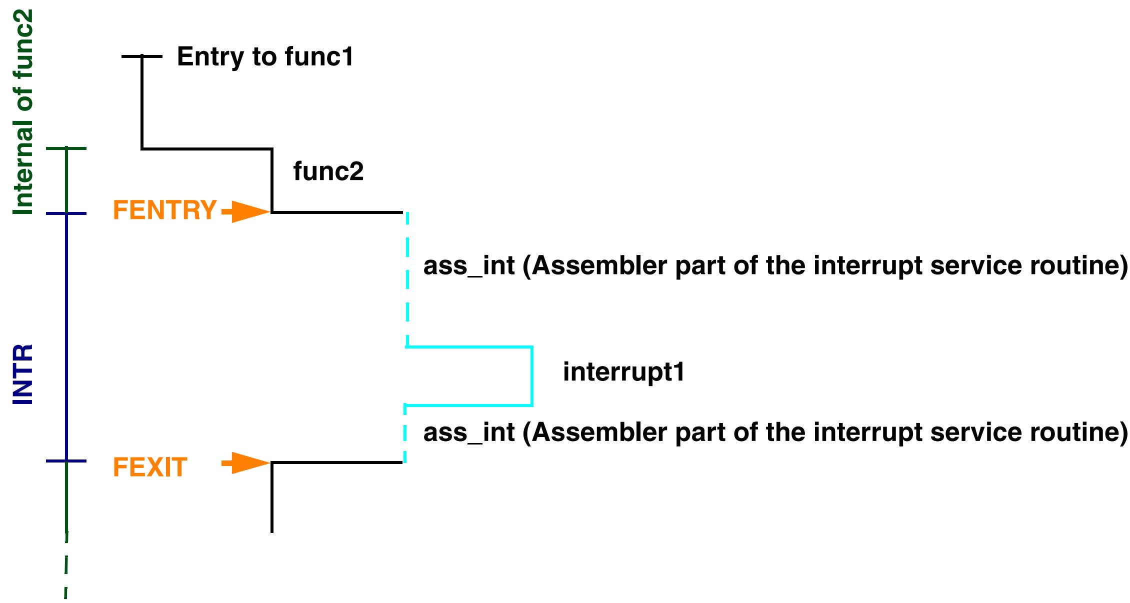
Since ass_int is now marked as a function, it is correctly identified as interrupt service routine.

```
; mark the entry of the assembler function ass_int as function entry
sYmbol.MARKER.Create FENTRY ass_int

; mark the exit of the assembler function ass_int as function exit
sYmbol.MARKER.Create FEXIT ass_int+0x15F

; list the marker
sYmbol.MARKER.list
```

Example 2:



Since interrupt1 is the HLL function executed after an interrupt occurred, it is regarded as interrupt service routine. The assembler code from ass_int is added to the time in func2.

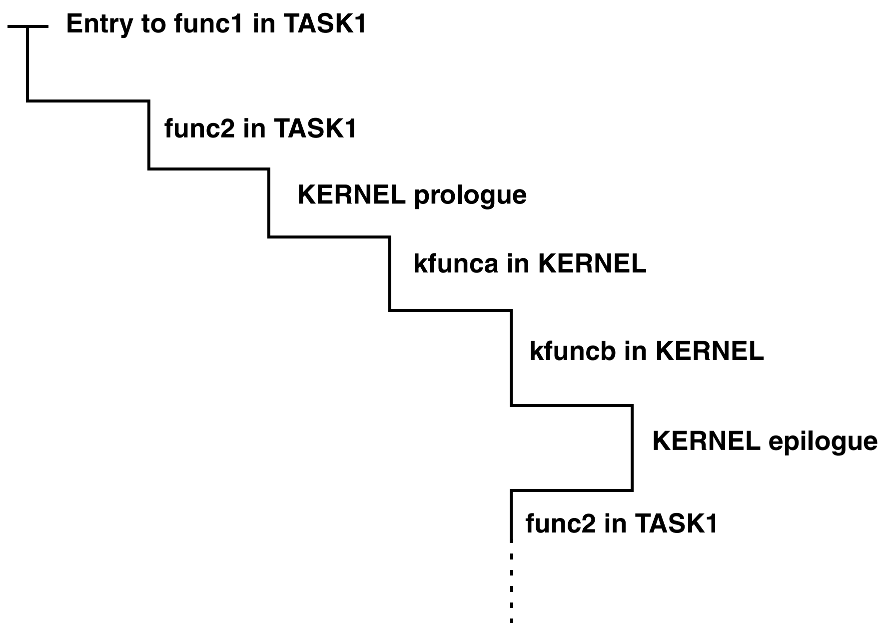


Since ass_int is now marked as function, it is correctly identified as interrupt service routine. interrupt1 is a sub-function called by ass_int now.

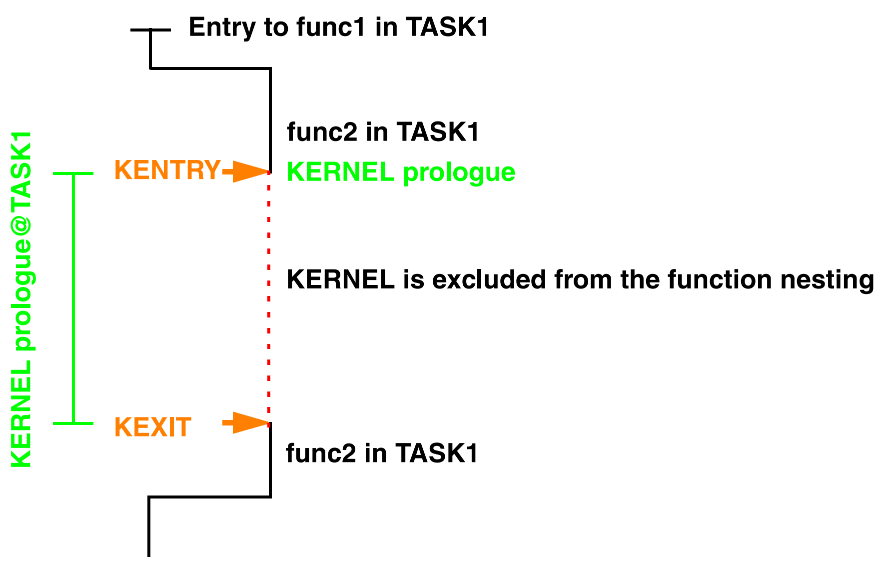
- **sYmbol.MARKER.Create KENTRY / KEXIT**

  If the KERNEL is using special methods to call/end KERNEL functions, this might annoy the function nesting analysis. In such a case it is recommended to exclude the KERNEL from the function nesting by using the markers KENTRY/KEXIT.

  Example:



  The KERNEL is manipulating the return address on the stack in order to return quickly into TASK1. This behavior will annoy the function nesting analysis.



  The usage of the markers KENTRY/KEXIT excluded the KERNEL from the function nesting in order to get a correct function nesting.

Advanced example for RTOS RTXC on a StarCore CPU:

```
; mark all interrupt service routines as kernel entries
sYmbol.ForEach "sYmbol.NEW.MARKER KENTRY *" "_isr_*"

; mark all RTE instructions in the specified program range as kernel exit
Data.Find P:RTXCProlog--P:RTXCProlog_end %Word 0x9f73
WHILE FOUND()
(
     sYmbol.MARKER.Create KEXIT P:TRACK.ADDRESS()
     Data.Find
)

sYmbol.MARKER.list
```

See also

■ <trace>.STATistic ■ BMC.STATistic.Func ■ CTS.STATistic.Func

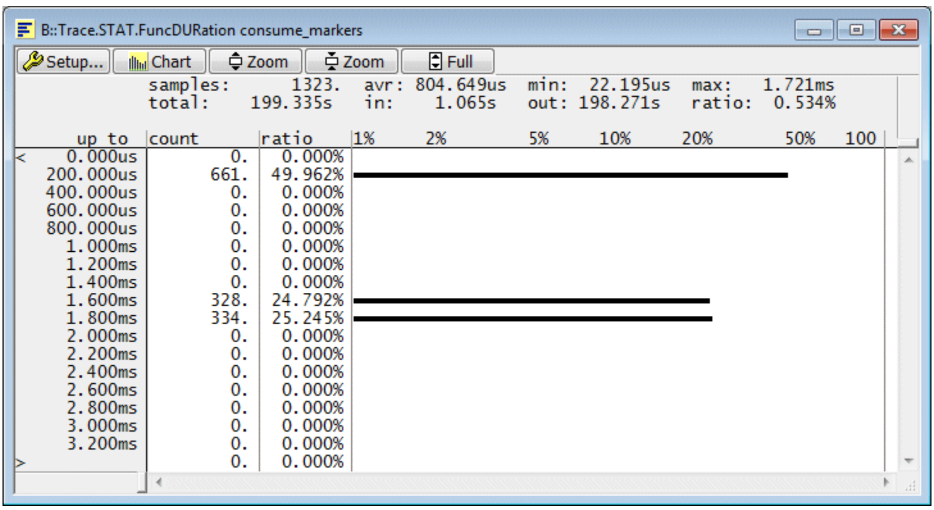
▲ 'Release Information' in 'Legacy Release History'
▲ 'Function Run-Times Analysis' in 'Training Arm CoreSight ETM Tracing'
▲ 'Function Run-Times Analysis - SMP Instance' in 'Training MPC5xxx/SPC5xx Nexus Tracing'

Format: *<trace>***.STATistic.FuncDURation** *<function_name* | *address>* [**/***<option>*]

<option>: **FILE** | **FlowTrace** | **BusTrace**
CORE *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only)
TASK *<task>* | **SplitTASK** | **MergeTASK**
MACHINE *<machine_magic>* | *<machine_id>* | *<machine_name>*
IncludeOwn | **IncludeTASK** | **IncludeINTR**
INTRROOT | **INTRTASK**
Number *<record>*
Filter *<filter>*
Address *<address* | *range>*
ACCUMULATE | **INCremental** | **FULL** | **CLOCKS** | **NoMerge**
LOG | **LINear**

Analyzes the function runtime between function entry and exit.

- The time spent in called subroutines is *included*.
- The time spent in called interrupt service routine and other tasks is *excluded*.



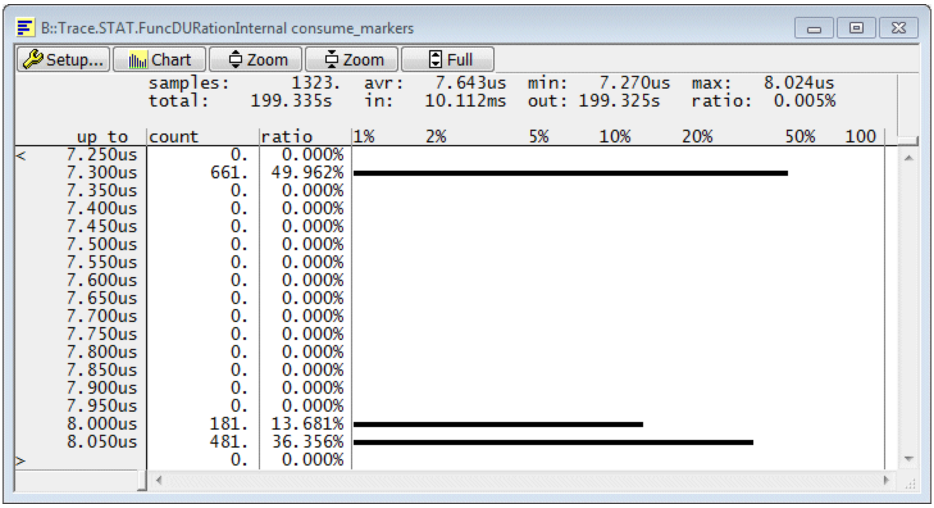
See also

■ <trace>.STATistic.FuncDURationInternal
■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'
▲ 'Function Run-Times Analysis - SMP Instance' in 'Training MPC5xxx/SPC5xx Nexus Tracing'

Format: *<trace>*.**STATistic.FuncDURationInternal** *<function_name* | *address>*

Analyzes the function runtime between function entry and exit. The time spent in called subroutines, traps, interrupt service routine and other tasks is excluded.



See also

- ■ <trace>.STATistic.FuncDURation
- ■ <trace>.STATistic
- ▲ 'Release Information' in 'Legacy Release History'

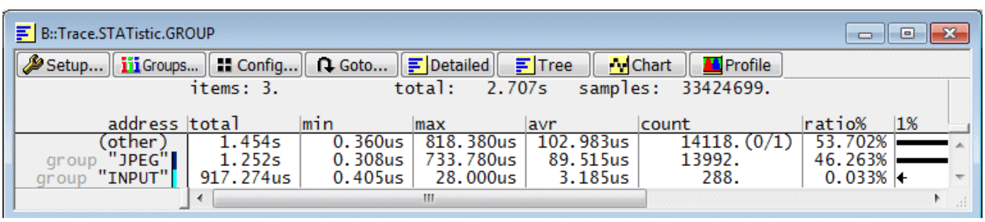
Format:	*<trace>*.**STATistic.GROUP** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* | **Filter** *<filter>* | **Sort** *<item>* | **Address** *<address* | *range>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** | **Track**

The time spent in **groups** and the number of calls is calculated (flat statistic). The results only include groups within the program range. Groups for data addresses are not included.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Example:

```
GROUP.Create "INPUT" \jquant2 \jquant1 \jidctred \jdinput /AQUA
GROUP.Create "JPEG" \jdapimin \jdcolor \jddctmgr \jdcoefct /NAVY
Trace.STATistic.GROUP
```



See also

- <trace>.STATistic
- BMC.STATistic.GROUP
- GROUP.Create
- <trace>.STATistic.AddressGROUP
- CTS.STATistic.GROUP

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.Ignore** [*<record>* | *<range>*] [**/***<options>*]
<option>:	**FILE**

The specified record(s) are ignored in the statistic analysis. This command can be used, when single records (caused by prefetch etc.) confuse the statistic analysis.

FILE	Displays trace memory contents loaded with **Trace.FILE**.

See also

- <trace>.STATistic

Format:	*<trace>*.**STATistic.INTERRUPT** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **NAME** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Analyzes the function nesting and calculates the time spent in interrupts and the number of interrupt calls.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

- ■ <trace>.STATistic ■ CTS.STATistic.INTERRUPT
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.InterruptIsFunction ON** | **OFF**

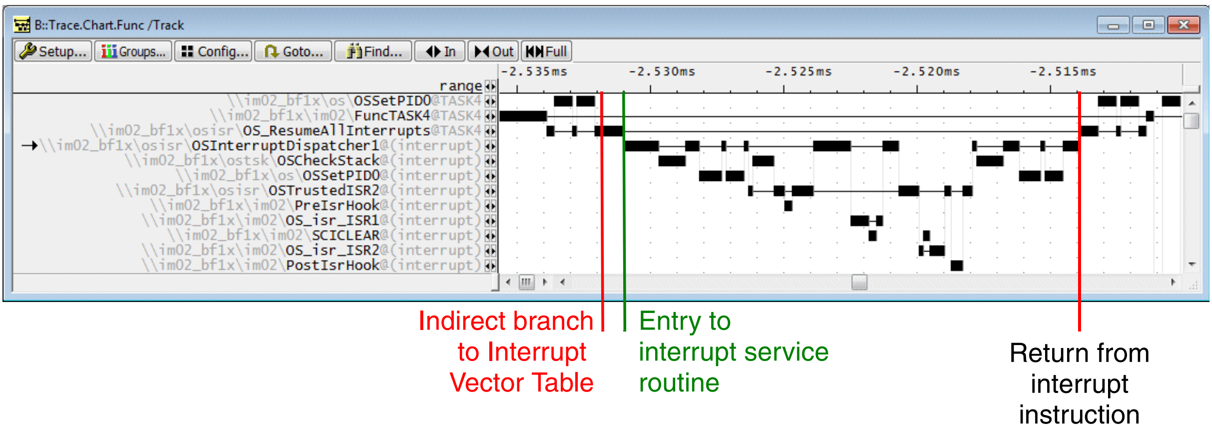
In order to calculate the results for the *nesting function run-time analysis* the trace recording is post-processed. One important issue in this processing is the identification of interrupt entries and exits.

TRACE32 provides two methods to identify interrupt entries and exits:

- Default: **Trace.STATistic.InterruptIsFunction OFF**
- Recommended: **Trace.STATistic.InterruptIsFunction ON**

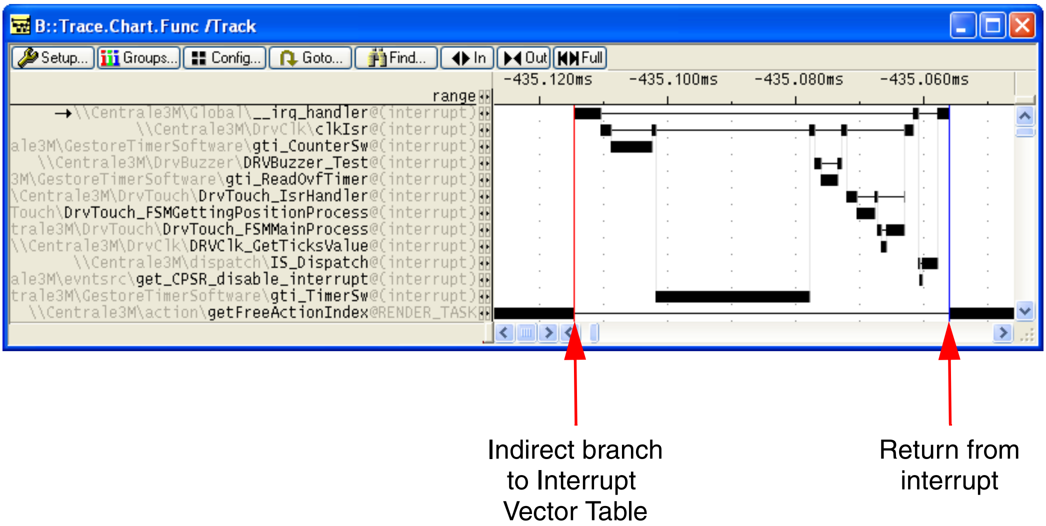
Trace.STATistic.InterruptIsFunction OFF

The screenshot below shows the function nesting for the interrupt.



1. The first HLL function called after the indirect branch to the Interrupt Vector Table is regarded as interrupt service routine (here OSInterruptDispatcher1).
2. The return from interrupt is regarded as the exit of this function (here OSInterruptDispatcher1).

Please be also aware that some trace port protocols require special setups for the Interrupt Vector Table. For details, please refer to your **Processor Architecture Manual**.



1. Interrupt entry is the point in the trace recording at which the indirect branch to the Interrupt Vector Table occurs.
2. Interrupt exit is the point in the trace recording at which the return from interrupt is executed.

TRACE32 handles the time between interrupt entry and exit as a function. The name given to this function is the label of the interrupt vector address.

Please be aware that method only works if interrupts are exit by regular return from interrupt.

Please be also aware that some trace port protocols require special setups for the Interrupt Vector Table. For details, please refer to your **Processor Architecture Manual**.

See also

- <trace>.STATistic

<trace>.STATistic.InterruptIsKernel   Statistics interrupt processing

Format: *<trace>*.**STATistic.InterruptIsKernel ON | OFF**

Same as **<trace>.STATistic.InterruptIsFunction**, however no function nesting analysis is performed inside interrupts.

See also

■ <trace>.STATistic

<trace>.STATistic.InterruptIsKernelFunction   Statistics interrupt processing

Format: *<trace>*.**STATistic.InterruptIsKernelFunction ON | OFF**

Same as **<trace>.STATistic.InterruptIsFunction**. The interrupt address ranges are additionally considered as KERNEL in TASKKernel analysis, e.g. using **<trace>.STATistic.TASKKernel**.

See also

■ <trace>.STATistic

<trace>.STATistic.InterruptIsTaskswitch   Statistics interrupt processing

Format: *<trace>*.**STATistic.InterruptIsTaskswitch ON | OFF**

Default: OFF.

When set to ON, this command delays a task switch that occurs within an interrupt after the return from interrupt instruction. The interrupt will be then assigned to the task that has been execution before the task switch. This can also be achieved using the command **sYmbol.MARKER.Create TASKSWITCH**.

The command only affects trace windows that analyze the program flow or task switches.

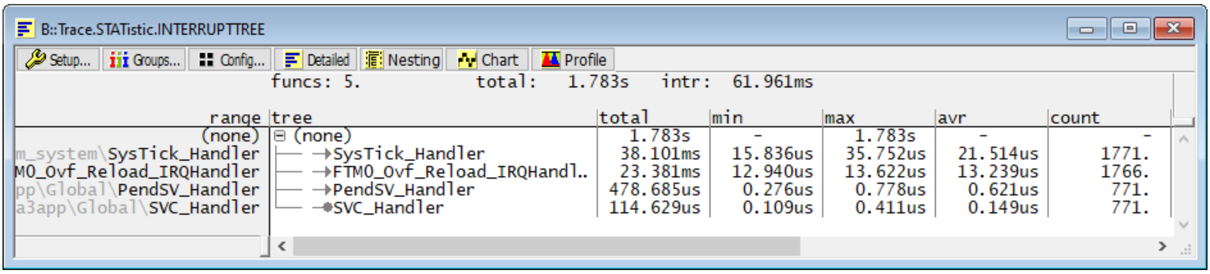
See also

■ <trace>.STATistic

Format:	*<trace>*.**STATistic.INTERRUPTTREE** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **NAME** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **THreshold** *<float>* **Track**

The results of this command shows a graphical tree of the interrupt nesting.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

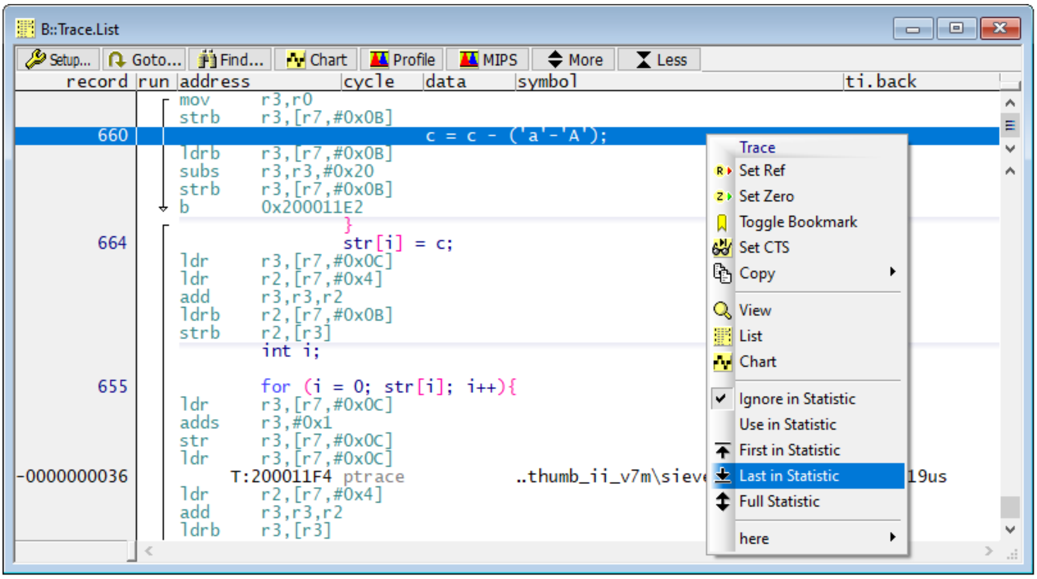


See also

- <trace>.STATistic
- CTS.STATistic.INTERRUPTTREE

Format:	*<trace>***.STATistic.LAST** *<value>* | *<time>* | *<string>*

The **Trace.STATistic** commands analyze the complete trace contents by default. The command **Trace.STATistic.LAST** allows to freely select an end point for the statistic analysis.



Example for <value>:

```
Trace.List                            ; display trace listing

Trace.STATistic.FIRST -123366.        ; select trace record -123366.
                                      ; as start point for the trace
                                      ; analysis

Trace.STATistic.LAST -36675.          ; select trace record -36675.
                                      ; as end point for the trace
                                      ; analysis

Trace.STATistic.Func                  ; perform a function run-time
                                      ; analysis
```

Example for <time>:

```
Trace.List TIme.ZERO DEFault    ; display trace listing

Trace.STATistic.LAST 468.2us    ; select trace record with timestamp
                                ; 468.2 µs (zero time) as end point for
                                ; the trace analysis

Trace.STATistic.Func            ; perform a function run-time analysis
                                ; from the beginning of the trace buffer
                                ; to the specified end point
```

See also

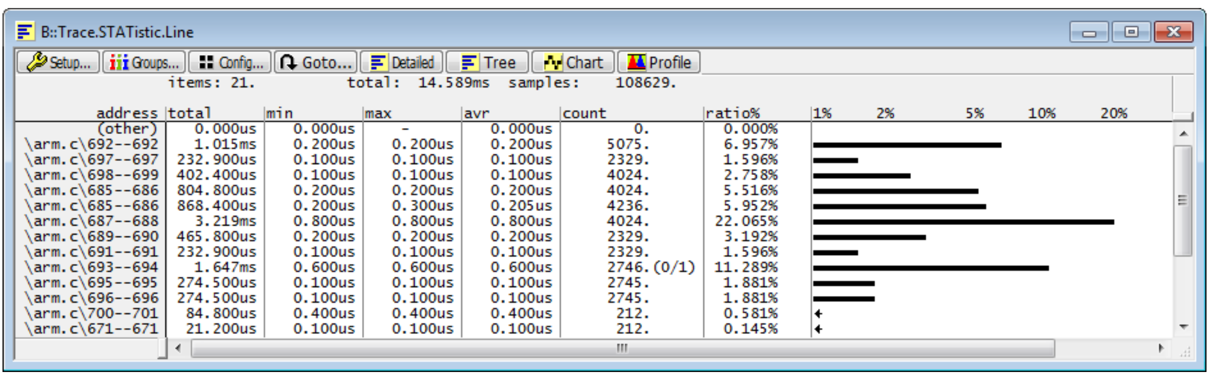
■ <trace>.STATistic ■ <trace>.STATistic.FIRST

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.Line** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<options>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* | **Filter** *<filter>* | **Address** *<address* | *range>* **ACCUMULATE** | **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

Analyzes the time spent in high-level source code lines.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.



See also

■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

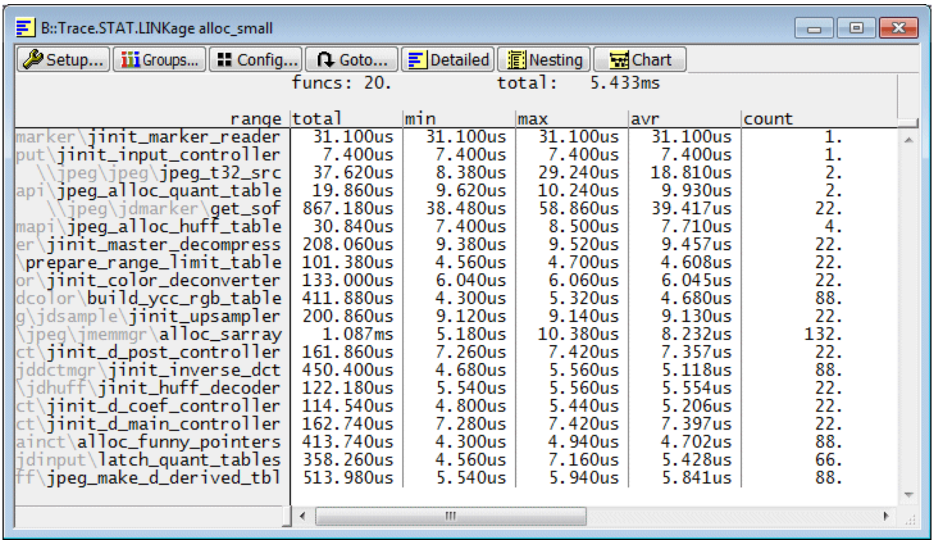
Format:	*<trace>*.**STATistic.LINKage** *<address>* [**%***<format>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **NAME** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **INTRROOT** | **INTRTASK** **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Performs a function run-time statistic for a single function itemized by its callers. The procedure for recording the data is the same as for the **<trace>.STATistic.Func** command.

<address>	Has to be the function entry address.
<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Example:

```
Trace.STATistic.LINKage alloc_small
```



range	total	min	max	avr	count
marker\jinit_marker_reader	31.100us	31.100us	31.100us	31.100us	1.
put\jinit_input_controller	7.400us	7.400us	7.400us	7.400us	1.
\\jpeg\jpeg\jpeg_t32_src	37.620us	8.380us	29.240us	18.810us	2.
api\jpeg_alloc_quant_table	19.860us	9.620us	10.240us	9.930us	2.
\\jpeg\jdmarker\get_sof	867.180us	38.480us	58.860us	39.417us	22.
mapi\jpeg_alloc_huff_table	30.840us	7.400us	8.500us	7.710us	4.
er\jinit_master_decompress	208.060us	9.380us	9.520us	9.457us	22.
\prepare_range_limit_table	101.380us	4.560us	4.700us	4.608us	22.
or\jinit_color_deconverter	133.000us	6.040us	6.060us	6.045us	22.
dcolor\build_ycc_rgb_table	411.880us	4.300us	5.320us	4.680us	88.
g\jdsample\jinit_upsampler	200.860us	9.120us	9.140us	9.130us	22.
\jpeg\jmemmgr\alloc_sarray	1.087ms	5.180us	10.380us	8.232us	132.
ct\jinit_d_post_controller	161.860us	7.260us	7.420us	7.357us	22.
jddctmgr\jinit_inverse_dct	450.400us	4.680us	5.560us	5.118us	88.
\jdhuff\jinit_huff_decoder	122.180us	5.540us	5.560us	5.554us	22.
ct\jinit_d_coef_controller	114.540us	4.800us	5.440us	5.206us	22.
ct\jinit_d_main_controller	162.740us	7.280us	7.420us	7.397us	22.
ainct\alloc_funny_pointers	413.740us	4.300us	4.940us	4.702us	88.
jdinput\latch_quant_tables	358.260us	4.560us	7.160us	5.428us	66.
ff\jpeg_make_d_derived_tbl	513.980us	5.540us	5.940us	5.841us	88.

The function alloc_small was called by the listed 20. functions. The dependency between the run-time of the function allow_small and its callers is analyzed.

See also

■ <trace>.STATistic ■ BMC.STATistic.LINKage ■ CTS.STATistic.LINKage

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.Measure** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | *<cpu>* | *<signals>* | **Port**[**.***<subitem>*] | **MARK**[**.***<marker>*] **LOW** | **HIGH**
<option>:	**FILE** | **ACCUMULATE** | **INCremental** | **FULL** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only)

This command allows to analyze the performance of a single signal. It is mainly used with **PowerProbe** or **PowerIntegrator**.

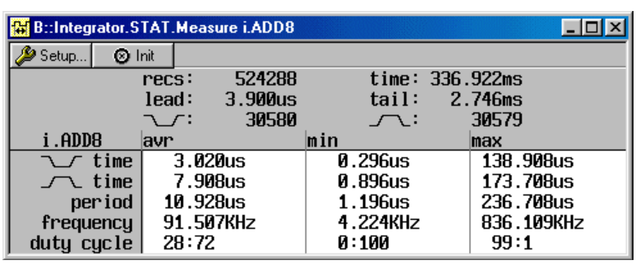
<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Typical application for the **<trace>.STATistic.Measure**:

- to check the best threshold level for a symmetric signal (e.g. a symmetric clock signal).
- to detect spikes (e.g. a signal has a defined period of 10.ns, detect if there is any much smaller period).

Example:

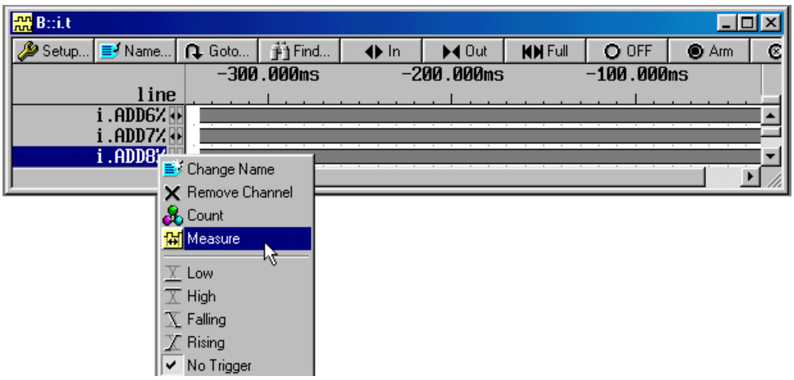
```
Trace.STATistic.Measure i.ADD8         ; analyze address line i.ADD8

; analyze the data line i.DAT6, start the analysis at record number
; -5000. and finish the analysis at record number -4000.
Trace.STATistic.Measure (-5000.)--(-4000.) i.DAT6
```



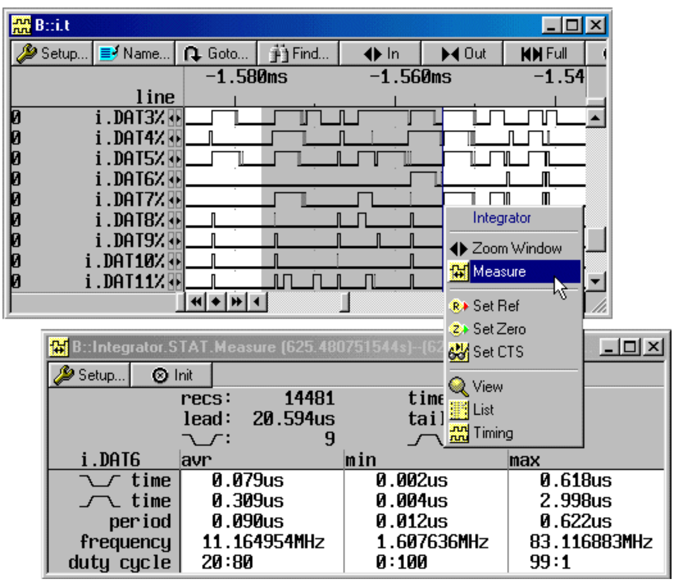
Description of the window elements:

recs	The number of records that are analyzed.
time	The time that is analyzed.
lead	The time from the beginning of the analysis until the first edge.
tail	The time from the last edge until the end of the analysis.
	The number of low states.
	The number of high states

The analysis can also be activated by selecting the signal in the **Trace.Timing** display and by using the pull-down menu provided via the right mouse button.



It is also possible to analyze only the selected part of the complete recording time.



See also

- <trace>.STATistic
- ▲ 'Release Information' in 'Legacy Release History'

<trace>.STATistic.MODULE — Code execution broken down by module

Format:	*<trace>*.**STATistic.MODULE** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

Shows a statistical analysis of the code execution broken down by symbol module. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

- <trace>.STATistic
- BMC.STATistic.MODULE
- <trace>.STATistic.PROGRAM
- CTS.STATistic.MODULE
- ▲ 'Release Information' in 'Legacy Release History'

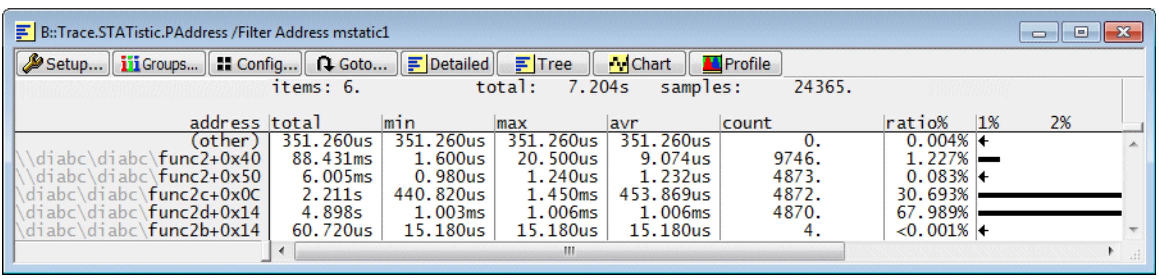
Format:	*<trace>*.**STATistic.PAddress** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* | **Filter** *<filter>* | **Address** *<address* | *range>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **Track**

The command provides a statistic about the instructions that accessed the data addresses. This command is generally used with the **/Filter Address** option.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Example:

```
Trace.STATistic.PAddress /Filter Address mstatic1
```



address	total	min	max	avr	count	ratio%	1% 2%
(other)	351.260us	351.260us	351.260us	351.260us	0.	0.004%	
\\diabc\diabc\func2+0x40	88.431ms	1.600us	20.500us	9.074us	9746.	1.227%	
\\diabc\diabc\func2+0x50	6.005ms	0.980us	1.240us	1.232us	4873.	0.083%	
\diabc\diabc\func2c+0x0C	2.211s	440.820us	1.450ms	453.869us	4872.	30.693%	
\diabc\diabc\func2d+0x14	4.898s	1.003ms	1.006ms	1.006ms	4870.	67.989%	
\diabc\diabc\func2b+0x14	60.720us	15.180us	15.180us	15.180us	4.	<0.001%	

See also

- <trace>.STATistic
- 'Release Information' in 'Legacy Release History'

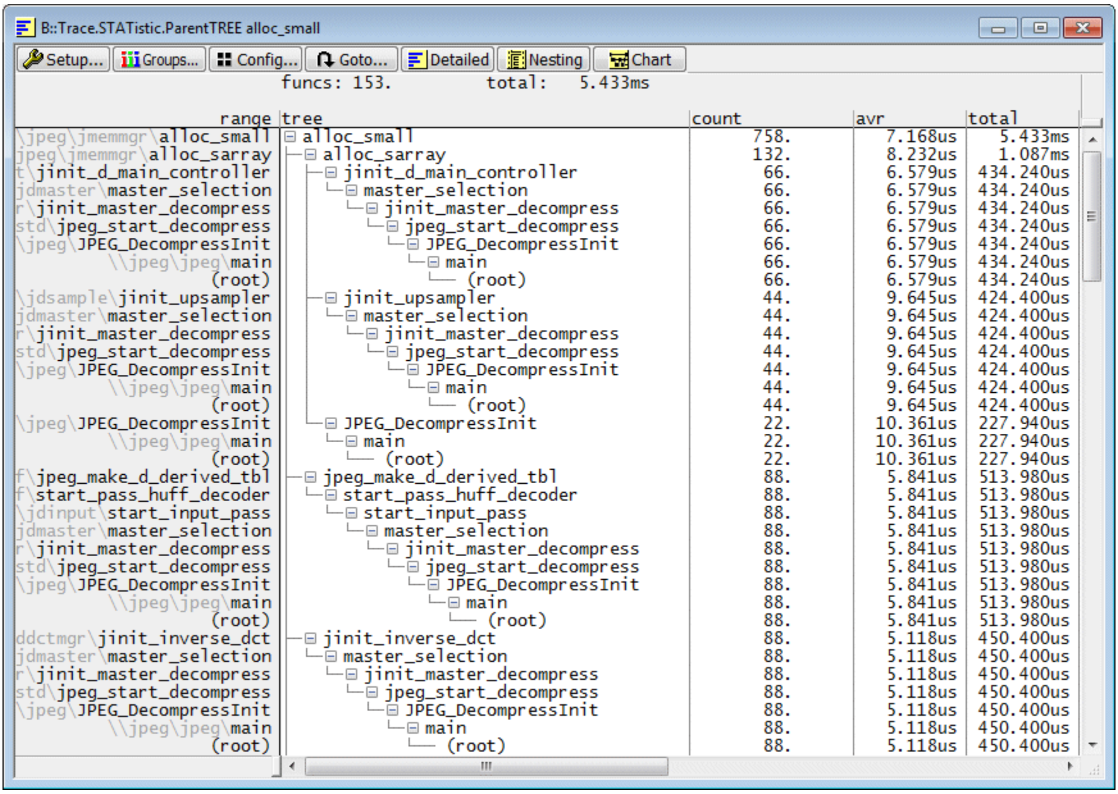
Format:	*<trace>*.**STATistic.ParentTREE** *<address>* [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** **TREE** | **LEVEL** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **INTRROOT** | **INTRTASK** **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

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

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Example:

```
Trace.STATistic.ParentTREE alloc_small
```



See also

- <trace>.STATistic
- BMC.STATistic.ParentTREE
- <trace>.STATistic.ChildTREE
- CTS.STATistic.ParentTREE

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.PROCESS** [**/***<option>*]
<option>:	**ARTIAP**

Starts re-processing of statistic and chart windows.

ARTIAP	Option for AUTOSAR Real-Time Interface on Adaptive Platform trace decoding. Decode MIPI STP (System Trace Protocol) format trace which is defined in ARTI Trace Driver on AUTOSAR Adaptive Platform. It can be used to process ARTIAP trace without executing **<trace>.Chart.TASK/TASKState** or **<trace>.STATistic.TASK/TASKState** related commands.

See also

- <trace>.STATistic

Format:	*<trace>*.**STATistic.PROGRAM** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* | **Filter** *<filter>* | **Address** *<address* | *range>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **Sort** *<item>* | **Track**

Shows a statistical analysis of the code execution broken down by loaded object files (program). The loaded programs can be displayed with the command **sYmbol.Browse ***.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

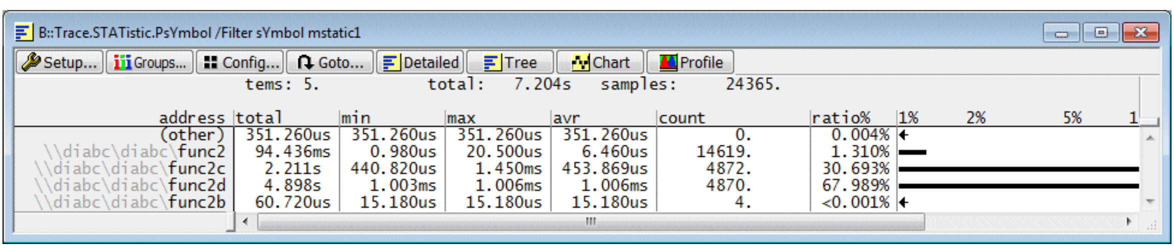
- <trace>.STATistic
- <trace>.STATistic.MODULE
- BMC.STATistic.PROGRAM
- CTS.STATistic.PROGRAM
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.PsYmbol** [%*<format>*] [*<list_items>* …] [/*<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<number>* | **SplitCORE** | **MergeCORE** | **JoinCORE** (SMP tracing only) **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* **Filter** *<filter>* **Address** *<address* | *range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

The command provides a statistic about the functions that accessed the data addresses. This command is generally used with the **/Filter Address** option.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

```
Trace.STATistic.PsYmbol /Filter sYmbol mstatic1

Trace.STATistic.PsYmbol /Filter sYmbol mstatic1 CYcle Write
```

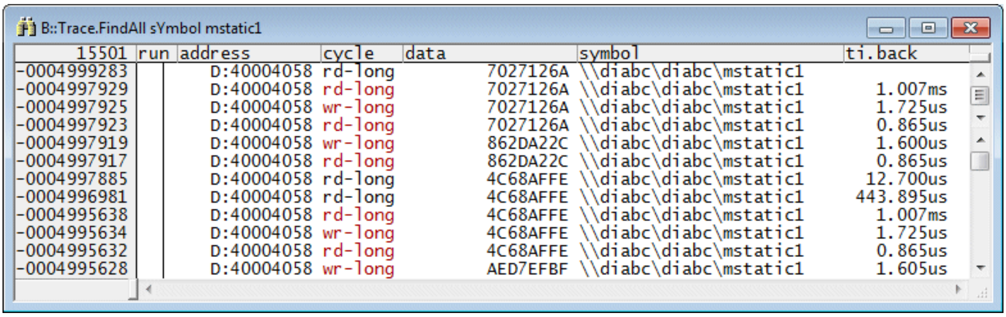


Preconditions:

- Has to be implemented for the processor architecture in use.
- Data access has to be clearly assignable to an instruction.

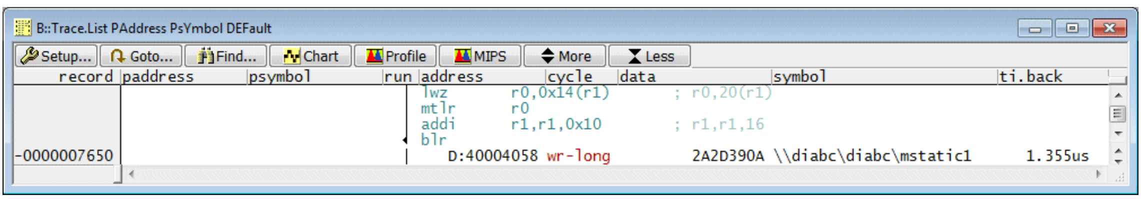
If TRACE32 was able to clearly assign the data access to an instruction can be checked as follows:

```
Trace.FindAll sYmbol mstatic1
```



A red cycle type indicates that a clear assignment was not possible.

```
; PAddress: address of instruction that performed the data access
; PsYmbol: symbolic address of instruction that performed the data access
Trace.List PAddress PsYmbol DEFault
```



Both columns are empty if no clear assignment is possible.

See also

- ■ <trace>.STATistic
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.RUNNABLE** [%*<format>*] [*<list_items>* ...] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<item>* | **SplitCORE** | **MergeCORE** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **InterVal** *<time>* **Filter** *<item>* **Address** *<address* | *range>* **INCremental** | **FULL** **ACCUMULATE** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Analyzes the function nesting and calculates the time spent in AUTOSAR Runnables and the number of Runnable calls.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

On TriCore AURIX there's a solution available for the Vector AUTOSAR tools that uses an automated instrumentation to trace runnables on all cores with minimum overhead. See *~~/demo/env/vector/rte_profiling*.

Otherwise, all functions that start an AUTOSAR “Runnable” have to be marked with the command **sYmbol.MARKER.Create RUNNABLESTARTPLUSSTOP**. Please refer to **“Trace Export for Third-Party Timing Tools”** (app_timing_tools.pdf) for more information.

See also

■ <trace>.STATistic ■ CTS.STATistic.RUNNABLE ■ TASK.Create.RUNNABLE ■ TASK.List.RUNNABLES
▲ 'Runnable Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'

Format:	*<trace>***.STATistic.RUNNABLE** *<address* | *name>* [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<item>* | **SplitCORE** | **MergeCORE** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **InterVal** *<time>* | **Filter** *<item>* **Address** *<address* | *range>* **INCremental** | **FULL** **ACCUMULATE** **CLOCKS** **NoMerge** **Number** *<record>* **LOG** | **LINear**

Analyzes the time spent in AUTOSAR Runnables. This is currently limited to runnable analysis based on regular function tracing.

<option>	Refer to **Options** under **<trace>.STATistic**.

See also

■ <trace>.STATistic
▲ 'Runnable Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'

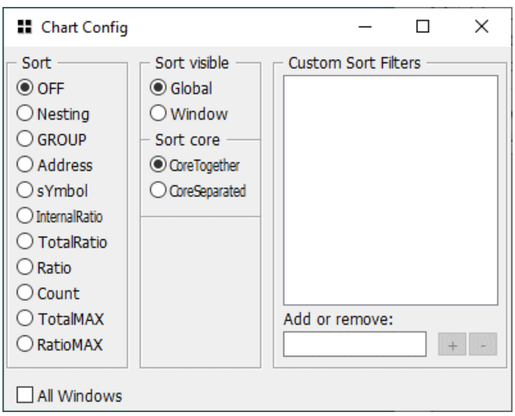
<trace>.STATistic.Sort

Format:	*<trace>*.**STATistic.Sort** [*<sort_visible>*] [*<sort_core>*] [*<sort>* [*<filters>*]]
<sort_ visible>:	**Window** | **Global**
<sort_core>:	**CoreTogether** | **CoreSeparated** (SMP systems only)
<sort>:	**OFF** **Address** | **sYmbol** [*<wildcard_list …>*] | **GROUP** **Nesting** | **InternalRatio** | **TotalRatio** **Ratio** **Count** **TotalMAX** | **RatioMAX**

Specify sorting criterion for the results of the command groups **<trace>.STATistic** and **<trace>.Chart**.

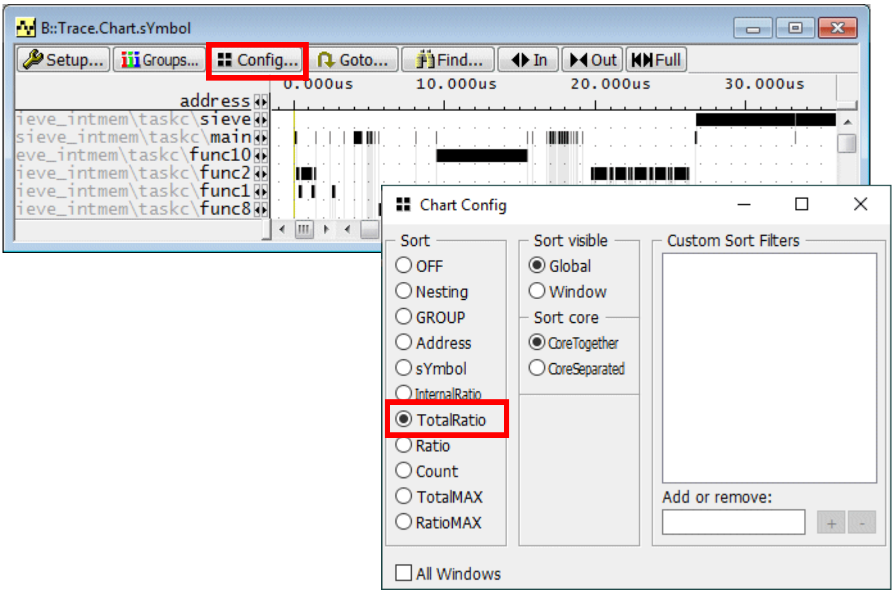
After item <sort>, a list of whitespace separated filters can be specified. The items that match any of the filters are shown first, then the rest if the items is shown according to the selected criterion. The filters can include wildcards (* and ?).

If the command is entered without parameters, a **Trace.STATistic.Sort** dialog is displayed.



The sorting criterion specified by **Trace.STATistic.Sort** applies to all **<trace>.STATistic** and **<trace>.Chart** analysis windows (check box All Windows ON).

To specify the sorting criterium for an individual statistic window use the **Config** button of this statistic window or use the **/Sort** option when you enter the command.




```
Trace.Chart.sYmbol /Sort TotalRatio          ; sort the time chart by the
                                             ; criterion TotalRatio
```

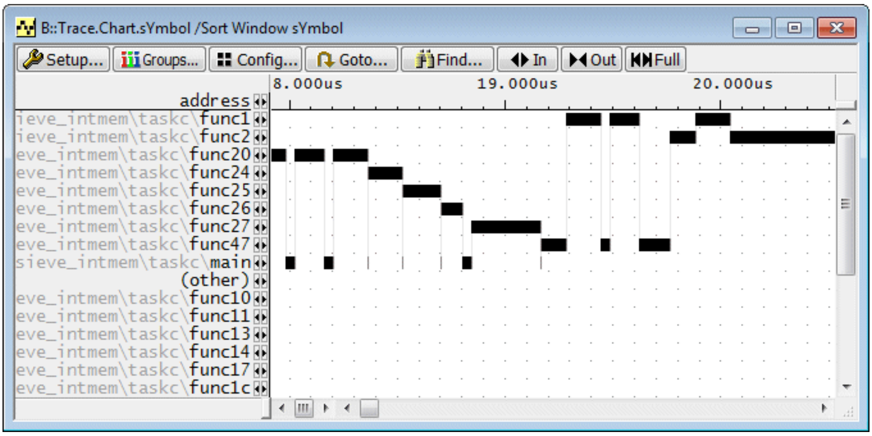
Default Sorting Criterion

OFF is the default mode for most statistic windows. **OFF** means that the analyzed items are displayed in their recording order.

Statistic windows that are focused on the program's call hierarchy e.g **Trace.STATistic.TREE** use **Nesting** as default mode.

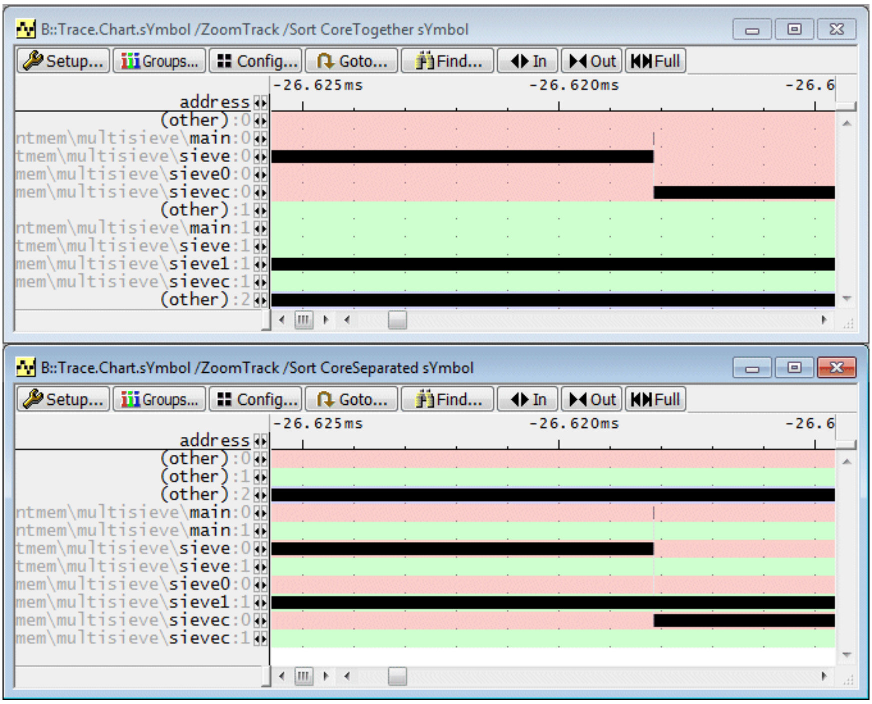
Global (default)	The sorting criterion is strictly maintained.
Window	The sorting criterion is applied. The analyzed items active in the displayed time interval are displayed first, followed by the non-active items. **Window** might be useful if you scroll horizontally.

```
Trace.Chart.sYmbol /Sort Window sYmbol
```



CoreTogether (default)	The analyzed items are displayed per core. Additional sorting criteria apply to this per core order.
CoreSeparately	The core information has no impact on the sorting order.

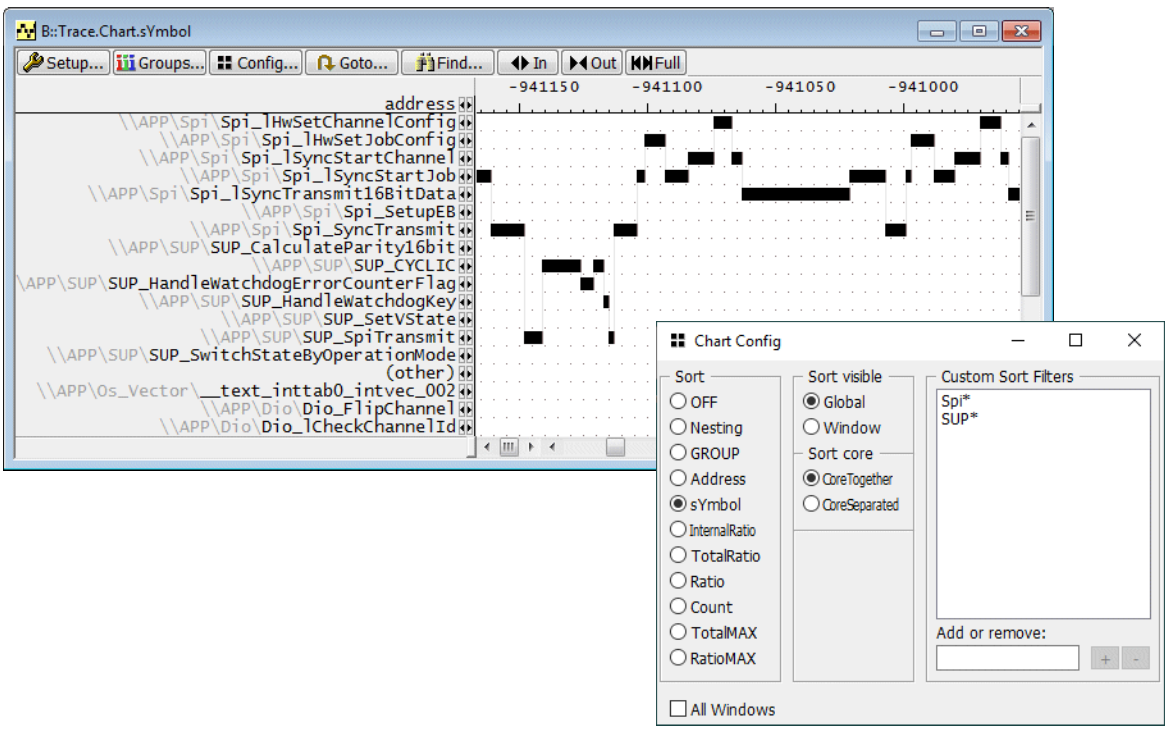
```
Trace.Chart.sYmbol /ZoomTrack /Sort CoreTogether sYmbol

Trace.Chart.sYmbol /ZoomTrack /Sort CoreSeparated sYmbol
```



Address	Sort result by address
sYmbol [<*filters*>]	Sort result alphabetically by symbol names
GROUP	Sort result by their grouping
Count	Sort analyzed items by their occurrence

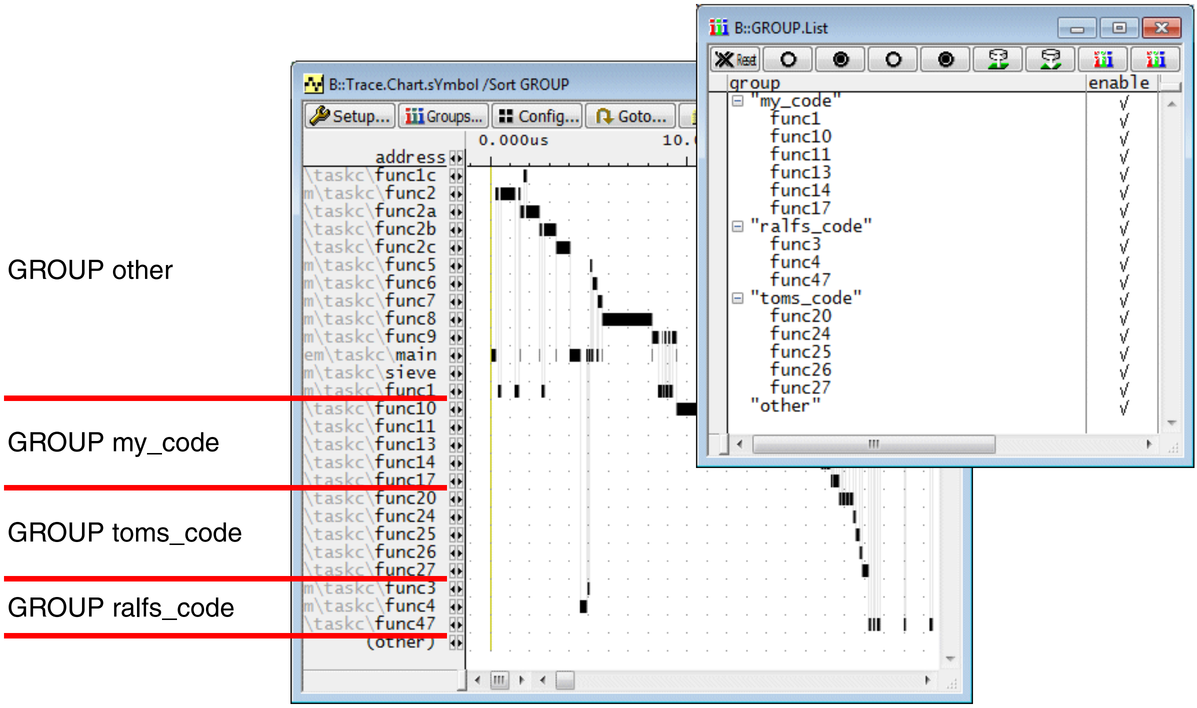
Example for sort criterion **sYmbol** [<*filters*>].

```
; display items starting with string "SPI" first, then items starting
; with string "SUP" then rest

Trace.STATistic.Sort sYmbol Spi* SUP*

Trace.Chart.sYmbol
```



Example for sort criterion **GROUP**.

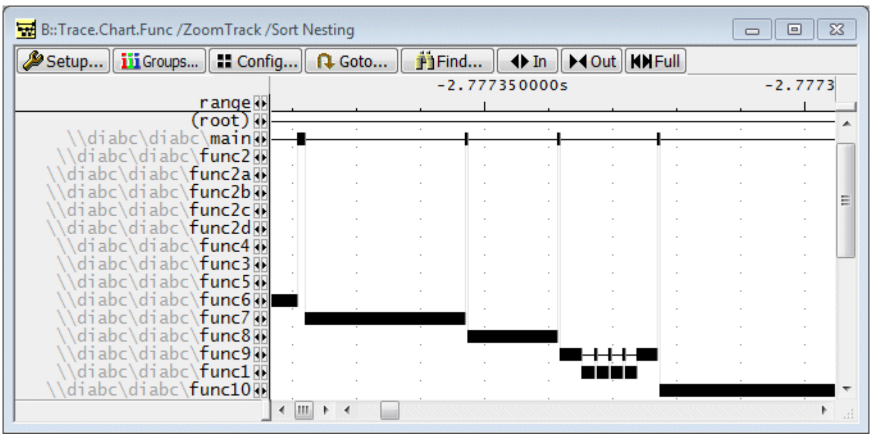
```
GROUP.List

Trace.STATistic.Sort GROUP

Trace.Chart.sYmbol
```



Nesting	Calling functions are displayed atop of called function.
InternalRatio	Sort result be internal ratio. **InternalRatio:** *<time_in_function>/<total_measurement_time>* as a numeric value.
TotalRatio	Sort result by total ratio. **InternalRatio:** *<total_time_of_function>/<total_measurement_time>* as a numeric value, *<total_time_of_function>* includes called subfunctions and traps.

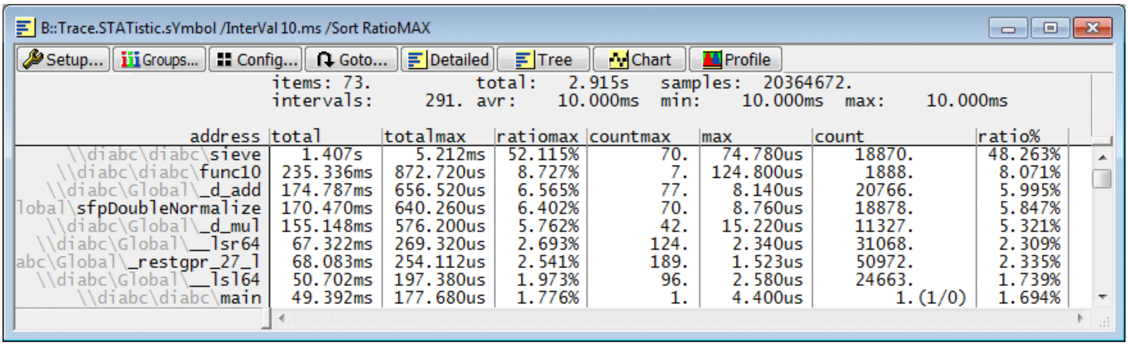
Example for criterion **Nesting**.

```
Trace.Chart.Func /ZoomTrack /Sort Nesting
```



Ratio	Sort analyzed items by their ratio.
TotalMAX	Flat analysis with InterVal option only. Sort analyzed items by maximal total time per specified interval.
RatioMAX	Flat analysis with InterVal option only. Sort analyzed items by maximal ratio per specified interval.

```
Trace.STATistic.sYmbol /InterVal 10.ms /Sort RatioMAX
```



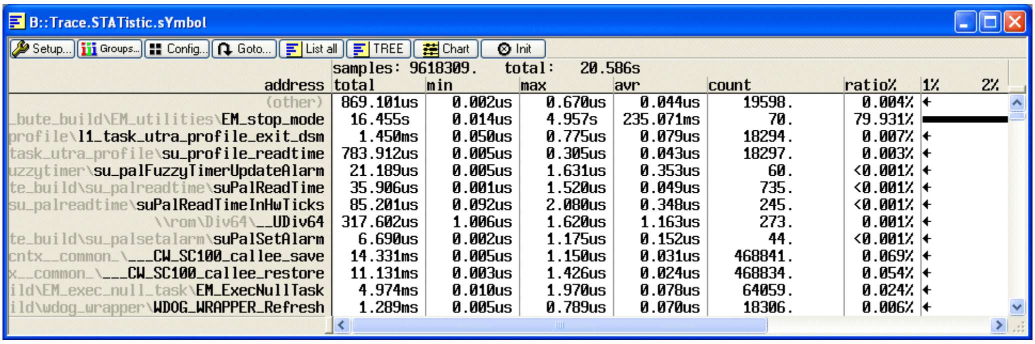
See also

- ■ <trace>.STATistic
- ▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.sYmbol** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<list_item>:	**DEFault** | **ALL** **Total** | **MIN** | **MAX** | **AVeRage** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **NAME** | **CountRatio** | **CountBAR** **CountChange** | **CountFirst** | **CountALL**
<options>:	**FILE** | **FlowTrace** | **BusTrace** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **CORE** *<item>* | **SplitCORE** | **MergeCORE** **LABEL** | **NoLABEL** | **INLINE** | **NoINLINE** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountAll** **InterVal** *<time>* | **Filter** *<item>* | **Sort** *<item>* | **Track** **Address** *<function1>***||***<function2>* … **Address** *<function_m>***--***<function_n>***ACCUMULATE** **INCremental** | **FULL** | **CLOCKS**

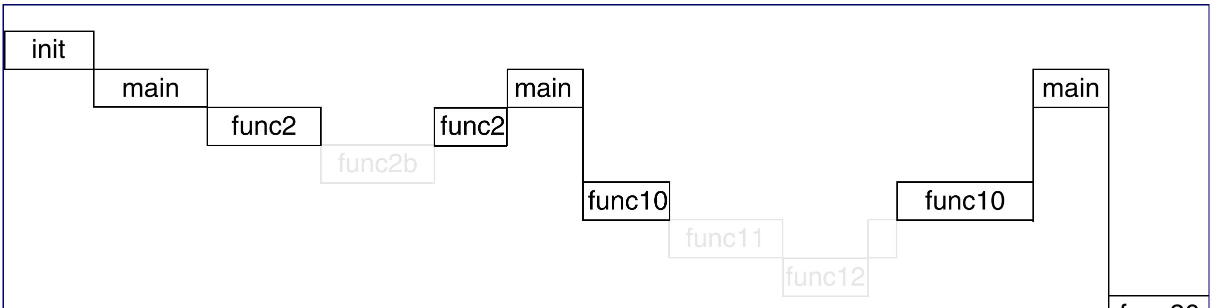
The execution time in different symbol regions is displayed. Displayed are the number of entries into the range and the time spent in the range.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

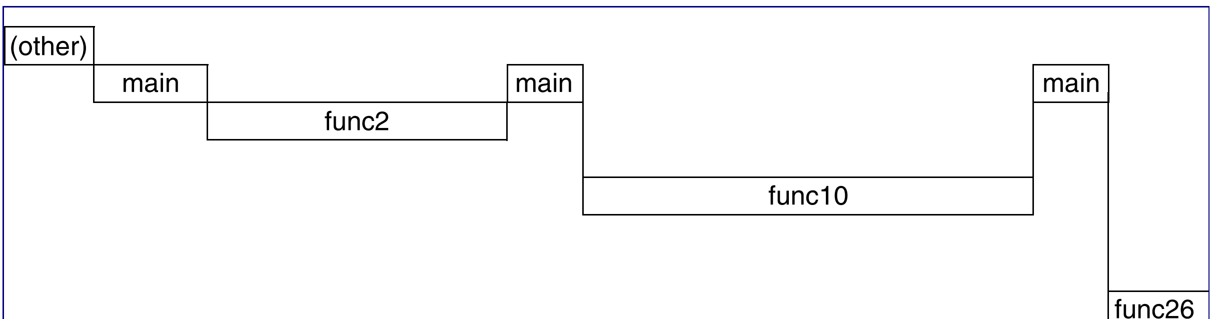


Example: filter the specified functions out of the trace stream and then analyze the filtered trace information

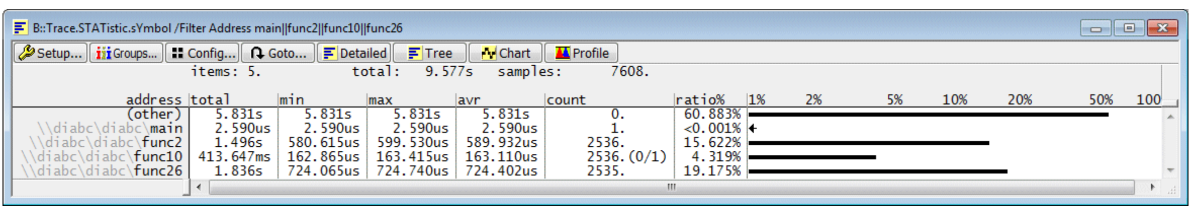
```
Trace.STATistic.sYmbol /Filter Address main||func2||func10||func26
```



Recording (filtered functions are displayed in black)

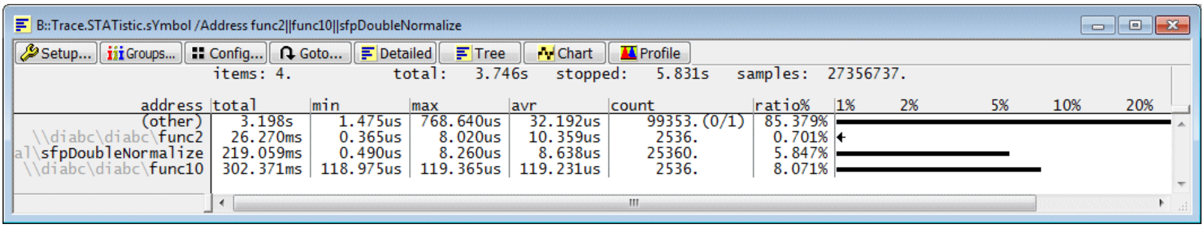


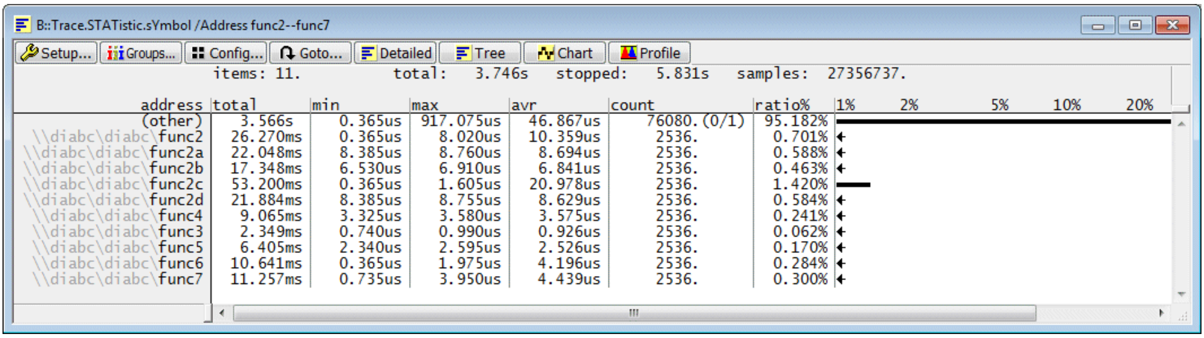
Analysis result



Example: Perform statistic on specified functions, assign statistic information for all other functions to (other)

```
Trace.STATistic.sYmbol /Address func2||func10||sfpDoubleNormalize
Trace.STATistic.sYmbol /Address func2--func7
```





See also

- ■ <trace>.Chart.sYmbol
- ■ <trace>.STATistic
- ■ BMC.STATistic.sYmbol
- ■ CTS.STATistic.sYmbol

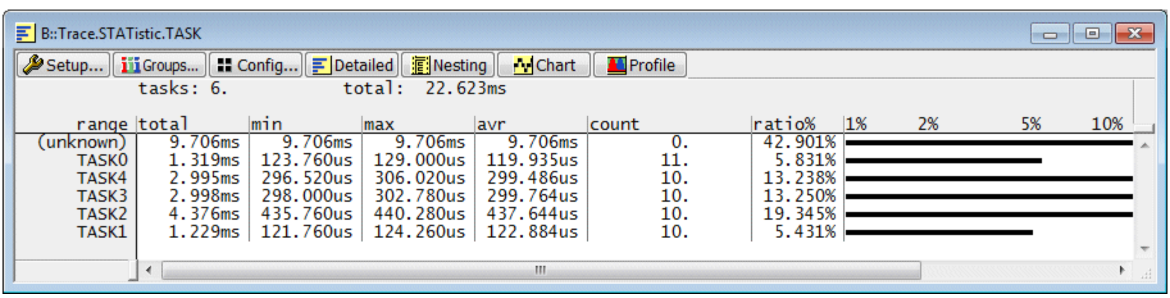
▲ 'Release Information' in 'Legacy Release History'
▲ 'Function Run-Times Analysis - SMP Instance' in 'Training MPC5xxx/SPC5xx Nexus Tracing'

Format:	*<trace>*.**STATistic.TASK** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** | **Filter** *<item>* | **Sort** *<item>* | **Track** **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **ARTIAP**

Task run-times are analyzed. **“OS-aware Tracing”** (trace32_concepts.pdf) has to be enabled in order to use this command.

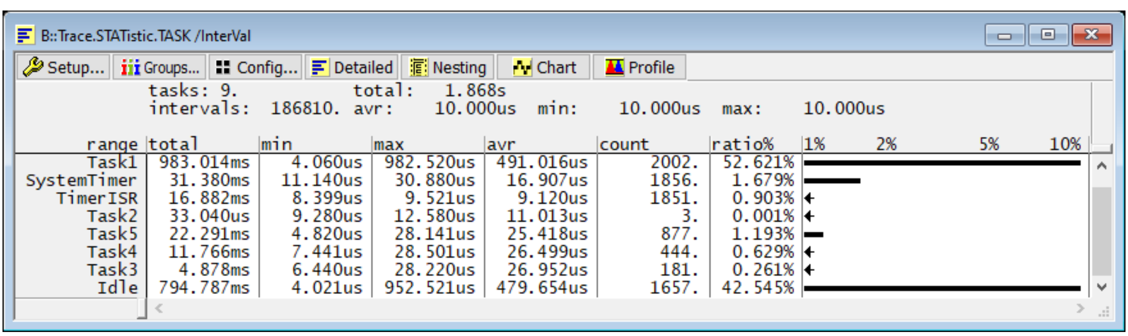
<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Survey	
tasks	Number of recorded tasks.
total	Time period recorded by trace.



Task details		
column	***<list_item>***	***description***
range		Task name **(unknown)**: TRACE32 assigns all trace information generated before the first task information to the (unknown) task.
total	**Total**	Time period in the task during the recorded time period.
min, max, avr	**MIN**	Shortest, longest and average time in task.
count	**Count**	Number of time in task.
ratio	**Ratio**	Ratio of time in the task with regards to the total time period recorded.
(graphical bar)	**BAR.LOG**	Ratio of time in the task with regards to the total time period recorded graphically.
group	**GROUP**	Display of group name assigned by command **GROUP.CreateTASK**.

Survey (InterVal option)	
tasks	Number of recorded tasks.
total	Time period recorded by trace.
intervals	Number of intervals.
min, max, avr	Shortest, longest and average interval length.



Task details (InterVal option)		
column	***item***	***description***
totalmax	**TotalMAX**	Longest time period in the task within an interval.
ratiomax	**RatioMAX**	Highest ratio of time in the task within an interval.
countmax	**CountMAX**	Highest number of time in the task within an interval
totalmin	**TotalMIN**	Shortest time period in the task within an interval.
ratiomin	**RatioMIN**	Shortest ratio of time in the task within an interval.
countmin	**CountMIN**	Shortest number of time in the task within an interval

See also

■ <trace>.STATistic
■ <trace>.STATistic.TASKFunc
■ BMC.STATistic.TASK
■ CTS.STATistic.TASK

▲ 'CPU Load Measurement' in 'Application Note Profiling on AUTOSAR CP with ARTI'
▲ 'Release Information' in 'Legacy Release History'
▲ 'OS-Aware Tracing' in 'Training Arm CoreSight ETM Tracing'
▲ 'OS-Aware Tracing - Single Core' in 'Training MPC5xxx/SPC5xx Nexus Tracing'
▲ 'OS-Aware Tracing - SMP Systems' in 'Training MPC5xxx/SPC5xx Nexus Tracing'

<trace>.STATistic.TASKFunc — Task related function run-time analysis

Format:	<trace>.**STATistic.TASKFunc** [%<format>] [<list_items> …] [/<option>] (legacy)

For details, refer to **<trace>.STATistic.Func**.

See also

- ■ <trace>.STATistic.TASK ■ <trace>.STATistic
- ▲ 'Release Information' in 'Legacy Release History'

<trace>.STATistic.TASKINFO — Context ID special messages

Format:	*<trace>*.**STATistic.TASKINFO** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** | **Track**

Displays a run-time statistic of special messages written to the Context ID register for ETM trace. The range of special values has to be reserved with the **ETM.ReserveContextID** command. These special values are then not interpreted for task switch or memory space switch detection. This can be used for cores without data trace to pass data by the target application to the trace tool by writing to the ContextID register.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

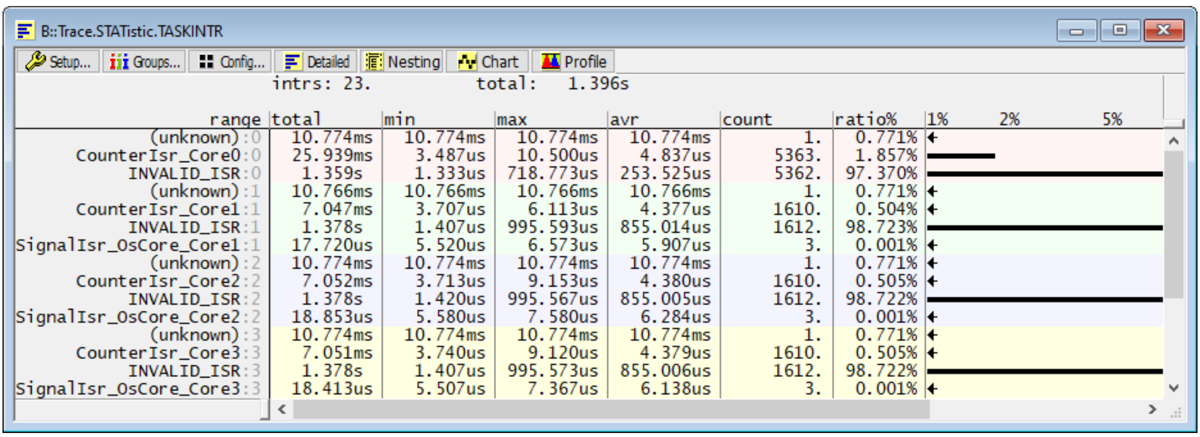
See also

- ■ <trace>.STATistic ■ BMC.STATistic.TASKINFO ■ CTS.STATistic.TASKINFO

Format:	*<trace>*.**STATistic.TASKINTR** [**%***<format>*] [*<list_item>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** | **Track**

Displays an ORTI based ISR2 run-time statistic. This feature can only be used if the ISR2 can be traced based on the information provided by the ORTI file.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.



See also

■ <trace>.STATistic ■ BMC.STATistic.TASKINTR ■ CTS.STATistic.TASKINTR

▲ 'ISR2 Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'
▲ 'Trace Features' in 'OS Awareness Manual OSEK/ORTI'

Format:	*<trace>*.**STATistic.TASKKernel** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** | **Track**

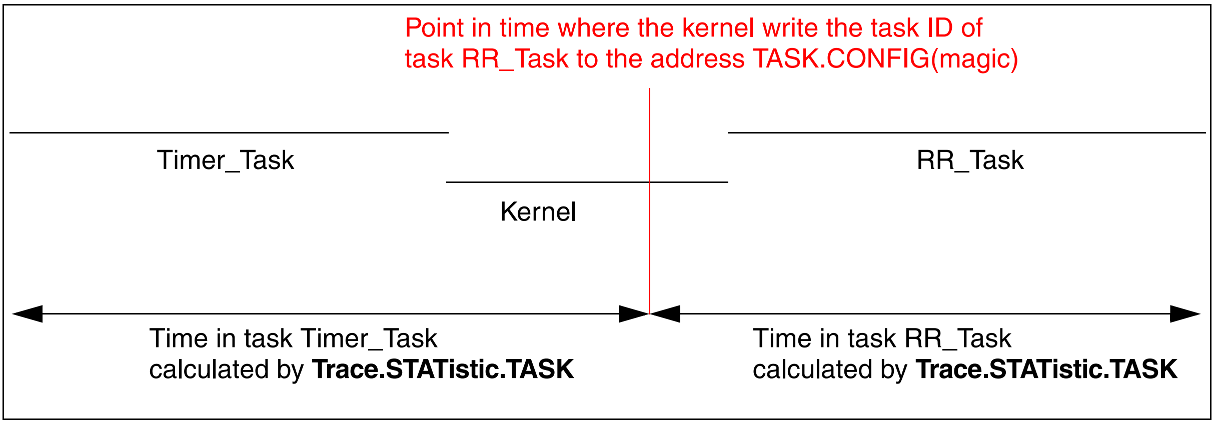
The command **Trace.STATistic.TASKKernel** refines the command **Trace.STATistic.TASK** for RTOS systems that don´t assign a task ID to the kernel. In such a case no task run-time is calculated for the kernel if the command **Trace.STATistic.TASK** is used.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

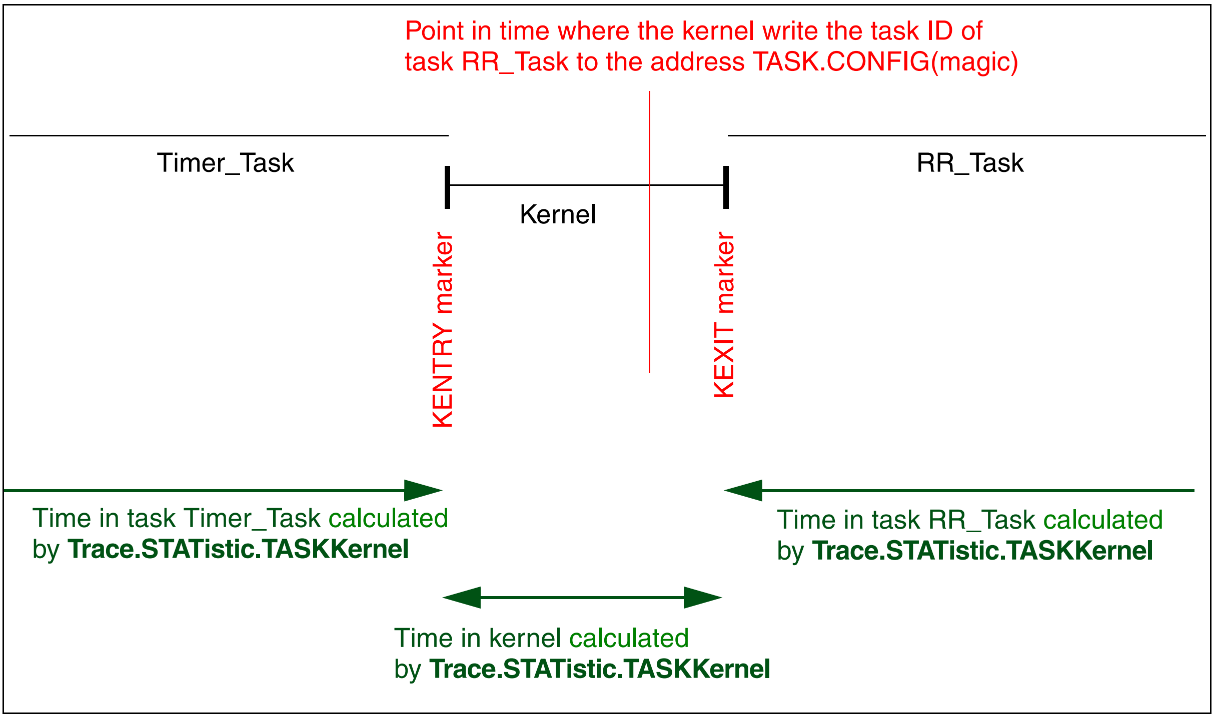
If the TRACE32 **TASK** awareness was configured, TRACE32 implies that the kernel writes the identifier of the current task to the address **TASK.CONFIG(magic)**.

```
PRINT TASK.CONFIG(magic)
```

Measurement performed by Trace.STATistic.TASK (no task ID for the kernel):



Measurement performed by Trace.STATistic.TASKKernel (KENTRY/KEXIT marker):

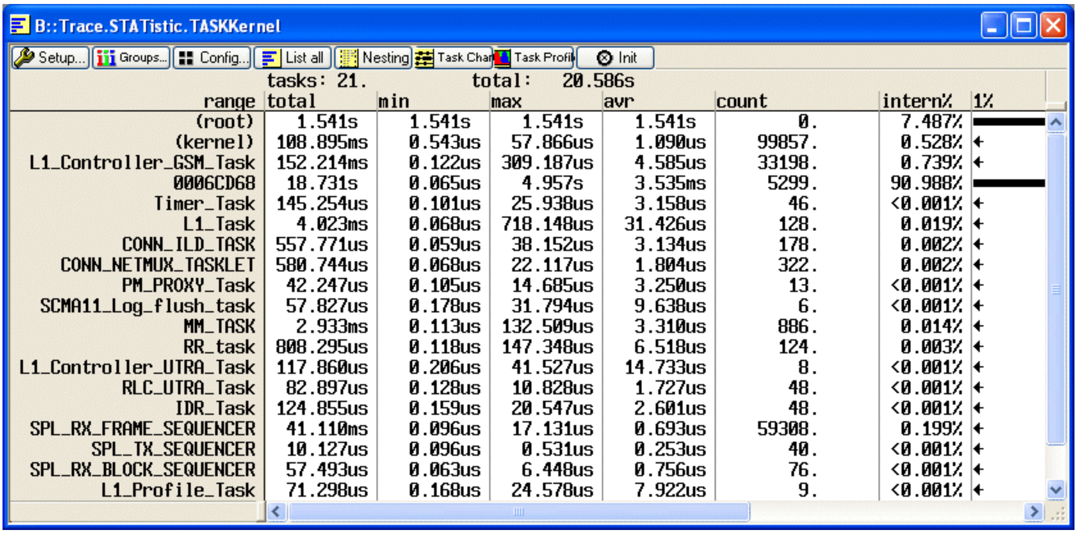


The refined measurement of **Trace.STATistic.TASKKernel** requires that the kernel entries and kernel exits are marked by the command **sYmbol.MARKER.Create**.

```
sYmbol.MARKER.Create KENTRY os_prologue    ; mark the address os_prologue
                                           ; as kernel entry point

sYmbol.MARKER.Create KEXIT os_epilogue     ; mark the address os_epilogue
                                           ; as kernel exit point

sYmbol.MARKER.list                         ; list all markers
```

Advanced example for RTOS RTXC on a StarCore CPU:

```
; mark all interrupt service routines as kernel entries
sYmbol.ForEach "sYmbol.MARKER.Create KENTRY *" "_isr_*"

; mark all RTE instructions in the specified program range as kernel exit
Data.Find P:RTXCProlog--P:RTXCProlog_end %Word 0x9f73
WHILE FOUND()
(
     sYmbol.MARKER.Create KEXIT P:TRACK.ADDRESS()
     Data.Find
)

sYmbol.MARKER.list
```



range	total	min	max	avr	count	intern%	1%
(root)	1.541s	1.541s	1.541s	1.541s	0.	7.487%	
(kernel)	108.895ms	0.543us	57.866us	1.090us	99857.	0.528%	
L1_Controller_GSM_Task	152.214ms	0.122us	309.187us	4.585us	33198.	0.739%	
0006CD68	18.731s	0.065us	4.957s	3.535ms	5299.	90.988%	
Timer_Task	145.254us	0.101us	25.938us	3.158us	46.	<0.001%	
L1_Task	4.023ms	0.068us	718.148us	31.426us	128.	0.019%	
CONN_ILD_TASK	557.771us	0.059us	38.152us	3.134us	178.	0.002%	
CONN_NETMUX_TASKLET	580.744us	0.068us	22.117us	1.804us	322.	0.002%	
PM_PROXY_Task	42.247us	0.105us	14.685us	3.250us	13.	<0.001%	
SCMA11_Log_flush_task	57.827us	0.178us	31.794us	9.638us	6.	<0.001%	
MM_TASK	2.933ms	0.113us	132.509us	3.310us	886.	0.014%	
RR_task	808.295us	0.118us	147.348us	6.518us	124.	0.003%	
L1_Controller_UTRA_Task	117.860us	0.206us	41.527us	14.733us	8.	<0.001%	
RLC_UTRA_Task	82.897us	0.128us	10.828us	1.727us	48.	<0.001%	
IDR_Task	124.855us	0.159us	20.547us	2.601us	48.	<0.001%	
SPL_RX_FRAME_SEQUENCER	41.110ms	0.096us	17.131us	0.693us	59308.	0.199%	
SPL_TX_SEQUENCER	10.127us	0.096us	0.531us	0.253us	40.	<0.001%	
SPL_RX_BLOCK_SEQUENCER	57.493us	0.063us	6.448us	0.756us	76.	<0.001%	
L1_Profile_Task	71.298us	0.168us	24.578us	7.922us	9.	<0.001%	

If the processor allows to restrict the trace information output to the program flow and specific write accesses, it is recommended to restrict the output to the program flow plus write cycles to `task.config(magic)`, since more information can be recorded into the trace buffer.

```
Break.Set TASK.CONFIG(magic) /Write /TraceData

Go

Break

Trace.STATistic.TASKKernel
```

See also

- <trace>.STATistic
- BMC.STATistic.TASKKernel
- CTS.STATistic.TASKKernel

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.TASKLOCK** *<address>* | *<name>* [**/***<option>*]
<option>:	**FILE** **FlowTrace** | **BusTrace** **List** *<item>* **Filter** *<item>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>*

Analyzes lock accesses from tasks.

<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

See also

■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.TASKORINTERRUPT %***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** | **Track**

Analyzes Task and interrupt run-times in one single window.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

- <trace>.STATistic
- CTS.STATistic.TASKORINTERRUPT
- BMC.STATistic.TASKORINTERRUPT

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.TASKORINTRState** [**%***<format>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_items>:	**DEFault** | **DefaultByItem** | **ALL** | **ALLByItem** *<state>***[.***<state_item>***]** **TIme** [**.***<state>*] | **MAX** [**.***<state>*]| **MIN** [**.***<state>*] | **AVeRage** [**.***<state>*] **Count** [**.***<state>*] | **Total** [**.***<state>*] **Count** | **Ratio** | **BAR.log** | **BAR.LINear**
<state>:	**UND** | **RUN** | **RDY** | **WAIT** | **REL** | **ACT** | **SUSP** | **INTR**
<state_item>:	**all** | **Total** | **MIN** | **MAX** | **AVeRage** | **Count** | **Ratio** | **BAR**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **Track**

The time tasks and interrupt spent in different states is measured.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Description of the states:

UND	Undefined
RUN	Running
RDY	Ready
WAIT	waiting
REL	Released
ACT	Activated

SUSP	Suspended
INTR	Interrupted

Before using this function the task and interrupt state transitions must be sampled by the trace. This feature is highly dependent on the used RTOS kernel, and needs the **TASK** to be configured. Please see kernel specific **"OS Awareness Manuals"** manuals for more information.

Please refer for more information to **<trace>.STATistic.TASKState**.

See also

- ■ <trace>.STATistic
- ▲ 'ISR2 Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'
- ▲ 'Release Information' in 'Legacy Release History'

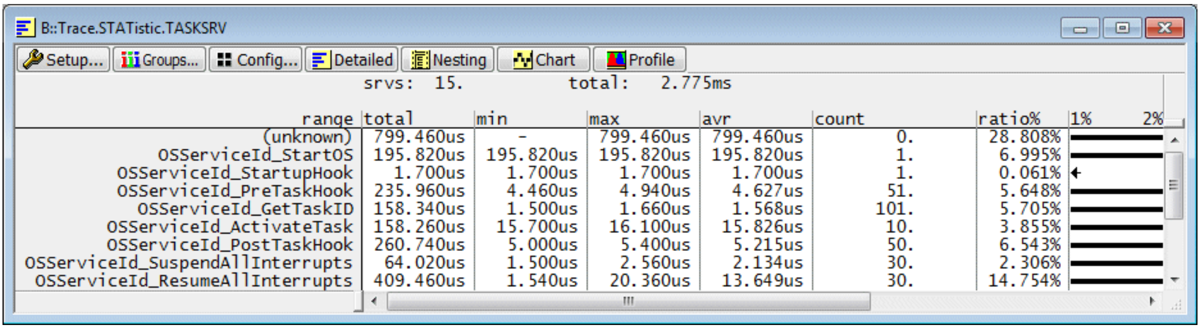
<trace>.STATistic.TASKSRV — Analysis of time in OS service routines

Format:	*<trace>*.**STATistic.TASKSRV** [**%***<format>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** | **Track**

The time spent in OS service routines and the number of calls is measured.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

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



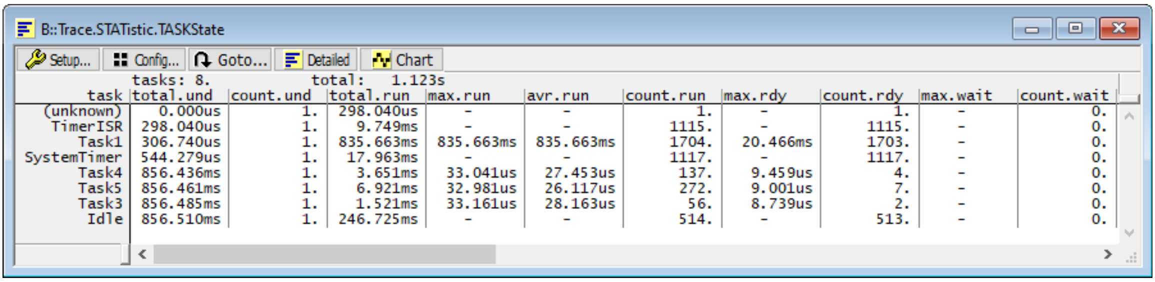
See also

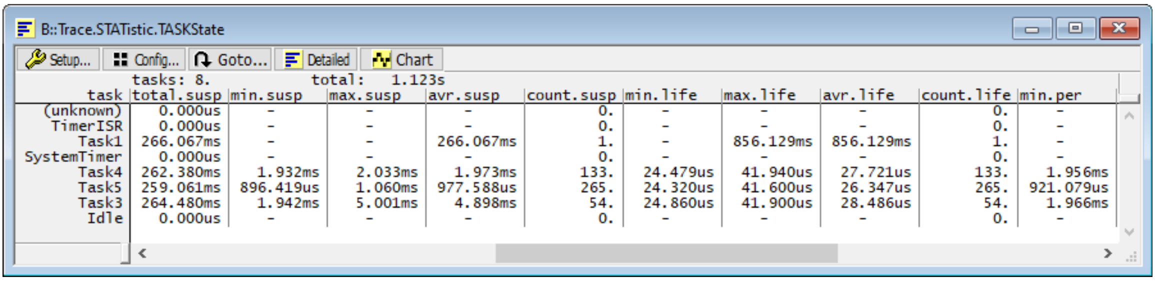
- <trace>.STATistic
- BMC.STATistic.TASKSRV
- CTS.STATistic.TASKSRV
- ▲ 'Release Information' in 'Legacy Release History'

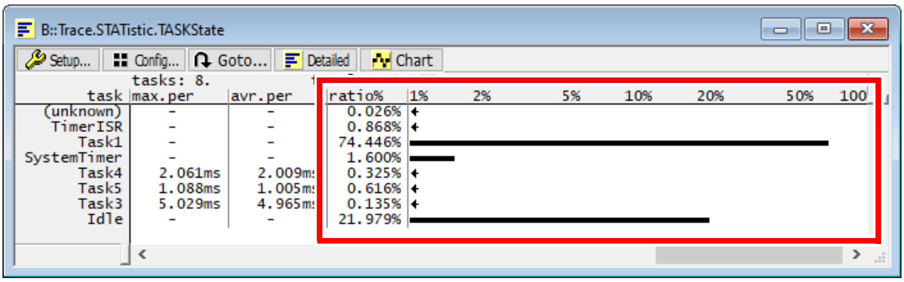
Format:	*<trace>***.STATistic.TASKState** [**%***<format>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_items>:	**DEFault** | **DefaultByItem** | **ALL** | **ALLByItem** *<state>***[.***<state_item>***]** **TIme** [**.***<state>*] | **MAX** [**.***<state>*]| **MIN** [**.***<state>*] | **AVeRage** [**.***<state>*] **Count** [**.***<state>*] | **Total** [**.***<state>*] **Count** | **Ratio** | **BAR.log** | **BAR.LINear** **ARTI** | **ALLARTI**
<state>:	**UND** | **WAIT** | **REL**
<state>: TRACE32 default	**RUN** | **RDY** | **ACT** | **SUSP** | **INTR** | **LIFE** | **PER**
<state>: TRACE32 ARTI	**IPT** | **CET** | **GET** | **RT** | **DT** | **PER** | **ST** | **NST** | **PRE**
<state_item>:	**all** | **Total** | **MIN** | **MAX** | **AVeRage** | **Count** | **Ratio** | **BAR**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* | **Filter** *<item>* | **Sort** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>* **Track** **ARTIAP**

The time tasks spent in different states is measured. Before using this function the task state transitions must be sampled by the trace. This feature is highly dependent on the used RTOS kernel, and needs the **TASK** to be configured. Please see kernel specific **"OS Awareness Manuals"** manuals for more information.

<format>	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.







ratio%	Percentage of CPU usage consumed by the task

List items

TIme[**.**<*state*>]	The total time the task was in this state.
MAX[**.**<*state*>]	The maximum time the task was in this state.
AVeRage[**.**<*state*>]	The average time the task was in this state. **NOTE:** This value can be wrong if intermediate states exist.
Count[**.**<*state*>]	The number of times a state was entered. **NOTE:** This value can be wrong if intermediate states exist.
Ratio	The ratio of CPU runtime consumed by this task.
BAR.log, **BAR.LIN**	Graphical display of ratio column.
ARTI	Shows ARTI timing metrics.
ALLARTI	Shows all ARTI timing metrics.

State

UND	**UND**efined	Shows the time spent by the task in an unknown state.
WAIT	**WAIT**ing	Shows the waiting time spent by the task.
REL	**REL**eased	Shows the time spent by the task in released state.

TRACE32 default states

Initially, not all items with their derived states are displayed, however it is possible to extend the window columns by opening the "config" menu and moving the entry from available to selected.

ACT	**ACT**ivated	Shows the time spent by the task from activation to start.
INTR	**INT**e**R**rupted	Shows the time elapsed from an interruption until the task is resumed.
LIFE	alive	Shows the time spent by the task from activation to termination.
PER	**PER**iod	Shows the time spent by the task from start to start.
RDY	**R**ea**Dy**	Shows the time spent by the task remaining in the ready state, from which it was pre-empted by one or more higher priority tasks.

RUN	**RUN**ning	Shows the execution time spent by the task without taking account of preemption or waiting periods.
SUSP	**SUSP**ended	Shows the time spent by the task from termination to activation.

TRACE32 ARTI states

AUTOSAR defines its own set of timing parameters, therefore we provide separate states for ARTI to align it. These states will be displayed using either the **/ARTI** or **/ALLARTI** options.

CET	**C**ore **E**xecution **T**ime	Same as **RUN**.
DT	**D**elay **T**ime	Same as **PER**.
GET	**G**ross **E**xecution **T**ime	Shows the execution time spent by the task, including all preemptions and waiting time. It also reflects the **LIFE** time minus the **ACT** time.
IPT	**I**nitial **P**endig **T**ime	Same as **ACT**.
NST	**N**et **S**lack **T**ime	Shows the “potential additional” run-time of the task: the **ST** minus all CET blocks of any task or ISRs with higher priority during the **ST**.
PER	**PER**iod	Shows the time spent by the task from activation to activation (period not as measured but as configured).
PRE	**PRE**empted	Same as **RDY**.
RT	**R**esponse **T**ime	Same as **LIFE**.
ST	**S**lack **T**ime	Same as **SUSP**.

See also **“Trace.STATistic.TASKState”** in Application Note Profiling on AUTOSAR CP with ARTI, page 26 (app_autosar_cp_arti.pdf).

See also

■ <trace>.STATistic

▲ ’Release Information’ in ’Legacy Release History’
▲ ’Task Runtime Analysis’ in ’Application Note Profiling on AUTOSAR CP with ARTI’

Format:	*<trace>*.**STATistic.TASKStateDURation** [*<state>* …] [**/***<option>*]
<state>:	**UND** | **RUN** | **RDY** | **WAIT** | **REL** | **ACT** | **SUSP** | **INTR**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>* **Number** *<record>* **FLAT** | **LOG** | **LINear**

Analyzes the time tasks spent in certain states

<option>	Refer to **Options** under **<trace>.STATistic**.

Possible *<state>* are: **UND**efined, **RUN**ning, **R**ea**DY**, **WAIT**ing, **SUSP**ended, **REL**eased, **ACT**ivated, **INT**e**R**rupted.

See also

- <trace>.STATistic
- ▲ 'Task Runtime Analysis' in 'Application Note Profiling on AUTOSAR CP with ARTI'

Format:	*<trace>*.**STATistic.TASKTREE** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **TREE** | **LEVEL** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** | **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **INTRROOT** | **INTRTASK** **InterVal** *<time* | *event>* | **Address** *<address* | *range>* | **Filter** *<item>* **ACCUMULATE** | **INCremental** | **FULL** | **CLOCKS** **NoMerge** | **Sort** *<item>* | **THreshold** *<float>* | **Track**

The results of this command shows a graphical tree of the function nesting.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

■ <trace>.STATistic.TREE   ■ <trace>.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>***.STATistic.TASKVSINTERRUPT** [**%***<format>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* **Filter** *<item>* **Address** *<address>* | *<range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Displays a runtime statistic of tasks that were interrupted by interrupt service routines.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

- <trace>.STATistic
- CTS.STATistic.TASKVSINTERRUPT
- BMC.STATistic
- MIPS.STATistic

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STATistic.TASKVSINTR** [**%**<*format*>] [<*items*> ...] [**/**<*option*>]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **InterVal** *<time>* | *<event>* **Filter** *<item>* **Address** *<address>* | *<range>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Displays an ORTI based ISR2 runtime statistic against task runtimes. This feature can only be used if the ISR2 can be traced based on the information provided by the ORTI file.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

See also

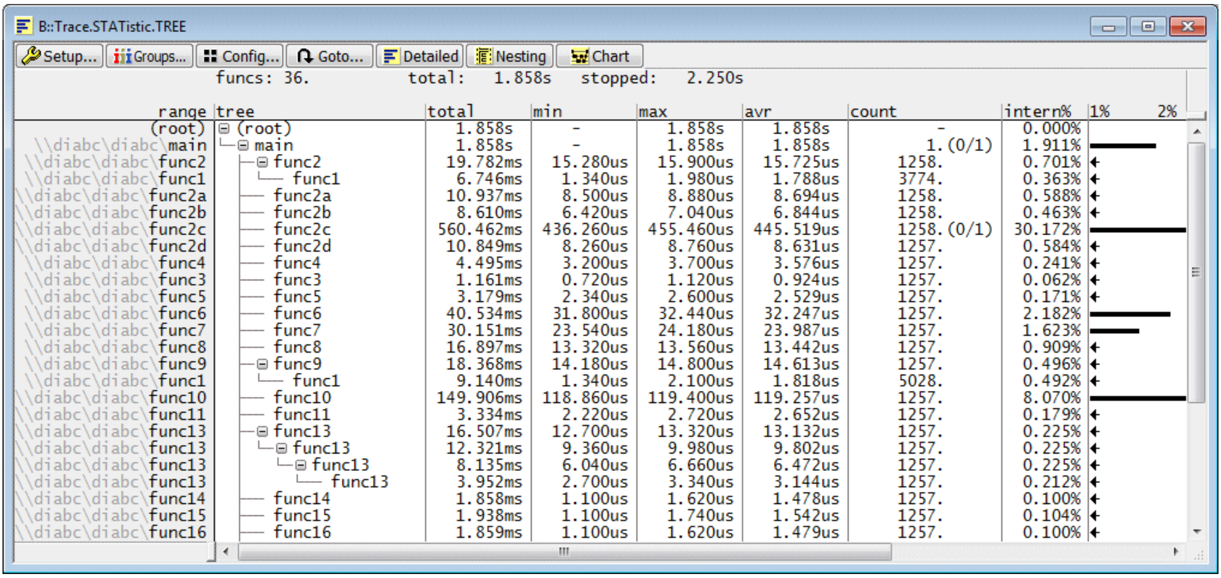
■ <trace>.STATistic

▲ 'Trace Features' in 'OS Awareness Manual OSEK/ORTI'

Format:	*<trace>*.**STATistic.TREE** [**%***<format>*] [{*<list_items>*}] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **TREE** | **LEVEL** | **GROUP** | **TASK** **Total** | **TotalRatio** | **TotalBAR** | **Count** | **MIN** | **MAX** | **AVeRage** **Internal** | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio** | **InternalBAR** **External** | **EAVeRage** | **EMAX** | **ExternalINTR** | **ExternalINTRMAX** **INTRCount** | **ExternalTASK** | **ExternalTASKMAX** | **TASKCount**
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **IncludeOwn** | **IncludeTASK** | **IncludeINTR** **INTRROOT** | **INTRTASK** **Address** *<address>* | *<range>* **Filter** *<item>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **THreshold** *<float>* **Track**

The results of this command shows a graphical tree of the function nesting.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.



B::Trace.STATistic.TREE

Setup... | Groups... | Config... | Goto... | Detailed | Nesting | Chart

funcs: 36.   total: 1.858s   stopped: 2.250s

range	tree	total	min	max	avr	count	intern%
(root)	(root)	1.858s	-	1.858s	1.858s	-	0.000%
\\diabc\diabc\main	main	1.858s	-	1.858s	1.858s	1.(0/1)	1.911%
\\diabc\diabc\func2	func2	19.782ms	15.280us	15.900us	15.725us	1258.	0.701%
\\diabc\diabc\func1	func1	6.746ms	1.340us	1.980us	1.788us	3774.	0.363%
\\diabc\diabc\func2a	func2a	10.937ms	8.500us	8.880us	8.694us	1258.	0.588%
\\diabc\diabc\func2b	func2b	8.610ms	6.420us	7.040us	6.844us	1258.	0.463%
\\diabc\diabc\func2c	func2c	560.462ms	436.260us	455.460us	445.519us	1258.(0/1)	30.172%
\\diabc\diabc\func2d	func2d	10.849ms	8.260us	8.760us	8.631us	1257.	0.584%
\\diabc\diabc\func4	func4	4.495ms	3.200us	3.700us	3.576us	1257.	0.241%
\\diabc\diabc\func3	func3	1.161ms	0.720us	1.120us	0.924us	1257.	0.062%
\\diabc\diabc\func5	func5	3.179ms	2.340us	2.600us	2.529us	1257.	0.171%
\\diabc\diabc\func6	func6	40.534ms	31.800us	32.440us	32.247us	1257.	2.182%
\\diabc\diabc\func7	func7	30.151ms	23.540us	24.180us	23.987us	1257.	1.623%
\\diabc\diabc\func8	func8	16.897ms	13.320us	13.560us	13.442us	1257.	0.909%
\\diabc\diabc\func9	func9	18.368ms	14.180us	14.800us	14.613us	1257.	0.496%
\\diabc\diabc\func1	func1	9.140ms	1.340us	2.100us	1.818us	5028.	0.492%
\\diabc\diabc\func10	func10	149.906ms	118.860us	119.400us	119.257us	1257.	8.070%
\\diabc\diabc\func11	func11	3.334ms	2.220us	2.720us	2.652us	1257.	0.179%
\\diabc\diabc\func13	func13	16.507ms	12.700us	13.320us	13.132us	1257.	0.225%
\\diabc\diabc\func13	func13	12.321ms	9.360us	9.980us	9.802us	1257.	0.225%
\\diabc\diabc\func13	func13	8.135ms	6.040us	6.660us	6.472us	1257.	0.225%
\\diabc\diabc\func13	func13	3.952ms	2.700us	3.340us	3.144us	1257.	0.212%
\\diabc\diabc\func14	func14	1.858ms	1.100us	1.620us	1.478us	1257.	0.100%
\\diabc\diabc\func15	func15	1.938ms	1.100us	1.740us	1.542us	1257.	0.104%
\\diabc\diabc\func16	func16	1.859ms	1.100us	1.620us	1.479us	1257.	0.100%

See also

- <trace>.STATistic
- BMC.STATistic.TREE
- CTS.STATistic.TREE
- <trace>.STATistic.TASKTREE
- CTS.STATistic.ChildTREE

▲ 'Release Information' in 'Legacy Release History'

<trace>.STATistic.Use — Use records

Format:	*<trace>***.STATistic.Use** [*<trace_area>*] [**/***<options>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]
<option>:	**FILE**

The specified record(s) are used for performance and nesting analysis. This command can be used, when some records should be used, which are ignored due to the **<trace>.STATistic.Ignore**.

<option>	Refer to **<trace>.STATistic** for a description of the **<trace>.STATistic** options.

See also

- <trace>.STATistic

Format:	*<trace>*.**STATistic.Var** [**%***<format>*] [{*<list_items>*}] [**/***<option>*]
<format>:	**DEFault** | **LEN** | **TimeAuto** | **TimeFixed**
<list_item>:	**DEFault** | **ALL** | **NAME** | **GROUP** | **CORE** | **BAR[.log** | **.LINear]** **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** **FlowTrace** | **BusTrace** **CORE** *<n>* | **SplitCORE** (default) | **MergeCORE** | **JoinCORE** **TASK** *<task>* | **SplitTASK** | **MergeTASK** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountALL** **Address** *<address>* | *<range>* **Filter** *<item>* **ACCUMULATE** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

The command provides a graphical chart of variable accesses.

<format>, *<list_item>*	Refer to **Parameters** under **<trace>.STATistic**.
<option>	Refer to **Options** under **<trace>.STATistic**.

Example:

```
; Display a statistic of all variable accesses:
Trace.STATistic.Var /Filter sYmbol mstatic1 /Filter CYcle Write

; Display a statistic of write accesses to the mstatic1 variable
Trace.STATistic.Var /Filter sYmbol mstatic1 /Filter CYcle Write
```

See also

- <trace>.STATistic
- 'Release Information' in 'Legacy Release History'

<trace>.STREAMCompression Select compression mode for streaming

Format:	*<trace>*.**STREAMCompression OFF** | **LOW** | **MID** | **HIGH**

LOW (default)	• Trace information is streamed compressed to the host computer. • Trace information is saved to file as received.
MID	• Trace information is streamed compressed to the host computer. • Trace information is zipped before it is saved to file.
HIGH	• Trace information is streamed compressed to the host computer. • Trace information is zipped very compactly before it is saved to file.
OFF (for diagnostic purposes only)	• Trace information is streamed un-compressed to the host computer. • Trace information is saved un-compressed to file.

Example:

```
Trace.STREAMCompression LOW

Trace.Mode STREAM
```

See also

- <trace>.STREAMFileLimit
- <trace>.STREAMSAVE
- <trace>.Mode
- IProbe.state

Format:	*<trace>*.**STREAMFILE** *<file>*

Set the path and file name for the temporary streaming file e.g. a high-capacity or high-speed drive dedicated for this use case.

TRACE32 automatically creates a streaming file which is placed into the TRACE32 temp directory (**OS.PresentTemporaryDirectory()**) by default and is named *<trace32_instance_id>***streama.t32** (**OS.ID()**).

Example:

```
Trace.STREAMFILE "d:\temp\mystream.t32"    ; specify the location for
                                           ; your streaming file

Trace.Mode STREAM                          ; select the trace mode STREAM
```

NOTE:	The file limit of the streaming file must be set before starting streaming. Later changes are not taken into account.

See also

- <trace>.STREAMFileLimit
- <trace>.Mode
- IProbe.state
- Onchip.Mode

Format:	*<trace>*.**STREAMFileLimit** *<+/- limit_in _bytes>*

Sets the maximum size allowed for a streaming file. If the maximum size is exceeded, the trace recording is stopped and the warning "Streaming trace terminated" is displayed.

The limit value is given in bytes and can have a positive or negative sign:

- Positive value: The maximum size of the streaming file in bytes
- Negative value: Specifies the amount of space to leave on the disk before stopping streaming. The maximum file size is calculated based on the amount of available disk space at the time of starting streaming.

The default setting is -1.000.000.000 i.e. stops trace recording when less than a GB of space is left on the storage medium.

NOTE:	The maximum size of the streaming file must be set before starting streaming. Later changes are not taken into account.

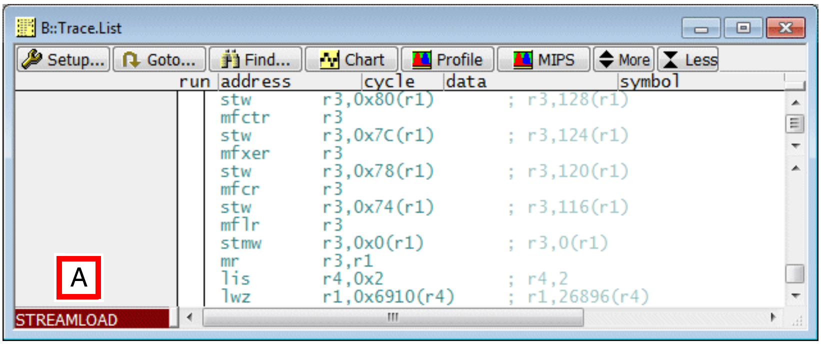
See also

- <trace>.STREAMFILE
- <trace>.Mode
- Onchip.Mode
- <trace>.STREAMCompression
- IProbe.state

Format:	*<trace>*.**STREAMLOAD** *<file>*

Load a streaming file that was saved with the command **Trace.STREAMSAVE** to TRACE32.

In order to display trace information the target state at the recording time has to be reconstructed within TRACE32. This can be complex, especially if target software with an operating system that uses dynamic memory management to handle processes/tasks (e.g. Linux) is used.



A After loading the trace data from the streaming file, the **STREAMLOAD** label in the bottom-left corner indicates that the contents of a loaded streaming file are being displayed.

Example 1: Reconstruction of the target state at the recording time for a bare metal Cortex-R4 application.

```
; specify the target CPU
SYSTEM.CPU TMS570PSFC61
SYStem.Option.BigEndian ON
SYStem.Up
; specify ETM settings that were used at the time of recording
ETM.PortSize 16.
ETM.PortMode Bypass
ETM.DataTrace OFF
ETM.ContextID OFF
ETM.ON
; load source code and debug information
Data.LOAD.Elf demo.axf
; load saved streaming file
Trace.STREAMLOAD C:\T32_ARM\r4_max.sad
Trace.List
```

Example 2: Reconstruction of the target state at the recording time for NEXUS Power Architecture:

```
; specify the target CPU
SYStem.CPU MPC5646C
; specify the NEXUS settings that were used at the time of recording
NEXUS.PortSize MDO12
NEXUS.PortMode 1/2
NEXUS.BTM ON
NEXUS.HTM ON
NEXUS.PTCM BL_HTM ON
NEXUS.ON
SYStem.Up
; mapping logical to physical address is 1:1
; load source code and debug information
Data.LOAD.Elf im02_bf1x.elf
; load the OS Awareness
TASK.ORTI im02_bf1x.ort
; load saved streaming file
Trace.STREAMLOAD my_stream
Trace.List
```

See also

■ <trace>.Mode ■ <trace>.STREAMSAVE

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**STREAMSAVE** *<file>*

Save the streaming file to a permanent file. Use **Trace.STREAMLOAD** to load this file for analysis.

The contents of the streaming file are in a proprietary format and not intended for use in external applications.

<file>	The default extension for the streaming file is ***.sad**.

See also

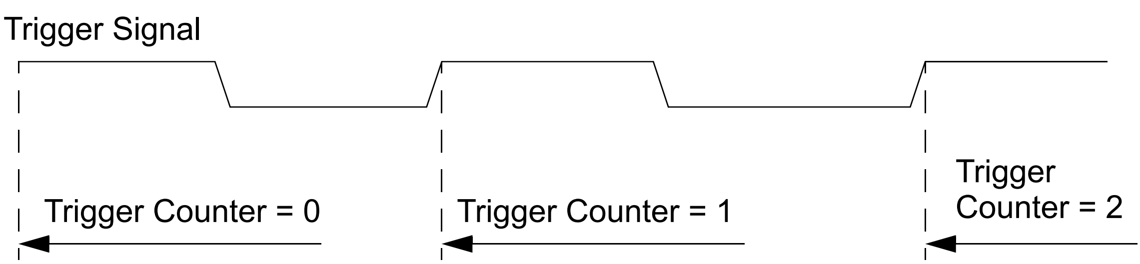
- <trace>.STREAMCompression
- <trace>.STREAMLOAD
- <trace>.Mode
- <trace>.SAVE

▲ 'Release Information' in 'Legacy Release History'

<trace>.TCount Set trigger counter

Format:	**Integrator.TCount** [*<value>*]
<value>:	**0. … 16777215.**

Sets the number of trigger events that will be ignored by the trace or logic analyzer, before a trigger event ends the recording (state: break). A counter value zero means that the recording stops immediately after the first trigger. A value of 1 halts the recording at the second trigger event, and so on.



See also

- IProbe.state

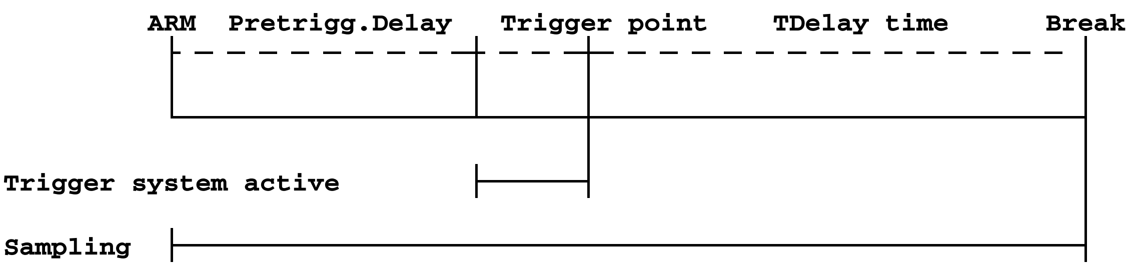
Format:	*<trace>***.TDelay** *<time>* | *<cycles>* | *<percent>***%** **ETM.TDelay** *<value>* (deprecated)
<time>:	**0 … 200.s**
<percent>:	**0 … 1000%**
<cycles>:	**0 … 4000000000.**

Selects the delay time between trigger point and break (end of recording). Use this command in order to record events that occurred after the trigger point.

The trigger delay may also be defined in percent of the trace buffer size. The delay can be larger (up to 10x) than the total trace buffer size. Logic analyzers also support setting a delay time.

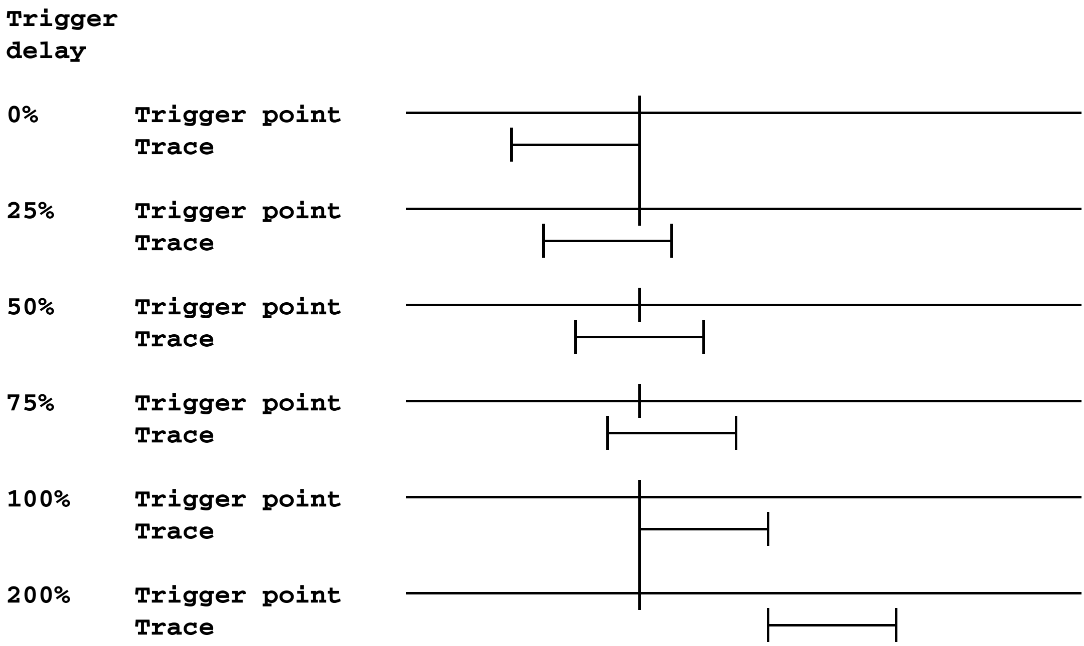
```
Analyzer.TDelay 40%             ; trigger delay is 40% of trace depth.
```

Selects the delay time between trigger point and break of the port analyzer. The time can be larger than the time for a full sample of the analyzer. The trigger delay time may be defined in percent relating to the total trace time.



With a mouse click to the corresponding area in the port analyzer state window this command can be executed too.

```
Port.TDelay 10.ms      ; the trigger delay is 10 ms

Port.TDelay 50%        ; the trigger point is in the mid of the trace
                       ; memory

Port.TDelay 99%        ; the trigger point is at the beginning to the trace
                       ; memory

Port.TDelay 200.       ; the trigger point 200 record before end of trace
```



See also

■ IProbe.state

Format:	*<trace>*.**TERMination ON** | **OFF**

By default the trace line termination of the preprocessor is used during a trace capture. Undefinable **FLOWERRORs** may occur if the output drivers of the CPU are not strong enough. In this case it is recommended to switch the trace line termination OFF.

Format:	*<trace>*.**TestFocus** [*<address_range>*] [**/***<option>*]
<option>:	**Accumulate** **Config** **KEEP** **ALTERNATE** **NoTraceControl**

The command **Trace.TestFocus** tests the recording at a high-speed trace port.

The command **Trace.TestFocus** can be used if:

- **The program execution is stopped.**

  To test the trace port, the test pattern generator of the trace port is used if available. Otherwise, a test program is loaded and started by TRACE32.

- **The program execution and the trace recording is running.**

  Testing the trace port while the application program is running might be helpful to detect trace port problems caused by the application program.

  The trace data from the application program are used to test the trace port. Here a reduced test scenario is processed that checks the correctness of the program flow recording and for short-circuits between the trace port lines. This test requires that the program code is loaded to the virtual memory.

```
Data.LOAD.Elf arm.elf /PlusVM         ; Load the application code
                                      ; to the target memories and
                                      ; to the virtual memory of
                                      ; TRACE32
```

- **The program execution is running and the trace recording is stopped.**

  To test the trace port, the test pattern generator of the trace port is used if available. Otherwise, the trace data from the application program are used.

If a test program is used, TRACE32 attempts to load the test program to the memory addressed by the PC or the stack pointer. It is also possible to define an *<address_range>* for the test program.

```
Trace.TestFocus                       ; start trace port test

Trace.AutoFocus 0x24000000++0xfff     ; start the test and load
                                      ; test program to address
                                      ; 0x24000000
```

If TRACE32 is unable to load the test program the following error message is displayed:
“Don‘t know where to execute the test code”.

By default, the original RAM content is restored after the trace port test and the trace recording is deleted.

Accumulate	If the application program varies the CPU clock frequency, this affects also the trace port. In such a case it is recommended to overlay the test results for all relevant CPU clock frequencies by using the option **/Accumulate**.
Config	Allows to define a RAM address range for the download of the test program.
KEEP	After a trace port test the trace is cleared and any loaded test program is removed from the target RAM. With the option **/KEEP**, the test trace is not cleared and can be viewed with the **Trace.List** command. If a test program was loaded by TRACE32, it also remains in the target RAM.
ALTERNATE	If the trace port provides a test pattern generator, it is always used for the test. The option **/ALTERNATE** forces TRACE32 to use its own test program.
NoTraceControl	Informs the TRACE32 software that the trace control signal is not available on the trace connector.

```
; advise the command Trace.TestFocus to download the test program
; always to the address range 0x24000000++0xfff
Trace.TestFocus 0x24000000++0xfff /Config
```

The result of the command **Trace.TestFocus** can be processed in a PRACTICE script as follows:

```
Trace.TestFocus

IF FOUND()
   PRINT %ERROR "Trace port test failed"

ELSE
   PRINT "Trace port recording ok"
```

The **Trace.TestFocus** command calls the data eye finder for the current hardware configuration of a preprocessor with AUTOFOCUS technology and verifies the correctness of traced test data. In contrast to **Trace.AutoFocus**, the preprocessor configuration remains unchanged.

A complete trace port test executes the following steps:

1. The data eye finder is called. The source for the trace data for the test are the trace port's pattern generator, a test program or the application program.

2. When the eye finder is done, the test is started once again to verify the correctness of the trace recording.

3. The data eyes resulting from the **<trace>.TestFocus** command can be viewed in the **<trace>.ShowFocus** window.

See also

- <trace>.TestFocusClockEye
- <trace>.ShowFocusClockEye
- <trace>.TestFocusEye
- AUTOFOCUS.OK()
- <trace>.ShowFocus
- <trace>.ShowFocusEye
- AUTOFOCUS.FREQUENCY()

▲ 'Release Information' in 'Legacy Release History'

<trace>.TestFocusClockEye — Scan clock eye

Format:	*<trace>*.**TestFocusClockEye** [*<address_range>*] [**/***<option>*]
<option>:	**Accumulate** | **Config** | **KEEP** | **ALTERNATE** | **Utilisation** | **NoTraceControl**

Scans the clock eye. To view the result, use the command **Trace.ShowFocusClockEye**.

NOTE:	The NEXUS AutoFocus adapter does not support this feature.

<option>	For a description of the options, see **Trace.TestFocus**.

See also

- <trace>.TestFocus
- <trace>.ShowFocusClockEye
- <trace>.TestFocusEye
- <trace>.ShowFocus
- <trace>.ShowFocusEye

<trace>.TestFocusEye — Check signal integrity

Format:	*<trace>*.**TestFocusEye** [*<address_range>*] [**/***<option>*]
<option>:	**Accumulate** **Config** **KEEP** **ALTERNATE** **NoTraceControl**

Scans the data eye to determine the integrity of the electrical trace signals.

The command **Trace.TestFocusEye** starts an eye finder to test the quality of the trace signals, if a preprocessor with AUTOFOCUS technology is used. The test result can be displayed with the command **Trace.ShowFocusEye**. If the result shows that an individual trace signal has a significantly smaller data eye than other signals, the hardware layout should be checked to see if this signal shows any unusual features.

The test procedure and the options used by the command **Trace.TestFocusEye** are similar to the command **Trace.TestFocus**.

<option>	For a description of the options, see **Trace.TestFocus**.

The command **Trace.TestFocusEye** can also be used with PowerTrace Serial. For this tool no additional parameters (e.g. *<address_range>*) or options are available.

See also

- <trace>.TestFocus
- <trace>.ShowFocus
- <trace>.ShowFocusEye
- <trace>.TestFocusClockEye
- <trace>.ShowFocusClockEye

▲ 'Release Information' in 'Legacy Release History'

<trace>.TestUtilization — Tests trace port utilization

Format:	**<trace>.TestUtilization** [**/***<option>*]
<option>:	**KEEP** | **CONTENTS**

Tests trace port utilization. The result is printed to the AREA window. This command is supported for the trace methods **Analyzer**, **CAnalyzer** and **Onchip**.

This command is only supported for ETM trace.

Format 1:	*<trace>*.**THreshold VCC** | **CLOCK** | *<level>* (Preprocessor for ARM-ETM with AUTOFOCUS only)
Format 2:	*<trace>*.**THreshold** *<clock> <data>*

The command **Trace.THreshold** can be used to optimize the threshold level for the trace lines sampled via a TRACE32-Preprocessor (e.g. ARM-ETM, OCDS Level 2, AUD …). The optimization of the threshold level should result in less errors in the trace recording.

VCC	The preprocessor and the TRACE32 software measure the VCC of the target. 1/2 VCC is then automatically used as the threshold level for the trace lines. The result is also displayed in the THreshold field of the **<trace>.state** window.
CLOCK	The threshold level is changed until the duty cycle of the trace clock reaches a ratio of 1:1. This setting is only recommended if the trace clock has a duty cycle of 1:1. The result is displayed in the **THreshold** field of the **<trace>.state** window.
<level>	The threshold level can be entered directly.

```
Trace.THreshold VCC

Trace.THreshold CLOCK

Trace.THreshold 1.6                  ; the unit is Volt
```

Enhanced parameters for the Preprocessor for ARM-ETM with AUTOFOCUS:

<clock> <data>

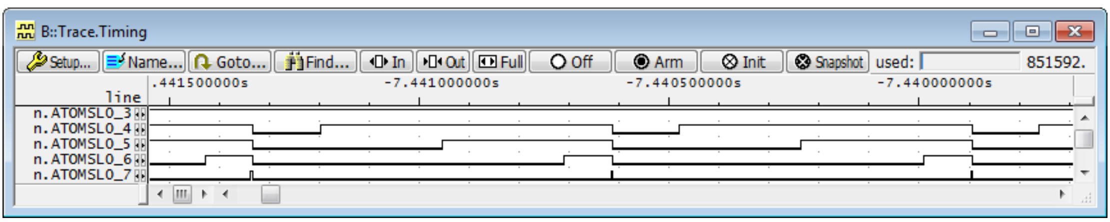
For the Preprocessor for ARM-ETM with AUTOFOCUS different threshold levels can be defined for the clock and the data lines

```
Trace.THreshold 0.86 0.79            ; the unit is Volt
```

Format:	*<trace>*.**Timing** [*<record_range>*] [{*<items>*}] [**/***<options>*]
<option>:	**FILE** **Track** **RecScale** **TimeScale** **TimeZero** **TimeREF**

Displays the trace memory contents like command **<trace>.List**, but in form of a timing display. As a default the external trigger channels are displayed.

FILE	Display trace memory contents loaded with **Trace.FILE**.
Track	The cursor in the **<trace>.Timing** window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed. The zoom factor of the **<trace>.Timing** window is retained, even if the trace content changes.
RecScale	Display trace in fixed record raster. This is the default.
TimeScale	Display trace as true time display, time relative to the trigger point.
TimeZero	Display trace as true time display, time relative to zero.
TimeREF	Display trace as true time display, time relative to the reference point.



Buttons	
Zoom In T	Zooms in Trace by a factor of 2.
Zoom Out T	Zooms out Trace by a factor of 2.
Zoom Full T	Display the complete trace buffer in the window.
Goto …	Open an **<trace>.GOTO** dialog box.
Find …	Open an **<trace>.Find** dialog box.
Set Ref	Set an analyzer reference point to the current record.
Set Zero	Set the global time reference to the current record.
View	Display all information about the current record (**<trace>.View**).
List	Open an **<trace>.List** window.

Examples:

```
; Open Port Analyzer timing window in standard display format
E::Port.Timing

; Open Port Analyzer timing window and display last file loaded with
; " Port.FILE <file>"
E::Port.Timing /FILE

; Open Port Analyzer timing window starting at record -100. in standard
; display format
E::Port.Timing -100. DEFault
```

See also

- ■ <trace>.List
- ■ <trace>.View
- ■ RunTime
- ❏ Analyzer.RECORD.ADDRESS()
- ❏ Analyzer.RECORD.OFFSET()
- ❏ Analyzer.REF()
- ■ <trace>.REF
- ■ IProbe.state
- ■ RunTime.state
- ❏ Analyzer.RECORD.DATA()
- ❏ Analyzer.RECORDS()

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**TMode** [**High** | **Low** | **Rising** | **Falling**]

Selects the trigger condition, edge or level trigger and the corresponding line polarity. The edge trigger is asynchronous and needs a minimum pulse width of 20 ns.

Example:

```
E::Port.TMode Rising
E::Port.TSELect Port
E::Port.SELect Port.00         ; trigger on the rising edge of P.00
E::Port.TDelay 100.us          ; sample till 100.us after trigger
```

<trace>.TraceCONNECT Select on-chip peripheral sink

Format:	*<trace>*.**TraceCONNECT** *<component>* *<trace>*.**TraceCONNECT NONE** *<trace>*.**TraceCONNECT AUTO**

Default: AUTO.

Selects the on-chip peripheral used as trace sink on the SoC.

Example: The two ETFs of an ARM CoreSight based SoC are selected as trace sink.

```
;note that the two approaches to select the first ETF are equivalent:
Onchip.TraceCONNECT ETF1        ; selects the ETF1 as onchip-trace sink
  ;or
Trace.METHOD Onchip
Trace.TraceCONNECT  ETF1        ; selects the ETF1 as onchip-trace sink


;note that the two approaches to select the second ETF are equivalent:
Onchip.TraceCONNECT ETF2        ; selects the ETF2 as onchip-trace sink
  ;or
Trace.METHOD Onchip
Trace.TraceCONNECT ETF2         ; selects the ETF2 as onchip-trace sink
```

<trace>.TRACK Set tracking record

Format:	*<trace>*.**TRACK** *<time>* | *<record>* | "*<trace_bookmark>*"

Sets the tracking record to the specified trace bookmark, time, or record number. The blue cursor moves to the specified destination in all **Trace.*** windows opened *with* the **/Track** option. All other **Trace.*** windows opened *without* the **/Track** option do not respond to the **<trace>.TRACK** command.

Example:

```
;set the tracking record to the record -12000.
Trace.TRACK  -12000.

;without /Track: this window does not respond to the Trace.TRACK command
Trace.List   %TimeFixed TIme.ZERO   DEFault

;with /Track: this window responds to the Trace.TRACK command, i.e. the
;blue cursor selects the tracking record -12000.
Trace.List   %TimeFixed TIme.ZERO   DEFault   /Track

;display only selected trace information about the record currently
;selected in the Trace.List ... /Track window
Trace.View   %TimeFixed TIme.ZERO   DEFault   /Track
```

See also

- ■ RunTime
- ■ RunTime.state
- ■ <trace>.GOTO
- ■ BookMark
- ■ IProbe.state
- ❏ TRACK.ADDRESS()
- ❏ TRACK.RECORD()
- ❏ TRACK.TIME()

<trace>.TRIGGER Trigger the trace

Format:	*<trace>*.**TRIGGER**

Forces a manual trigger.

See also

- ■ Trace

Format:	*<trace>*.**TSELect** [*<source>*]
<source>:	**BusA** [**ON** | **OFF**] **EXT** [**ON** | **OFF**]

Selects the trigger source for the port analyzer.

BusA	Trigger lines on the trigger bus. This lines may be controlled by the state analyzer, by the timing analyzer or the pattern generator.
EXT ON | **OFF**	The external trigger input on the ETM connector is turned off by default. **Analyzer.TSELect EXT** can enable or disable the trigger source.

See also

■ Trace

Format: *<trace>*.**View** [*<record>*] [*<channels>*] [**/***<options>*]

<option>: **FILE**
Track

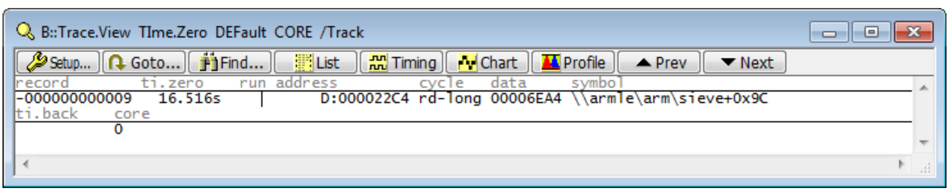
Displays a single record in a more detailed format. The syntax of the channel definitions is the same as for the **<trace>.List** command. Without arguments all channels are displayed.

Example 1:

```
;display all information about a specific record, here record -12000.
Trace.View  -12000.

;display only selected trace information about a specific record
Trace.View  -12000. TIme.ZERO  DEFault  CORE
```

Example 2:

```
;open a Trace.List window with all records. Display the ti.zero column
;as the first column, followed by the DEFault columns
Trace.List  TIme.ZERO  DEFault        /Track

;display only selected trace information about the record currently
;selected in the Trace.List window
Trace.View  TIme.Zero  DEFault  CORE  /Track
```



See also

- <trace>.List
- <trace>.Timing
- RunTime
- Analyzer.RECORD.ADDRESS()
- Analyzer.RECORD.OFFSET()
- Analyzer.REF()
- <trace>.REF
- IProbe.state
- RunTime.state
- Analyzer.RECORD.DATA()
- Analyzer.RECORDS()
- Analyzer.SIZE()

▲ 'Release Information' in 'Legacy Release History'

Format:	*<trace>*.**ZERO** [*<time>* | *<record>* | "*<trace_bookmark>*"]

Use this command to align the zero time point for trace and timing analyzer sources with time bases of different origin.

<time>	Moves the ZERO point by specified time.
<record>	Sets the ZERO point to the time index of the specified record number.
<bookmark>	Sets the ZERO point to the time index of the specified bookmark location. You can create trace bookmarks with the **<trace>.BookMark** command.
no parameter	Reset zero time point back to initial location.

The table below shows the different sources for time information. As the different sources are not related, they all have an individual zero time point.

Timestamp	**Trace data source**	**Original zero time point**
Timestamps generated by TRACE32 hardware	Analyzer (no processor generated timestamps) PowerProbe, Integrator, IProbe, etc.	Permanently set to beginning of first debug session or trace recording after starting up TRACE32 PowerView. All trace data sources using TRACE32 hardware generated timestamps have a common zero time point.
Timestamps generated by target processor	Onchip Trace Analyzer Trace (with processor generated timestamps enabled)	Depends on CPU architecture and trace protocol. Starting a new trace recording usually moves the zero time point to a new location.
Timestamps loaded from files.	Trace.LOAD *<file>* /FILE	Same as in original recording

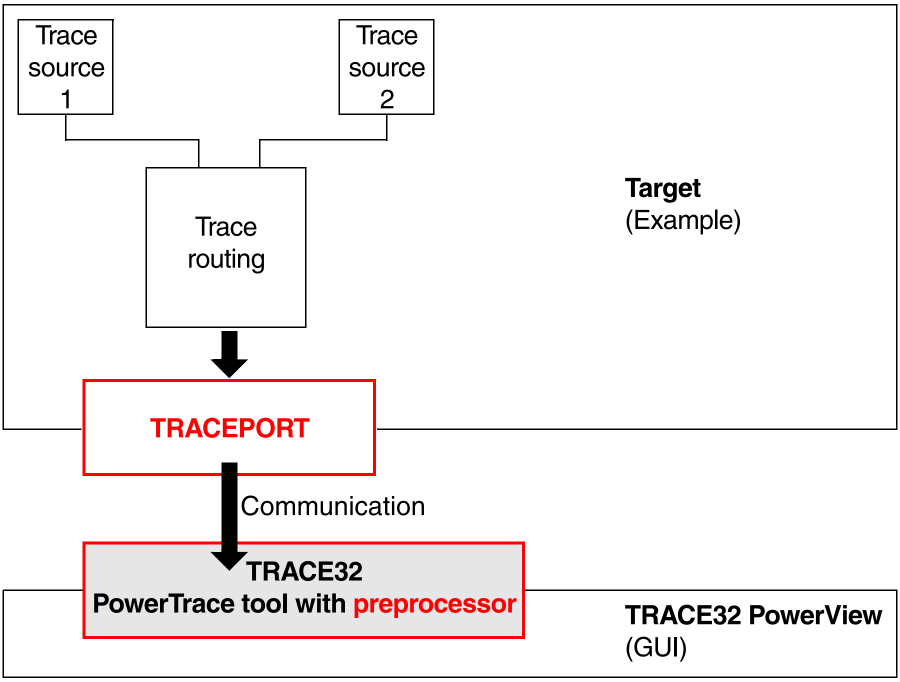
Due to the different zero time points of the various data sources, it is required to align the zero time points, before trace or timing recordings can be observed in a correlated manner. This is usually achieved by locating a common event in the different sources and selecting this event as common zero time point.

See also

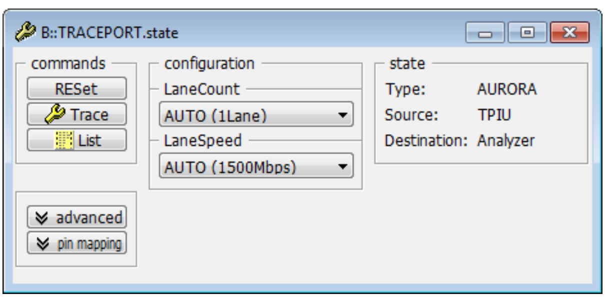
- IProbe.state
- RunTime
- RunTime.state

TRACEPORT Configure trace hardware

Using the **TRACEPORT** command group, you can configure the communication between the target trace port and the TRACE32 PowerTrace tool. Logically the **TRACEPORT** command group is located between the physical pins of the target platform and the TRACE32 trace input stage (preprocessor), see illustration below.



For trace port configuration, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **TRACEPORT.state** window.



See also

- TRACEPORT.EndsKiP
- TRACEPORT.LaneCount
- TRACEPORT.LanePolarity
- TRACEPORT.LaneSpeed
- TRACEPORT.MsgBItEndian
- TRACEPORT.MsgBYteEndian
- TRACEPORT.MsgLOngEndian
- TRACEPORT.MsgWOrdEndian
- TRACEPORT.OSCFrequency
- TRACEPORT.PinReMap
- TRACEPORT.RefCLocK
- TRACEPORT.RESet

■ TRACEPORT.StartsKiP
■ SYStem.CONFIG.TRACEPORT
■ TRACEPORT.state

▲ 'TRACEPORT Function' in 'General Function Reference'
▲ 'Trace Port Utilization' in 'PowerTrace Serial User's Guide'

TRACEPORT.EndsKiP   Define number of bytes skipped at the end of frame

For serial trace ports (AURORA) only

Format:	**TRACEPORT.EndsKiP** [*<option>*]
<option>:	**AUTO** | **0** | **2** | **8**

Allows to cut off data bytes at the end of each data packet or data frame. Depending on the target configuration, the last bytes of a frame contain CRC information, which is not used by TRACE32. With the command **TRACEPORT.EndsKiP** it is possible to remove the unused bytes.

AUTO	TRACE32 defines the number of bytes to be cut.
0	Don't cut any bytes.
2	Cut 2 bytes at the end of each frame.
8	Cut 8 bytes at the end of each frame.

See also

■ TRACEPORT
■ TRACEPORT.state

TRACEPORT.LaneCount — Select port size of the trace port

For serial trace ports (AURORA/PCIe) only

Format:	**TRACEPORT.LaneCount** *<size>*
<size>:	**AUTO** | **1Lane** | **2Lane** | **3Lane** | **4Lane** | **5Lane** | **6Lane** | **7Lane** | **8Lane**

Specifies the number of used lanes for the trace port. The number must match the target configuration, else the trace link between the target and the TRACE32 hardware cannot be established.

AUTO	TRACE32 defines the lane count.
1Lane, **2Lane**, **3Lane**, **4Lane**, **5Lane**, **6Lane**, **7Lane**, **8Lane**	Number of used lanes. In case of PCIe the lane setup will be done automatically.

See also

■ TRACEPORT ■ TRACEPORT.state ❏ TRACEPORT.LaneCount()

TRACEPORT.LanePolarity — Set polarity for each lane of the trace port

For serial trace ports (AURORA) only

Format:	**TRACEPORT.LanePolarity** *<value>*
<value>:	**AUTO** | *<bit mask>*

Allows to change the polarity for each lane separately. This is necessary when the p/n-signals of a lane are crossed due to e.g. layout reasons.

AUTO	TRACE32 defines the value.
0y…	Polarity defined by user bit mask depending on the lane count.

Example:

```
TRACEPORT.LaneCount 4Lane
TRACEPORT.LanePolarity 0y0101  ; polarity changed for Lane 0 and 2

TRACEPORT.LaneCount 2Lane
TRACEPORT.LanePolarity 0y11    ; polarity changed for Lane 0 and 1
```

See also

■ TRACEPORT ■ TRACEPORT.state

TRACEPORT.LaneSpeed Inform debugger about trace port rate

For serial trace ports (AURORA/PCIe) only

Format:	**TRACEPORT.LaneSpeed** *<data_rate>*
<data_rate>: For AURORA only	**AUTO** | **625Mbps** | **750Mbps** | **850Mbps** | **931Mbps** | **1000Mbps** | **1040Mbps** | **1250Mbps** | **1500Mbps** | **1563Mbps** | **1700Mbps** | **1862Mbps** | **2000Mbps** **2079Mbps** | **2500Mbps** | **3000Mbps** | **3125Mbps** | **3400Mbps** | **3724Mbps** | **4000Mbps** | **4158Mbps** | **4250Mbps** | **5000Mbps** | **6000Mbps** | **6250Mbps** | **6800Mbps** | **7448Mbps** | **8000Mbps** | **10000Mbps** | **1075Mbps** | **12000Mbps** | **12500Mbps** | **2150Mbps** | **2340Mbps** | **4300Mpbs**
<data_rate>: For PCIe only	**GEN1** | **GEN2** | **GEN3**

Informs the debugger about the lane *<data_rate>*. The data rate must match the configuration on the target side, else the link between the target and the TRACE32 hardware (Aurora trace channel) cannot be established.

Remember that not all TRACE32 PowerTrace tools support all data rates.
Contact support@lauterbach.com if a lane speed is not supported.

AUTO	TRACE32 defines the value.
625Mbps, …	Data rate in megabits per second.
GEN1, …	Limits the data rate of the PCIe link to 2500Mbps (**GEN1)**, 5000Mbps (**GEN2)** or 8000Mbps (**GEN3).**

Example:

```
TRACEPORT.LaneSpeed 3125Mbps
TRACEPORT.LaneSpeed 3125M          ; M is the short form of Mbps
```

See also

■ TRACEPORT ■ TRACEPORT.state

TRACEPORT.MsgBItEndian Change bit-order within each byte

For serial trace ports (AURORA) only

Format:	**TRACEPORT.MsgBItEndian** [*<option>*]
<option>:	**AUTO** | **LittleEndian** | **BigEndian**

Allows you to change the bit order of the payload data if the bit order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

AUTO	TRACE32 defines the value.
LittleEndian	Bit order is normal ([31-24],[23-16],[15-8],[7-0]).
BigEndian	Bit order is reversed ([24-31],[16-23],[8-15],[0-7]).

See also

■ TRACEPORT ■ TRACEPORT.state

TRACEPORT.MsgBYteEndian — Change byte-order within each word

For serial trace ports (AURORA) only

Format:	**TRACEPORT.MsgBYteEndian** [*<option>*]
<option>:	**AUTO** | **LittleEndian** | **BigEndian**

Allows you to change the byte order of the payload data if the byte order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

AUTO	TRACE32 defines the value.
LittleEndian	Byte order is normal ([31-24],[23-16],[15-8],[7-0]).
BigEndian	Byte order is reversed ([23-16],[31-24],[7-0],[15-8]).

See also

- TRACEPORT
- TRACEPORT.state

TRACEPORT.MsgLOngEndian — Change dword-order within each qword

For serial trace ports (AURORA) only

Format:	**TRACEPORT.MsgLOngEndian** [*<option>*]
<option>:	**AUTO** | **LittleEndian** | **BigEndian**

Allows you to change the byte order of the payload data if the byte order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

AUTO	TRACE32 defines the value.
LittleEndian	Double-word order is normal ([63-32],[31-0]).
BigEndian	Double-word order is reversed ([31-0],[63-32]).

See also

- TRACEPORT
- TRACEPORT.state

TRACEPORT.MsgWOrdEndian — Change word-order within each dword

For serial trace ports (AURORA) only

Format:	**TRACEPORT.MsgWOrdEndian** [*<option>*]
<option>:	**AUTO** | **LittleEndian** | **BigEndian**

Allows you to change the byte order of the payload data if the byte order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

AUTO	TRACE32 defines the value.
LittleEndian	Word order is normal ([31-16],[15-0]).
BigEndian	Word order is reversed ([15-0],[31-16]).

See also

■ TRACEPORT ■ TRACEPORT.state

TRACEPORT.OSCFrequency — Set OSC clock frequency

For serial trace ports (AURORA) only

Format:	**TRACEPORT.OSCFrequency** *<value>*
<value>:	**AUTO** | *<frequency in kHz>*

Allows you to set the OSC clock frequency. To become active it is required to select reference clock source **OSC**. Please refer to **TRACEPORT.RefCLocK**.

See also

■ TRACEPORT ■ TRACEPORT.state

For serial trace ports (AURORA) only

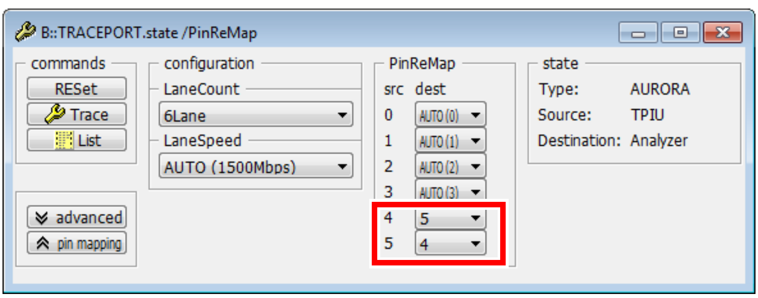
Format:	**TRACEPORT.PinReMap** *<source_lane> <destination_lane>* | *<option>*
<source_ lane>:	**0** | **1** | … | *<n>*
<destination_ lane>:	**AUTO** | **0** | **1** | … | *<n>*
<option>:	**RESET**

Adapts the lane order of the trace port to the lane order of your target. You need the **TRACEPORT.PinReMap** command only in rare cases where the lane orders of trace port and target actually differ from each other.

AUTO	TRACE32 defines the values.
RESET	Sets all values to **AUTO** again.
<source_lane>	Number of the **target lane** which needs to be remapped.
<destination_lane>	Number of the **TRACE32 tool lane** which will get the new *<source_lane>*. Number *<n>* is TRACE32 tool dependent; e.g. for PowerTrace Serial *<n>* can be 5 or 7 depending on the used tool connector.

Example:

```
TRACEPORT.state /PinReMap  ;optionally, open the TRACEPORT.state window
TRACEPORT.LaneCount 6Lane  ;the number of used lanes for the trace port
TRACEPORT.PinReMap 4. 5.   ;map source lane 4. to destination lane 5.
TRACEPORT.PinReMap 5. 4.   ;map source lane 5. to destination lane 4.
```



See also

■ TRACEPORT ■ TRACEPORT.state

TRACEPORT.RefCLocK — Set up reference clock for trace port

For serial trace ports (AURORA) only

Format:	**TRACEPORT.RefCLocK** [*<option>*]
<option>:	**AUTO** | **OFF** | **OSC** | **1/1** | **1/2** | **1/10** | **2/25** | **1/20** | **1/25** | **1/30** | **1/34** | **1/40** | **1/50**

Defines the reference clock frequency the serial trace hardware outputs to the target. The availability of parameters and the default values depend on the architecture:

- PowerPC: not configurable
- TriCore: not configurable
- RH850: not configurable
- ARM: configurable

AUTO (default)	TRACE32 defines the value.
OFF	TRACE32 does not send any reference clock to the target.
OSC	An asynchronous oscillator will be enabled. Its frequency is programmable. Refer to **TRACEPORT.OSCFrequency**.
1/*<x>*	A synchronous clock source will be enabled. Its dividers generate a reference clock as a fraction of the bit clock (lane speed), e.g. 100MHz at 5Gbps with divider 1/50. Once a divider is selected, the reference clock will automatically change with the lane speed.

See also

■ TRACEPORT ■ TRACEPORT.state

TRACEPORT.RESet — Reset trace port configuration

Format:	**TRACEPORT.RESet**

Resets the trace port configuration to its default values (AUTO).

See also

■ TRACEPORT ■ TRACEPORT.state

Format:	**TRACEPORT.StartsKiP** [*<option>*]
<option>:	**AUTO** | **0** | **1**

Allows to cut off leading bytes of each data packet or data frame. Only a few targets requires this due to protocol irregularities.

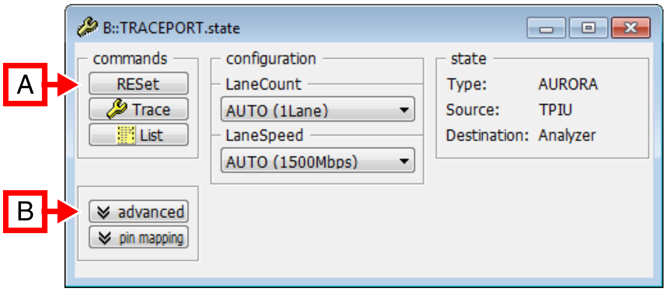
AUTO (default)	TRACE32 defines the value.
0	No data byte will be cut off.
1	The first data byte of each data frame will be cut off.

See also

- TRACEPORT
- TRACEPORT.state

Format:	**TRACEPORT.state** [**/**<*gui_option*>]
<*gui_option*>:	**ADVanced** **PinReMap**

Displays the **TRACEPORT.state** window, where you can configure the communication between the target trace port and the TRACE32 PowerTrace tool.



A For descriptions of the commands in the **TRACEPORT.state** window, please refer to the **TRACEPORT.*** commands in this chapter.
Example: For information about the **RESet** button, see **TRACEPORT.RESet**.

B Click **advanced** and **pin mapping** to display more configuration options in the window.

ADVanced	Extends the list of options in the **configuration** section.
PinReMap	Displays the **PinReMap** section. For an example, see **TRACEPORT.PinReMap**.

See also

- TRACEPORT
- TRACEPORT.LaneCount
- TRACEPORT.LaneSpeed
- TRACEPORT.MsgBYteEndian
- TRACEPORT.MsgWOrdEndian
- TRACEPORT.PinReMap
- TRACEPORT.RESet
- TRACEPORT.EndsKiP
- TRACEPORT.LanePolarity
- TRACEPORT.MsgBItEndian
- TRACEPORT.MsgLOngEndian
- TRACEPORT.OSCFrequency
- TRACEPORT.RefCLocK
- TRACEPORT.StartsKiP

TRANSlation — Debugger address translation

See also

■ TRANSlation.AutoEnable	■ TRANSlation.AutoSCAN	■ TRANSlation.CacheFlush	■ TRANSlation.CLEANUP
■ TRANSlation.COMMON	■ TRANSlation.Create	■ TRANSlation.CreateID	■ TRANSlation.CreateTab
■ TRANSlation.Delete	■ TRANSlation.DeleteID	■ TRANSlation.List	■ TRANSlation.ListID
■ TRANSlation.NoProtect	■ TRANSlation.OFF	■ TRANSlation.ON	■ TRANSlation.PAGER
■ TRANSlation.Protect	■ TRANSlation.RESet	■ TRANSlation.SCANall	■ TRANSlation.ScanID
■ TRANSlation.SHADOW	■ TRANSlation.state	■ TRANSlation.TableWalk	■ TRANSlation.TlbAutoScan
■ TRANSlation.TRANSparent	■ MMU	■ MMU.FORMAT	❏ TRANS.ENABLE()
❏ TRANS.INTERMEDIATE()	❏ TRANS.INTERMEDIATEEX()	❏ TRANS.LINEAR()	❏ TRANS.LINEAREX()
❏ TRANS.LOGICAL()	❏ TRANS.PHYSICAL()	❏ TRANS.PHYSICALEX()	❏ TRANS.TABLEWALK()

▲ 'TRANS Functions (Debugger Address Translation)' in 'General Function Reference'
▲ 'Release Information' in 'Legacy Release History'

Overview TRANSlation

NOTE:	Formerly, the **MMU** command group was used for address translation inside the debugger. With the wide-spread adoption of hardware MMUs, it was necessary to rename this command group to **TRANSlation** to avoid confusion with hardware MMUs.

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

TRANSlation	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.	Lets you access and view the real hardware MMU.

The **TRANSlation** commands are used for the following purposes:

- To debug an OS that runs several processes at the same logical addresses (e.g. Linux, PikeOS, etc.).
- To allow a transparent display of hardware translation tables and OS-based translation tables.
- To provide the user with unrestricted access to the target memory using either logical or physical addresses.

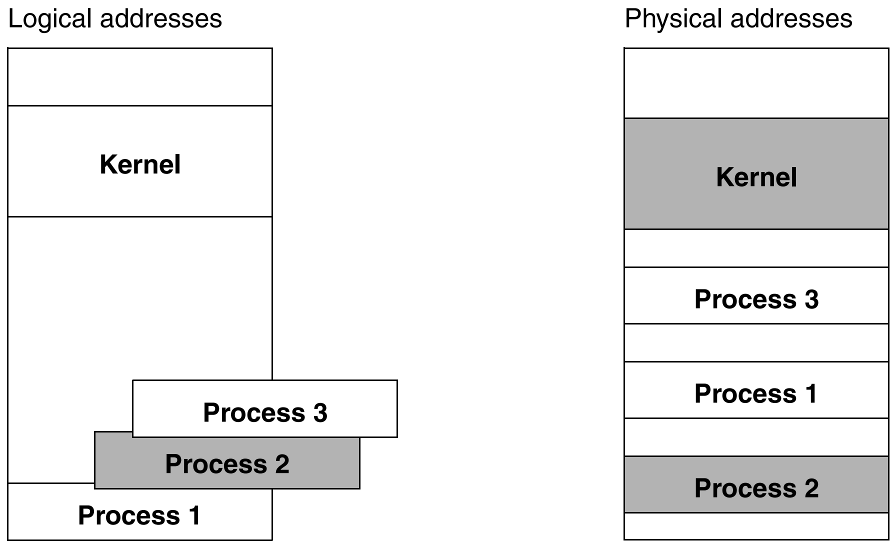
To apply the MMU commands properly, it is important to differentiate between the following MMU table types:

1. **The hardware MMU table**

   The hardware MMU usually consists of registers and/or dedicated memory areas and is held in the CPU. It holds the translation tables that are used by the CPU to translate the logical addresses used by the CPU into the physical addresses required for memory accesses.

   In OSs like Linux, PikeOS, etc. each process has its own address space. Usually all processes start at the logical address 0x0. The result is that, while a process is running, the process has only access to its own address space and to the address space of the kernel.

   The hardware MMU is programmed by the scheduler for this view. If, for example, process 2 is running, the hardware MMU provides only translation tables for process 2 and the kernel.



   - If the OS uses demand paging, the hardware MMU table is extended at each page fault.
   - At each process switch the hardware MMU is reprogrammed so that the logical address space of the current process can be translated to the physical address area.

2. **The software/OS MMU table**

   If an OS like Linux, PikeOS etc. is used, the OS maintains the translation tables for all processes, because the OS is responsible for the reprogramming of the hardware MMU on a process switch.

   The hardware MMU is usually only a subset of the OS MMU tables.

3. **The debugger MMU table**

   If an OS that runs several processes at the same logical addresses (e.g. Linux) is used, the hardware MMU in the CPU only holds translation tables that allow the debugger memory accesses to the code/data of the kernel and the currently running process.

The debugger can access code/data from a not currently running process only with the help of the OS MMU tables. Based on the information held in the translation tables of the OS MMU, the debugger can translate any logical address to a physical address and that way perform a memory access without changing the hardware MMU. If demand paging is used, the OS MMU table contains the translation from the logical to the physical address only if the page was loaded before.

Reading the OS MMU tables on every memory access in quite time consuming. Therefore the debugger can scan the OS MMU tables once and re-use the scan for all following accesses.

The OS MMU table is scanned into the so-called debugger MMU. The debugger MMU provides also the flexibility to add user-defined entries.

	Please be aware that as soon as the debugger MMU is active, all memory accesses performed by the debugger use only the information of the debugger MMU. Please be aware that the OS MMU tables have to be scanned again if the OS has added or removed entries from these tables while running.

TRANSlation.AutoEnable — Auto-enable debugger MMU translation

Format: **TRANSlation.AutoEnable**

Auto-enable the debugger address translation if the CPU's hardware MMU is enabled. When the hardware MMU is on, the debugger also performs translations. When the hardware MMU is off, the debugger performs no translation and treats all logical addresses as physical addresses. The state of the hardware MMU is read from the CPU-specific MMU-enable bit in a system control register. This command is only available for certain CPUs.

See also

- TRANSlation
- TRANSlation.List
- TRANSlation.OFF
- TRANSlation.ON

TRANSlation.AutoSCAN — Autoscan feature for debugger MMU

Format: **TRANSlation.AutoSCAN ON | OFF**

If the operating system adds or removes entries from its page table while running those changes are not performed within the debugger MMU. Trying to access those newly created logical addresses with the debugger may cause an error. If **TRANSlation.AutoSCAN** is set to ON the translation tables hold by the operating system are automatically scanned into the debugger MMU, if the debugger fails to access a logical address.

TRANSlation.AutoSCAN scans only pages that are already present. Depending on the JTAG speed of the processor and on the number of processes in the system scanning the translation tables can take some time. In this cases autoscanning may be more disturbing than helping.

See also

- TRANSlation
- TRANSlation.List

TRANSlation.CacheFlush Flush TRACE32 address translation cache

Format: **TRANSlation.CacheFlush** [**ALL**]

Successful MMU address translations are cached internally by TRACE32. This speeds up recurring accesses to a logical address in debug mode - mostly when the OS Awareness is enabled. Caching is most beneficial for translations done via an MMU table walk as this generates many memory accesses while parsing the OS page table.

TRANSlation.CacheFlush flushes the TRACE32 internal address translation caches, so a new readout of the OS page table is enforced for the next memory access. This can be useful when modifying page table content or debugging MMU table walks.

ALL	Additionally invalidates the complete register set cached by the debugger, including all cached MMU registers. Upon the next MMU page table walk, the registers will be re-read from the target.

See also

- TRANSlation
- TRANSlation.List
- TRANSlation.TableWalk

TRANSlation.CLEANUP Clean up MMU table

Format: **TRANSlation.CLEANUP**
MMU.CLEANUP (deprecated)

Removes multiple translations for one physical address, directly joining translations and double translations.

See also

- TRANSlation
- TRANSlation.List

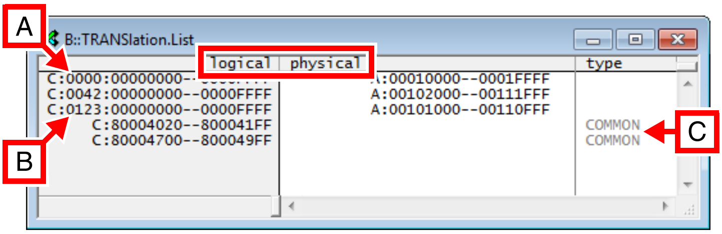
[Example]

Format:	**TRANSlation.COMMON** *<logical_range>* [{*<logical_range>*}] **MMU.COMMON** (deprecated)

Defines one or more mappings of logical address ranges that are shared by the kernel and the tasks.

When the address of a memory access falls into a common address range, TRACE32 uses the kernel address translation (and not the task page table of the current process). Internally, TRACE32 always uses space ID 0x0000 to find the translation of a common address.

This allows to apply the kernel address translation to modules or libraries that are called by a user process in the context of the currently running task.



A Space ID of the kernel = 0x0000

B Space IDs of the tasks ≠ 0x0000
TRACE32 assigns space IDs if **SYStem.Option.MMUSPACES** is set to **ON**

C Common logical address ranges are flagged as "COMMON" in the **TRANSlation.List** window, which displays the mappings between logical and physical address ranges.

<logical_range>	You can specify up to 10 common ranges in one line. For an example with two common address ranges, see below. Overlapping common address ranges are merged automatically.

NOTE:	Executing the **TRANSlation.COMMON** command again discards all previously existing common address ranges. Use **TRANSlation.COMMON.ADD** to add further common address ranges without discarding existing common address ranges.

Example:

```
;Enable the space IDs to display them in the TRANSlation.List window
SYStem.Option.MMUSPACES ON

TRANSlation.List ;Open the Translation.List window

;Create some translation entries for a particular debug session
TRANSlation.Create 0x0:0x00000000++0xffff 0x10000
TRANSlation.Create 0x123:0x00000000++0xffff 0x101000
TRANSlation.Create 0x042:0x00000000++0xffff 0x102000

;Define two common logical address ranges:
;                    <logical_range_1>        <logical_range_2>
TRANSlation.COMMON 0x80004020--0x800041ff   0x80004700--0x800049ff

;Alternatively, you can define the common ranges as follows:
;TRANSlation.COMMON 0x80004020--0x800041ff      ;define the first range
;TRANSlation.COMMON.ADD 0x80004700--0x800049ff  ;add the second range
```

See also

- TRANSlation.COMMON.ADD
- TRANSlation
- MMU.FORMAT
- TRANSlation.COMMON.CLEAR
- TRANSlation.List

▲ 'Release Information' in 'Legacy Release History'

TRANSlation.COMMON.ADD — Add another common address range

Format:	**TRANSlation.COMMON.ADD** *<logical_range>*

Adds another mapping for a common logical address range that is shared by the kernel and the tasks.

NOTE:	Use **TRANSlation.COMMON.ADD** to add further common address ranges without discarding existing common address ranges. Executing **TRANSlation.COMMON** again discards all previously existing common address ranges.

Example:

```
;Define the first common logical address range
TRANSlation.COMMON     0x80000200--0x800007ff

;Add two additional ranges
TRANSlation.COMMON.ADD 0x80004020--0x800041ff
TRANSlation.COMMON.ADD 0x80004700--0x800049ff
```

See also

- TRANSlation.COMMON
- TRANSlation.COMMON.CLEAR

TRANSlation.COMMON.CLEAR — Clear all common logical address ranges

Format:	**TRANSlation.COMMON.CLEAR**

Clears only those logical address ranges that are flagged as “COMMON” in the **TRANSlation.List** window.

See also

- TRANSlation.COMMON
- TRANSlation.COMMON.ADD

Format:	**TRANSlation.Create** *<logical_range>* [*<physical_range>*] [**/***<option>*] **MMU.Create** (deprecated)
<option>:	**More** **Logical** **Physical**

The defined translation can either be function code specific or generic for all function codes (except I/O). The physical address or range is not allowed on probes with fixed MMU translation (e.g. 80186,Z180).

More	The **More** option suppresses the generation of the MMU tables. This speeds up the entry of large translation tables with PRACTICE scripts (*.cmm). The last translation command should not have a **More** option, otherwise the translations are not accessible.
Logical **Physical**	The **Logical** and **Physical** options create translations that work only in one direction. This allows to create multiple logical addresses that map to the same physical address and still having a well-defined logical address for the reverse translation.

Example: Translation for 68030 TRANSlation

```
TRANSlation.Create 0x1000--0x1fff a:0x20000--0x20fff /More
TRANSlation.Create sd:0x2000--0x2fff asd:0x0--0x0fff /More
TRANSlation.Create ud:0x2000--0x2fff aud:0x1000--0x1fff /More
TRANSlation.Create sp:0x2000--0x2fff asp:0x2000--0x2fff /More
TRANSlation.Create up:0x2000--0x2fff aup:0x3000--0x3fff
```

See also

■ TRANSlation ■ TRANSlation.List ■ TRANSlation.TlbAutoScan ■ MMU.FORMAT
■ MMU.SCAN

▲ 'Arm Specific Implementations' in 'Arm Debugger'
▲ 'Arm Specific Implementations' in 'Armv8 and Armv9 Debugger'

TRANSlation.CreateID

Add entry to MMU space ID table

Format: **TRANSlation.CreateID** *<space_id>***:0x0** *<base_address>*

<space_id>	Space ID to be added.
<base_address>	Physical base address of task page table associated with *<space_id>*.

See also

- TRANSlation
- TRANSlation.DeleteID
- TRANSlation.List
- TRANSlation.ListID
- TRANSlation.ScanID

TRANSlation.CreateTab

Create multiple translations

Format: **TRANSlation.CreateTab** *<logical_range> <increment> <logical_range>* [*<physical_range>*] [**/***<option>*]

MMU.CreateTab (deprecated)

<option>: **More**
Logical
Physical

Same as **TRANSlation.Create**, but creates multiple translations with one command. The first range defines the logical range for the created translations. The increment parameter is the offset added to the logical address to generate the next address. The other parameters are interpreted like the **TRANSlation.Create** command.

Example:

```
; Translation for COMMON area from 0x08000--0x0ffff
TRANSlation.CreateTab 0x0--0x0fffff 0x10000 0x08000--0x0ffff 0x08000--
0x0ffff

; Translation for 16 BANKS
TRANSlation.CreateTab 0x0--0x0fffff 0x10000 0x0--0x7fff
```

See also

- TRANSlation
- TRANSlation.List

TRANSlation.Delete — Delete translation

Format:	**TRANSlation.Delete** *<logical_range>* **MMU.Delete** (deprecated)

The defined translation is removed from the list; see **TRANSlation.List**. Use **TRANSlation.Delete** without parameter to clear the whole static translation list and the command **TRANSlation.RESet** to reset all TRANSlation and MMU settings.

Example:

```
TRANSlation.Delete 0x1000--0x1fff
```

See also

- TRANSlation
- TRANSlation.List

TRANSlation.DeleteID — Remove entry from MMU space ID table

Format:	**TRANSlation.DeleteID** *<space_id>***:0x0**

<space_id>	Space ID to be removed.

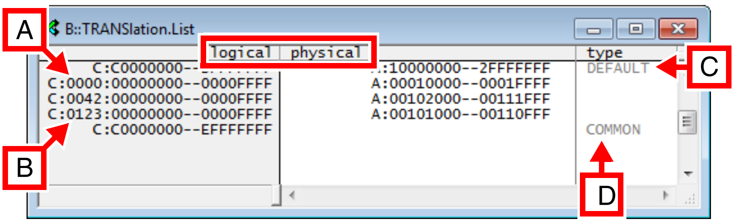
See also

- TRANSlation
- TRANSlation.CreateID
- TRANSlation.List
- TRANSlation.ListID
- TRANSlation.ScanID

Format:	**TRANSlation.List** [**/Logical**	**/Physical**] **MMU.List** (deprecated)

Displays the list of static address translations created with the commands **TRANSlation.Create** or **MMU.SCAN**

The static MMU translation table of TRACE32 contains relations between logical address spaces and physical address spaces. This table is consulted when the debugger address translation is enabled with **TRANSlation.ON** and a logical address must be converted into a physical address. In some cases this table is also used for reverse translating a physical address into its logical counterpart.



A Space ID of the kernel = 0x0000

B Space IDs of the tasks ≠ 0x0000
TRACE32 assigns space IDs if **SYStem.Option.MMUSPACES** is set to **ON**

C The default logical-to-physical address translation, which is used for fast memory accesses into the kernel address range.
The default address translation is specified with the command **MMU.FORMAT**.

D Common address ranges are created with the commands **TRANSlation.COMMON** or **TRANSlation.COMMON.ADD**

Logical	Sorts logical addresses in ascending order.
Physical	Sorts physical addresses in ascending order.

See also

- TRANSlation.ListID
- TRANSlation
- TRANSlation.AutoEnable
- TRANSlation.AutoSCAN
- TRANSlation.CacheFlush
- TRANSlation.CLEANUP
- TRANSlation.COMMON
- TRANSlation.Create
- TRANSlation.CreateID
- TRANSlation.CreateTab
- TRANSlation.Delete
- TRANSlation.DeleteID
- TRANSlation.NoProtect
- TRANSlation.OFF
- TRANSlation.ON
- TRANSlation.PAGER
- TRANSlation.Protect
- TRANSlation.RESet
- TRANSlation.SCANall
- TRANSlation.ScanID
- TRANSlation.SHADOW
- TRANSlation.state
- TRANSlation.TableWalk
- TRANSlation.TlbAutoScan
- TRANSlation.TRANSparent
- MMU.FORMAT
- MMU.SCAN
- MMU.DEFAULTPT()
- MMU.FORMAT()

▲ 'Release Information' in 'Legacy Release History'

Format:	**TRANSlation.ListID**

The table is used to translate MMU root pointer contents into a memory space ID. Memory space IDs may also be obtained from the OS Awareness without the use of this table.

See also

- TRANSlation.List
- TRANSlation
- TRANSlation.CreateID
- TRANSlation.DeleteID

TRANSlation.NoProtect

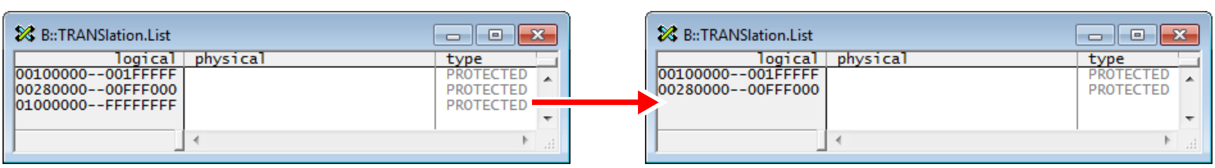
Unprotect memory

Format:	**TRANSlation.NoProtect** *<logical_range>* **MMU.NoProtect** (deprecated)

Removes the protection for the specified logical address range. As a result, the debugger can access this range. See **TRANSlation.Protect.ADD**.

Example:

```
TRANSlation.Protect.ADD 0x100000--0x1fffff              ;no access here
TRANSlation.Protect.ADD 0x280000--0x0fff000             ;no access here
TRANSlation.Protect.ADD 0x1000000--0x0ffffffff          ;no access here

TRANSlation.ON
TRANSlation.List            ;display overview of protected memory ranges

;your code

;remove this logical address range from the list of protected memory
;ranges
TRANSlation.NoProtect 0x1000000--0x0ffffffff
```



See also

- TRANSlation
- TRANSlation.List
- TRANSlation.Protect
- TRANSlation.Protect.ADD

TRANSlation.OFF — Deactivate debugger address translation

Format:	**TRANSlation.OFF** **MMU.OFF** (deprecated)

Deactivates the TRACE32 internal debugger address translation.

Logical addresses used by the debugger are directly sent to the target CPU without translation. Also, the protection of address ranges which have been declared as protected is disabled.

See also

- TRANSlation.ON
- TRANSlation
- TRANSlation.AutoEnable
- TRANSlation.List

TRANSlation.ON — Activate debugger address translation

Format:	**TRANSlation.ON** **MMU.ON** (deprecated)

Activates the TRACE32 internal debugger address translation. For Intel® architecture debuggers, the address translation is enabled by default. For all other architectures, the default is **TRANSlation.OFF**.

With **TRANSlation.ON**, the following features are enabled:

- Logical addresses are translated to physical addresses. The address translation is based on the following translation tables:
    - The static address translation list (see **TRANSlation.List**)
    - Intel® architectures only: the segment translation for boot mode, real mode and protected mode (see **MMU.view**, **MMU.DUMP.GDT**, **MMU.DUMP.LDT** and **SYStem.Option.MEMoryMODEL**)
    - MIPS architectures only: the EVA or fixed mapping KSEG0/1 translations are done.
    - OS page tables if the TRACE32 table walk is enabled (see **TRANSlation.TableWalk** and **MMU.FORMAT**).
    - For some architectures, TLBs can be evaluated. This feature is also enabled with **TRANSlation.TableWalk** and **MMU.FORMAT**.
- Address ranges declared as protected are no longer accessible to the debugger (see **TRANSlation.Protect**).

For an overview of the state of the debugger address translation, see **TRANSlation.state**.

See also

- TRANSlation.OFF
- TRANSlation
- TRANSlation.AutoEnable
- TRANSlation.List
- ❏ TRANS.ENABLE()

TRANSlation.PAGER — Allow paged breakpoints for Linux

Format:	**TRANSlation.PAGER ON** *<address>* | **OFF**
	MMU.Protect (deprecated)

The TRACE32 software and a suitable Linux patch enable a software breakpoint to be set for program code that has not yet been loaded.

Details on the Linux patch can be found in the directory ~~/demo/arm/kernel/linux/etc/t32pager

See also

- TRANSlation
- TRANSlation.List

Using the **TRANSlation.Protect** command group, you can protect the entire logical address range or individual logical address ranges from debugger access. This can be useful if an access would otherwise cause a fatal hardware error or cause the debugger to go down.

What is the difference between...?

TRANSlation.Protect.ON	**TRANSlation.Protect.ADD**
• Protects the *entire* logical address range from debugger access. • However, you can allow debugger access to individual logical address ranges by specifying them with **TRANSlation.Create** *<logical_range>*.	• Protects *individual* logical address ranges from debugger access. • **TRANSlation.Protect** must **not** be set to **ON**.

See also

- TRANSlation.Protect.ADD
- TRANSlation.Protect.OFF
- TRANSlation.Protect.ON
- TRANSlation
- TRANSlation.List
- TRANSlation.NoProtect

[Example]

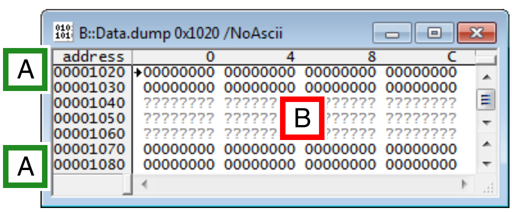
Format:	**TRANSlation.Protect.ADD** *<logical_range>*

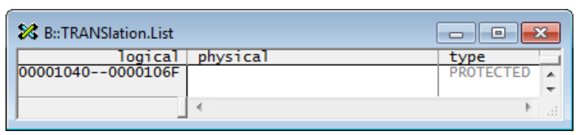
Protects the specified logical address range from debugger access.

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

Example:

```
;[A] allow debugger access to the logical address ranges 0x0--0x103F
;    and 0x1070--0xFFFFFFFF, i.e. almost the entire logical range, ...

;[B] ...but protect this logical address range from debugger access
TRANSlation.Protect.ADD 0x1040--0x106F

TRANSlation.ON

;display overview of protected memory range(s)
TRANSlation.List

;let’s open this window for demo purposes to visualize the result
Data.dump 0x1020 /NoAscii
```





See also

- TRANSlation.Protect
- TRANSlation.NoProtect
- MAP.DenyAccess

Format:	**TRANSlation.Protect.OFF**

Re-allows debugger access to the entire logical address range. See **TRANSlation.Protect.ON**.

See also

- TRANSlation.Protect

TRANSlation.Protect.ON

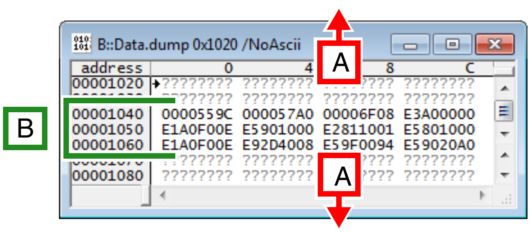
Protect entire target memory

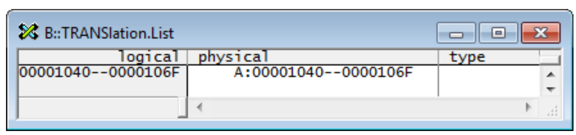
Format:	**TRANSlation.Protect.ON**

Protects the entire logical address range from debugger access, provided the address translation is enabled with **TRANSlation.ON**.

Example:

```
TRANSlation.ON

;protect entire logical address range from debugger access (see red [A])
TRANSlation.Protect.ON

;but allow debugger access to this logical address range (see green [B]
TRANSlation.Create 0x1040--0x106F

;display overview of static translations
TRANSlation.List

;let's open this window for demo purposes to visualize the result
Data.dump 0x1020 /NoAscii
```





See also

- TRANSlation.Protect

TRANSlation.RESet — Reset MMU configuration

Format:	**TRANSlation.RESet** **MMU.RESet** (deprecated)

The translation table is cleared and all setups are set to the defaults.

See also

- TRANSlation
- TRANSlation.List

TRANSlation.SCANall — Scan MMU tables

Format:	**TRANSlation.SCANall** [**/***<option>*] **MMU.SCAN ALL** (as an alias) **MMU.SCANALL** (deprecated)
<option>:	**Clear**

Scans all page translation tables known to the debugger into the static translation list. That is, this command is a repeated call of the **MMU.SCAN** command for all known page tables of an architecture known to the debugger.

Clear	Clears the list of static translations before reading it from all page translation tables.

See also

- TRANSlation
- TRANSlation.List
- MMU.SCAN

TRANSlation.ScanID — Scan MMU address space tables from kernel

Format:	**TRANSlation.ScanID**

Scans the translation information from the kernel into the MMU space ID table. The operation is target and kernel dependent.

See also

- TRANSlation
- TRANSlation.CreateID
- TRANSlation.DeleteID
- TRANSlation.List

TRANSlation.SHADOW — Enable shadow access to target memory

Format:	**TRANSlation.SHADOW** [**ON** | **OFF** | **ANY**]

Use VM: for data access, if the address translation on the target failed.

The debugger first tries to resolve a logical address with the standard address translation, and then accesses the target to read the requested data. If the translation fails (due to missing table entries, or due to an access error), and if **TRANSlation.SHADOW** is **ON**, the debugger uses the data within VM: at the requested address.

The debugger provides a “virtual memory” (access class VM:) that is not accessible from the CPU, but only by the debugger (stored within the host). The idea is to have a (partial) copy of the target memory in the host for unlimited access.

VM: usually is a “virtual physical memory”. The debugger does an address translation (logical -> physical), then accesses VM: with the physical address. I.e. VM: maps a physical memory.

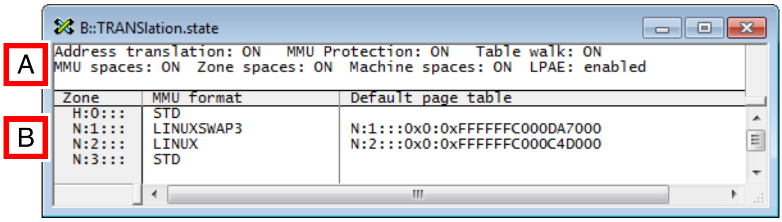
If **TRANSlation.SHADOW** and **SYStem.Option.MMUSPACES** is ON, VM: is used as several logical addressed memory areas, separated by the space ID. No address translation is done, instead the debugger directly accesses the memory in VM: with space ID:address. I.e. VM: maps several logical memory areas. In complex OS target systems (e.g. Linux), you may load the code of several processes into VM: to have access to the code, even if the target does currently not allow memory access.

See also

- TRANSlation
- TRANSlation.List

Format:	**TRANSlation.state**

Opens the **TRANSlation.state** window.



A The header displays an overview of all settings affecting the debugger address translation:

- Address translation: ON, OFF = TRANSlation.ON or TRANSlation.OFF
- MMU protection: ON, OFF = TRANSlation.Protect.ON or TRANSlation.Protect.OFF
- Table walk: ON, OFF = TRANSlation.TableWalk [ON | OFF]
- MMU spaces: ON, OFF = SYStem.Option.MMUSPACES [ON | OFF]
- Zone spaces: ON, OFF = SYStem.Option.ZoneSPACES [ON | OFF]
- Machine spaces: ON, OFF = SYStem.Option.MACHINESPACES [ON | OFF]
- Architecture-specific settings (here LPAE)

B The columns below the header list the settings configured with the **MMU.FORMAT** command.

Description of Columns in the TRANSlation.state Window

Zone	For information about zones, refer to the glossary.
MMU format	The MMU formats for each zone.
Default page table	The start addresses of the default page tables for all active zones.

See also

- TRANSlation
- TRANSlation.List

Format:	**TRANSlation.TableWalk** [**ON** | **OFF**]

Configures the debugger to perform an MMU page table walk (short: table walk). If enabled, the debugger will try the following steps upon a logical-to-physical address translation request:

1. Look up the logical address in the debugger's static address translation table (see **TRANSlation.List** and **TRANSlation.Create** for details about the static address translation table).
2. If the address lookup in the static address translation table fails, walk through the software/OS MMU tables to find a valid logical-to-physical translation.
3. For Intel® architecture debuggers, the boot mode, real mode, or protected mode segment translation is done before the page table walk is performed.
4. For MIPS architectures only: the EVA or fixed mapping KSEG0/1 translations are done before the page table walk is performed.

Valid address translations found are cached by TRACE32. When debug mode is left, i.e. at a **Go** or **Step**, the cached translations are flushed because page table contents may change when the target continues execution.

ON	Configure TRACE32 to use the automatic MMU table walk. Only physical addresses are sent to the target. **NOTE for expert users**: For some architectures, although a valid translation is available, TRACE32 sends logical addresses in certain situations in order to ensure cache coherency. This behavior can be controlled with the architecture-specific command **SYStem.Option.MMUPhysLogMemaccess**.
OFF	Configure TRACE32 to not use the automatic MMU table walk.

NOTE:	Page tables are dynamic structures and are frequently modified by the OS. The MMU page table walk of the debugger dynamically parses the page tables on demand for every debugger address translation. The table walk ensures that the debugger address translations correspond to the current OS address translations.

If no valid translation could be found for a logical address in any available translation table, then the error handling depends on whether **TRANSlation.TableWalk** is set to **OFF** or **ON**:

OFF	No error will be produced by TRACE32. The logical address will be sent to the target CPU without translation.
ON	A “MMU translation failed” error will be produced by TRACE32. Scripts will stop upon a failing translation. This mimics the behavior of the target MMU, where a failing translation causes an exception.

See also

■ TRANSlation ■ TRANSlation.CacheFlush ■ TRANSlation.List ■ MMU.FORMAT
❏ TRANS.TABLEWALK()

▲ 'Arm Specific Implementations' in 'Arm Debugger'
▲ 'Arm Specific Implementations' in 'Armv8 and Armv9 Debugger'
▲ 'Release Information' in 'Legacy Release History'

TRANSlation.TlbAutoScan — Allow automatic TLB scans during table walk

Format:	**TRANSlation.TlbAutoScan** [**ON** | **OFF**] [*<logical_range>* [*<logical_range>*]]

Enable automatic scan of the TLBs for missing kernel address translations during MMU table walks. Ignore TLB entries with logical addresses outside the specified *<logical_range>*.

NOTE:	This command is not available for all architectures

Some OS specify logical base addresses for kernel or task page tables. The table walk algorithm must translate them to physical addresses before the page table can be parsed. If there is no suitable user-defined default translation (**MMU.FORMAT**) or debugger MMU table (**TRANSlation.List**) entry, **TRANSlation.TlbAutoScan** will search the target MMU TLBs for a suitable translation that has been set up and used by the OS itself. If a suitable translation is found, it is copied into the debugger MMU table. This automatism can prevent debugger memory access failures caused by incomplete MMU setup scripts.

NOTE:	Only TLB entries in the kernel address range must be auto-extracted from TLBs. If you specify the typical kernel address range(s) for your target’s OS in *<logical_range>*, **TRANSlation.TlbAutoScan** will ignore dynamic TLB entries used for user processes.

Place the **TRANSlation.TlbAutoScan** command into the MMU section of your PRACTICE script preparing the debugger for OS Awareness as in this example:

```
;*******************************************************************
; example MMU setup section for Linux awareness
; - “TRANSlation.TlbAutoScan ON” replaces the explicit
;   default translation in MMU.FORMAT and fixed kernel
;   address translations in TRANSlation.Create.
;*******************************************************************

PRINT "Initializing debugger MMU..."
MMU.FORMAT LINUX swapper_pg_dir
TRANSlation.COMMON 0xC0000000--0xFFFFFFFF

; this translation will be auto-extracted by TlbAutoScan from the TLB
; TRANSlation.Create 0xC0000000--0xCFFFFFFF 0x0

; enable TlbAutoScan - TLB entries in 0x80000000--0xFFFFFFF are kernel
; addresses here and ok to be auto-extracted
TRANSlation.TlbAutoScan ON 0xC0000000--0xFFFFFFF

TRANSlation.TableWalk ON
TRANSlation.ON
```

See also

- TRANSlation
- TRANSlation.Create
- TRANSlation.List
- MMU.DUMP
- MMU.FORMAT
- MMU.SCAN

Format: **TRANSlation.TRANSparent** *<logical_range>*
MMU.TRANS *<logical_range>* (deprecated)
MMU.TRANSparent *<logical_range>* (deprecated)

A debugger access to a logical address within *<logical_range>* will not be translated to a physical address, even if a page table translation for it is defined. Instead, this access will use the logical address.

Example

In a banked memory system, we want the debugger to see the current memory bank (selected by the CPU's BNK register) for memory accesses within *<logical_range>*. The following example shows a PRACTICE script for such a setup for a CPU with 16-bit logical addresses:

```
sYmbol.RESet
TRANSlation.RESet
; define fixed translation window into bank 0
TRANSlation.Create 0x1000000--0x100ffff A:0x00000--0x0ffff
; define fixed translation window into bank 1
TRANSlation.Create 0x1010000--0x101ffff A:0x10000--0x1ffff
; define transparent address window (no translation in this range)
TRANSlation.TRANSparent 0x0--0xffff
TRANSlation.ON
; load code into current bank, somewhere in 0x0--0xffff
Data.LOAD.Ubrof example.dbg
; shift symbols to logical addresses at 1000000
sYmbol.RELOCate C:0x1000000
```

Any access within 0x0..0xffff is defined as transparent and will thus not be translated to a physical address by the debugger. Instead, such an access will be carried out with the logical address, so the CPU's "current bank" register will decide which data is seen. That is, examining a variable pointing to a certain logical address somewhere within 0x0..0xffff with bank 1 being active, will show the data stored in bank 1.

We want to make sure that symbols belonging to code or data loaded into a certain bank are always tied to the correct bank. Addresses in 0x0..0xffff may show any bank, depending on the BNK register. So we first define fixed translation windows of 0x1000000..0x100ffff to bank 0 and 0x1010000..0x101ffff to bank 1. Note that those address windows exist only for the debugger.

Now we load code (assuming bank 0 being selected by register BNK) into memory. Finally, we shift the symbols belonging to the code into the address window belonging to bank 0, i.e. we add an offset of 0x1000000 after loading. Now we have a clear assignment between the symbols and the data in bank 0, while debugger accesses to logical addresses in 0x0..0xffff still see the data the CPU sees currently.

See also

■ TRANSlation ■ TRANSlation.List

TrBus

Trigger bus

See also

- TrBus.Arm
- TrBus.Connect
- TrBus.Mode
- TrBus.OFF
- TrBus.Out
- TrBus.RESet
- TrBus.Set
- TrBus.state
- TrBus.Trigger

Overview TrBus

The **TrBus** command group allows:

- To generate a trigger pulse that can be used to trigger an external device e.g. a Logic Analyzer.
- To connect an incoming trigger signal to TRACE32-ICD.

In both cases the TRIG connector is used. The TRIG connector has the following characteristics on the different TRACE32 tools:

TRACE32 tool	Output voltage	Input voltage	Comment
PowerDebug X50/X51	4.4V	3.3V	Input: 5V tolerant, 10K pull-up/down*
PowerDebug PRO/E40	4.4V	3.3V	Input: 5V tolerant, 10K pull-up/down*
µTrace (MicroTrace) for Cortex-M	3.3V	3.3V	Input: 5V tolerant, 10K pull-up/down*
PowerDebug Module USB 3.0	4.4V	3.3V	Input: 5V tolerant, 10K pull-up/down*
PowerDebug II Ethernet	5.0V	3.3V	Input: 5V tolerant, 10K pull-up/down*
PowerDebug Module Ethernet / PowerTrace Ethernet	3.3V	3.3V	Input: 5V tolerant, 10K pull-up/down*

TRACE32 tool	Output voltage	Input voltage	Comment
Power Debug Module USB 2.0	3.3V	3.3V	Input: 5V tolerant, 10K pull-up/down*
PODBUS Ethernet Controller	3.3V	3.3V	Input: 5V tolerant, 10K pull-up/down*
Power Debug Module USB 1.x	3.3V	3.3V	Input: 5V tolerant, 10K pull-up/down*

* Pull-up/down selected automatically depending on low-active or high-active settings.

An **external trigger pulse** of at least 100ns can be generated when:

- The program execution is stopped.
- A trigger is generated for the trace (not available on all CPUs, depends on the implementation of the trace trigger feature).
- The sampling to the trace buffer is stopped ((not available on all CPUs, depends on the implementation of the trace trigger feature).
- A breakpoint with the Action BusTrigger is used (not available on all CPUs).

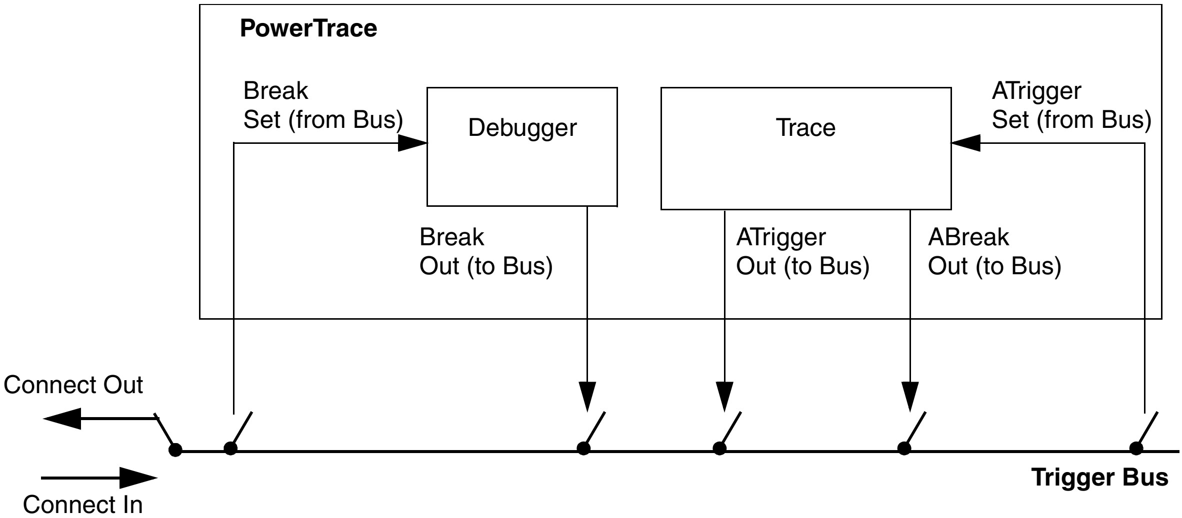
An **incoming trigger signal** can be used:

- To stop the program execution.
- To generate a trigger for the trace (not available on all CPUs, depends on the implementation of the trace trigger feature).

The sources for the external trigger pulse and the targets for the incoming trigger signal are connected to the trigger bus.

Trigger Bus on the PowerTrace

The following picture shows the Trigger Bus on the PowerTrace as an example.



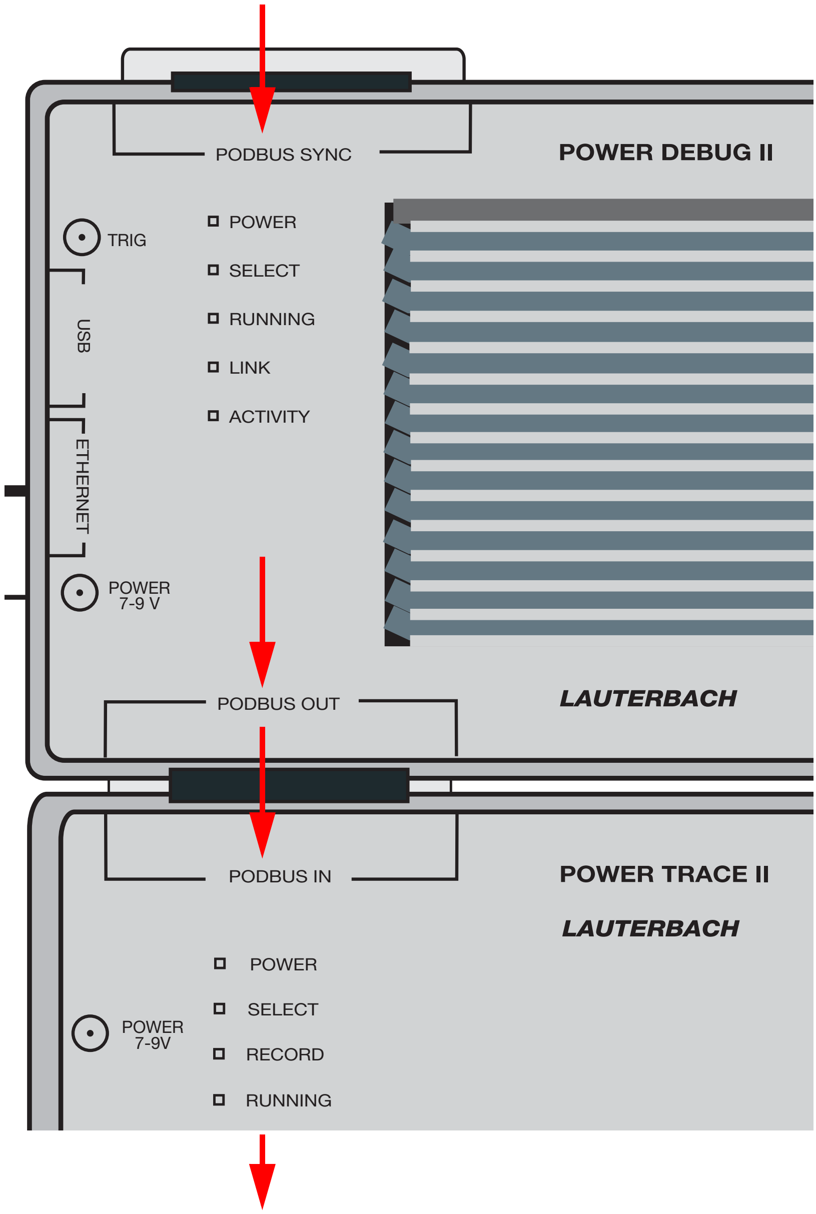
Example: Generate a trigger for the trace at a falling edge of the incoming trigger signal.

```
TrBus.Arm                   ; Switch the trigger bus ON

TrBus.Connect In            ; Configure the TRIGGER connector as input

TrBus.Mode.Falling          ; define that the trigger target should react
                            ; on the falling edge of the incoming trigger
                            ; signal

TrBus.Set ATrigger ON       ; generate a trigger for the trace (trigger
                            ; target) on the falling edge of the external
                            ; trigger signal
                            ; a trigger for the trace can stop the
                            ; sampling to the trace directly or it can be
                            ; delayed by the command Analyzer.TDelay

TrBus.Set Break OFF         ; switch all other sources and targets to OFF
TrBus.Out Break OFF
TrBus.Out ABreak OFF
TrBus.Out ATrigger OFF
```

Interaction Between Independent PODBUS Devices

If several independent PODBUS devices are plugged together, they share the same trigger bus. Example configurations are:

- A POWER DEBUG II and a POWER TRACE II / POWER TRACE III
- A POWER DEBUG INTERFACE / USB and a POWERPROBE
- A POWERTRACE / ETHERNET and a POWERINTERGRATOR
- Several POWER DEBUG INTERFACEs that form a multi-processor debugging environment.



The common trigger bus allows a synchronization between the PODBUS devices.

Example: A soon as the POWERPROBE is stopped by a trigger, the program execution should be stopped via the connected POWER DEBUG INTERFACE:

```
;PowerProbe

;…                               ; definition of the trigger condition
PP:Analyzer.TOut BUSA ON         ; generate a trigger for the trigger bus
                                 ; when the defined trigger event occurs

;Debugger

TrBus.Arm                        ; switch the trigger bus ON
TrBus.Connect Out                ; Configure the TRIGGER connector as output
TrBus.Set Break ON               ; allow any trigger from the trigger bus to
                                 ; stop the program execution
```

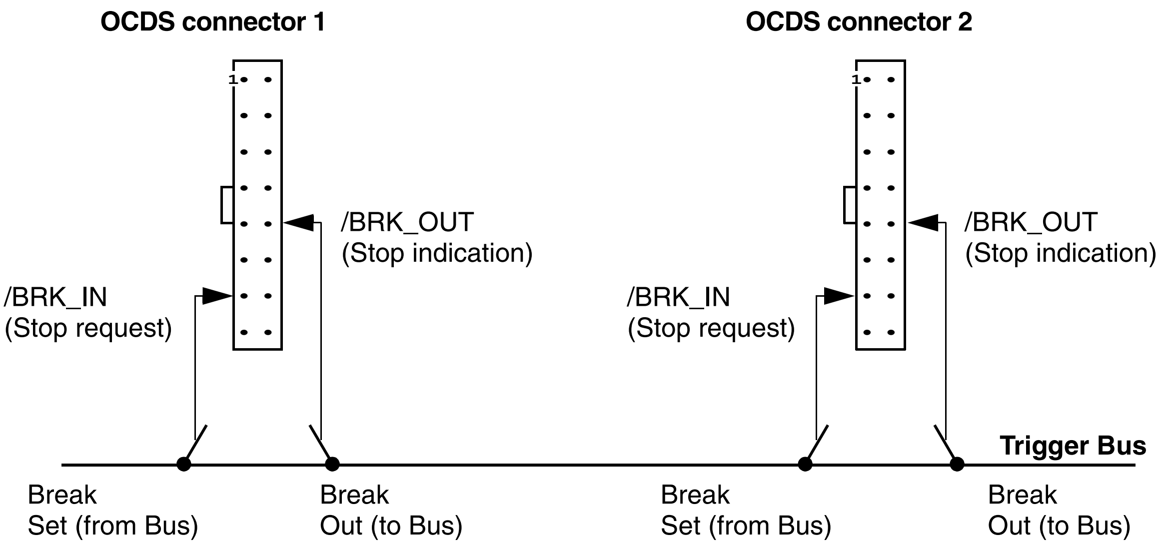
The trigger bus also allows to stop the processors in a multi-processor configuration synchronously. Precondition is that the JTAG/OCDS/BDM … connector provides:

- A signal which indicates that the program execution was stopped (stop indication).
- A signal that allows to stop the program execution immediately (stop request).

After the configuration for the synchronous start and stop by the **SYnch** command is done, you can configure the stop synchronization per hardware by the **TrBus** commands.

	Multicore chip sets provide normally internal (chip specific) trigger connections.

Example: Configure the stop synchronization per hardware for the TRICORE OCDS connector:




```
TrBus.Arm                ; Switch the trigger bus ON

TrBus.Connect Out        ; Configure the TRIGGER connector as output

TrBus.Set Break ON       ; Connect Break to stop request
TrBus.Out Break ON       ; Connect Break to stop indication
```

TrBus.Arm   Arm the trigger bus

Format:	**TrBus.Arm**

Arms the trigger bus.

See also

- TrBus
- TrBus.state

TrBus.Connect

Configure TRIGGER as input or output

Format:	**TrBus.Connect In | Out**

The TRIGGER connector should work as:

- Input for an incoming trigger signal.
- Output for the generation of an external trigger signal.

See also

- TrBus
- TrBus.state

TrBus.Mode

Define polarity/edge for the trigger signal

Format:	**TrBus.Mode Low | High | Falling | Rising**

If **TrBus.Connect Out** is set a Low or High pulse is generated on TRIGGER (at least 100 ns) as soon as the defined source becomes active.

If **TrBus.Connect In** is set, the defined target can react on a Low/High pulse or Falling/Rising edge of the incoming trigger signal.

See also

- TrBus
- TrBus.state

TrBus.OFF

Switch trigger bus off

Format:	**TrBus.OFF**

Switches the trigger bus off.

See also

- TrBus
- TrBus.state

TrBus.Out — Define source for the external trigger pulse

Format:	**TrBus.Out Break | ABreak | ATrigger** [**ON** | **OFF**]

Defines the source for the external trigger pulse.

Break	Generate an external trigger pulse when the program execution is stopped.
ABreak	Generate an external trigger pulse when the sampling to the trace buffer is stopped.
ATrigger	Generate an external trigger pulse when a trigger is generated for the trace. A trigger for the trace can be used to stop the sampling to the trace buffer after a specified delay **Analyzer.TDelay**.

See also

■ TrBus ■ TrBus.state

▲ 'Release Information' in 'Legacy Release History'

TrBus.RESet — Reset setting for trigger bus

Format:	**TrBus.RESet**

Resets the settings for the trigger bus.

See also

■ TrBus ■ TrBus.state

TrBus.Set — Define the target for the incoming trigger

Format:	**TrBus.Set Break** | **ATrigger** [**ON** | **OFF**]

Selects the target for the incoming trigger signal.

Break	Stop the program execution as soon as the external trigger signal becomes active.
ATrigger	Generate a trigger for the trace as soon as the external trigger signal becomes active. A trigger for the trace can be used to stop the sampling to the trace buffer directly or after a specified delay **Analyzer.TDelay**.

See also

- TrBus
- TrBus.state

TrBus.state — Display settings for the trigger bus

Format:	**TrBus.state** **TrBus.view** (deprecated)

Displays all settings for the trigger bus.

See also

- TrBus
- TrBus.Arm
- TrBus.Connect
- TrBus.Mode
- TrBus.OFF
- TrBus.Out
- TrBus.RESet
- TrBus.Set
- TrBus.Trigger

TrBus.Trigger — Stimulate a trigger on the trigger bus

Format:	**TrBus.Trigger**

Stimulates a trigger on the trigger bus.

See also

- TrBus
- TrBus.state

TrOnchip

The **TrOnchip** command group provides low-level access to the on-chip debug register.

TrOnchip — Onchip triggers

See also

- ■ TrOnchip.RESet ■ TrOnchip.state
- ▲ 'CPU specific TrOnchip Commands' in 'CPU32/ColdFire Debugger and Trace'
- ▲ 'Arm Specific TrOnchip Commands' in 'Arm Debugger'
- ▲ 'Arm specific TrOnchip Commands' in 'Armv8 and Armv9 Debugger'
- ▲ 'CPU specific TrOnchip Commands' in 'RH850 Debugger and Trace'
- ▲ 'Nexus specific TrOnchip Commands' in 'RH850 Debugger and Trace'
- ▲ 'TrOnchip' in 'StarCore Debugger and Trace'
- ▲ 'CPU specific TrOnchip Commands' in 'TriCore Debugger and Trace'
- ▲ 'CPU specific TrOnchip Commands - Onchip Triggers' in 'Intel® x86/x64 Debugger'
- ▲ 'Release Information' in 'Legacy Release History'

TrOnchip.RESet — Reset settings to defaults

Format:	**TrOnchip.RESet**

Set on-chip trigger system to initial state.

See also

- ■ TrOnchip ■ TrOnchip.state
- ▲ 'CPU specific TrOnchip Commands' in 'Xtensa Debugger and Trace'
- ▲ 'CPU specific TrOnchip Commands' in 'Simulator for XTENSA'

TrOnchip.state — Display onchip trigger window

Format:	**TrOnchip.state**

Displays a window with the state of the on-chip trigger setting.

See also

- ■ TrOnchip ■ TrOnchip.RESet

▲ 'TrOnchip Commands' in 'CEVA-Oak/Teak/TeakLite Debugger and Trace'
▲ 'Release Information' in 'Legacy Release History'

TrPOD — Trigger probe

NOTE:	The Trigger Probe for PODBUS is out of production.

See also

- TrPOD.Clock
- TrPOD.ClockPOL
- TrPOD.Data
- TrPOD.DataPOL
- TrPOD.Mode
- TrPOD.OFF
- TrPOD.ON
- TrPOD.RESet
- TrPOD.state
- TrPOD.Time

TrPOD.Clock — Defines data mask

Format:	**TrPOD.Clock** [*<mask>*]

The clock mask is defined. Every input line can be high or low or don't care.

See also

- TrPOD
- TrPOD.state

TrPOD.ClockPOL — Defines data polarity

Format:	**TrPOD.ClockPOL** [*<polarity>*]
<polarity>:	**+** | **-**

The clock polarity can be set to true or false.

See also

- TrPOD
- TrPOD.state

TrPOD.Data

Defines data mask

Format:	**TrPOD.Data** [<*mask*>]

The data mask is defined. Every input line can be high or low or don't care.

See also

- TrPOD
- TrPOD.state

TrPOD.DataPOL

Defines data polarity

Format:	**TrPOD.DataPOL** [<*polarity*>]
<*polarity*>:	**+** | **-**

The data polarity can be set to true or false.

See also

- TrPOD
- TrPOD.state

Format:	**TrPOD.Mode** [*<mode>*]
<mode>:	**DATA** **CLOCK** **SYNC** **LONGER** **SHORTER** **GLITCH** **GLITCH+** **GLITCH-**

The state display shows all the settings of the trigger probe and the level of the input pins.

DATA	Asynchronous trigger on inputs with data comparator
CLOCK	Asynchronous trigger on inputs with clock comparator
SYNC	Synchronous trigger
LONGER	Pulse width trigger when pulse exceeds time
SHORTER	Pulse width trigger when pulse width below time limit
GLITCH	Glitch trigger on both edges
GLITCH+	Glitch trigger on positive glitch
GLITCH-	Glitch trigger on negative glitch

See also

■ TrPOD ■ TrPOD.state

TrPOD.OFF — Switch off

Format:	**TrPOD.OFF**

The trigger probe is disabled.

See also

- TrPOD
- TrPOD.state

TrPOD.ON — Switch on

Format:	**TrPOD.ON**

The trigger probe is enabled.

See also

- TrPOD
- TrPOD.state

TrPOD.RESet — Reset command

Format:	**TrPOD.RESet**

The trigger probe is initialized to the default setup condition

See also

- TrPOD
- TrPOD.state

TrPOD.state

State display

Format:	**TrPOD.state**

Using this command the operating mode of the analyzer may be selected. During operation this command displays the current state of the analyzer.

See also

- TrPOD
- TrPOD.Clock
- TrPOD.ClockPOL
- TrPOD.Data
- TrPOD.DataPOL
- TrPOD.Mode
- TrPOD.OFF
- TrPOD.ON
- TrPOD.RESet
- TrPOD.Time

▲ 'Release Information' in 'Legacy Release History'

TrPOD.Time

Defines the time for the pulse width trigger

Format:	**TrPOD.Time** [<*time*>]

The time limit for the pulse width detection can be set between 20 ns and 6 ms.

See also

- TrPOD
- TrPOD.state