General Commands Reference Guide P

Release 09.2023

General Commands Reference Guide P

Version 09-Oct-2023

History

28-Aug-2023	New command PER.<format>.TestProgram.
21-Jul-2023	New command PER.IMPORT.SortTopTrees.
07-Apr-2023	New PER.IMPORT command group.
05-Aug-2022	New **/AccessClass** option for PERSVD commands.
06-May-2022	New command PER.Set.ByName.

PCI — Legacy PCI configuration

The command group **PCI** supports the access to the legacy PCI configuration space (first 256 bytes of device data).

NOTE:	This command group is only implemented for a few specific chips.

See also

■ PCI.Dump	■ PCI.Option.DOMAIN	■ PCI.Read	■ PCI.Scan
■ PCI.Write			

PCI.Dump Display PCI device data

Format:	**PCI.Dump** *<bus>* *<device>* *<function>* [**/***<option>*]
<bus>:	**0..Max_PCI_Busnumber**
<device>:	**0..Max_PCI_Devicenumber**
<function>:	**0..Max_PCI_Functionnumber**
<option>:	**Byte** | **Word** | **Long** | **Quad** **BE** | **LE**

Displays the raw PCI device data.

<bus>	PCI bus number
<device>	PCI device number
<function>	PCI function number
<option>	Data display format and endianness

See also

■ PCI

PCI.Option.DOMAIN Set PCI domain

Format:	**PCI.Option.DOMAIN** *<domain>*
<domain>:	**0…65535**

Default: 0

Configures the PCI domain used as default by other **PCI** commands. A PCI domain is an isolated set of PCI bus segments. Usually multiple PCI domains are used when there are multiple independent PCI controllers on a chip.

See also

■ PCI

PCI.Read — Read a PCI register

Format:	**PCI.Read** *<bus> <device> <function> <register>* [**/***<option>*]
<bus>:	**0..Max_PCI_Busnumber**
<device>:	**0..Max_PCI_Devicenumber**
<function>:	**0..Max_PCI_Functionnumber**
<register>:	**0..Max_PCI_Registernumber**
<option>:	**Byte** | **Word** | **Long** | **Quad** **BE** | **LE**

Reads the selected PCI register. The read access is always 32bit (long), using a byte or word format is only for convenience.

<bus>	PCI bus number
<device>	PCI device number
<function>	PCI function number
<register>	PCI register number
<option>	Data display format and endianness

See also

■ PCI

Format:	**PCI.Scan** [*<range>*]
<range>:	*<start>***--***<end>*
<start>:	**0..Max_PCI_Busnumber**
<end>:	**0..Max_PCI_Busnumber**

Scans the PCI bus and lists the found devices.

<range>	PCI bus range, default: 0.--1.
<start>	*<start>* **must be smaller than or equal to** *<end>.*
<end>	*<end>* **must be greater than or equal to** *<start>.*

See also

■ PCI

Format:	**PCI.Write** *<bus> <device> <function> <register>* [**%***<format>*] *<value>*
<bus>:	**0..Max_PCI_Busnumber**
<device>:	**0..Max_PCI_Devicenumber**
<function>:	**0..Max_PCI_Functionnumber**
<register>:	**0..Max_PCI_Registernumber**
<format>:	**Byte** | **Word** | **Long** | **Quad** **BE** | **LE**
<value>:	**Number**

Writes the selected PCI register. The write access is always 32bit (long), using a byte or word format is only for convenience (read-modify-write operation).

<bus>	PCI bus number
<device>	PCI device number
<function>	PCI function number
<register>	PCI register number
<format>	Data display format and endianness
<value>	New PCI register value

See also

- PCI

PCPOnchip

The **PCPOnchip** command group allows to display and analyze the PCP trace information stored to the on-chip trace provided by an ED device e.g. for the TriCore architecture.

The **PCPOnchip** command is only applicable if the PCP debugging and tracing is performed with the same TRACE32 instance then the core debugging (legacy PCP).

For a description of the command usage, refer to the **<trace>** command group.

PER

Peripheral files

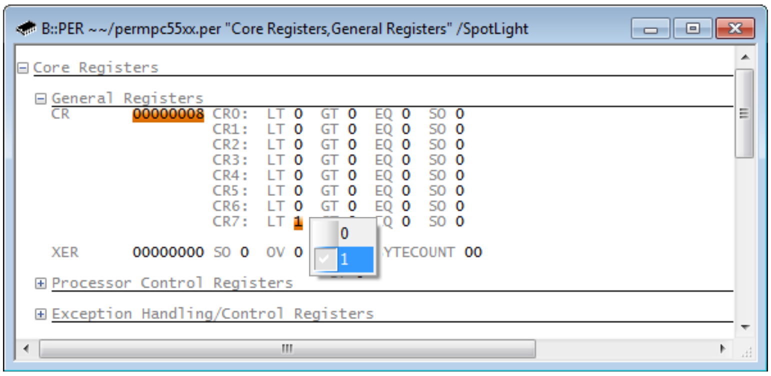
See also

■ PER.IMPORT	■ PER.In	■ PER.Program	■ PER.ReProgram
■ PER.ReProgramDECRYPT	■ PER.Set	■ PER.STOre	■ PER.TestProgram
■ PER.view	■ PER.viewDECRYPT	■ SETUP.DropCoMmanD	

▲ 'Release Information' in 'Legacy Release History'

Overview PER

The command **PER.view** displays a window with a view on the control registers of integrated peripherals. The so-called *peripherals files* (*.per) controlling the contents of this window can be freely configured for displaying memory structures or I/O structures.

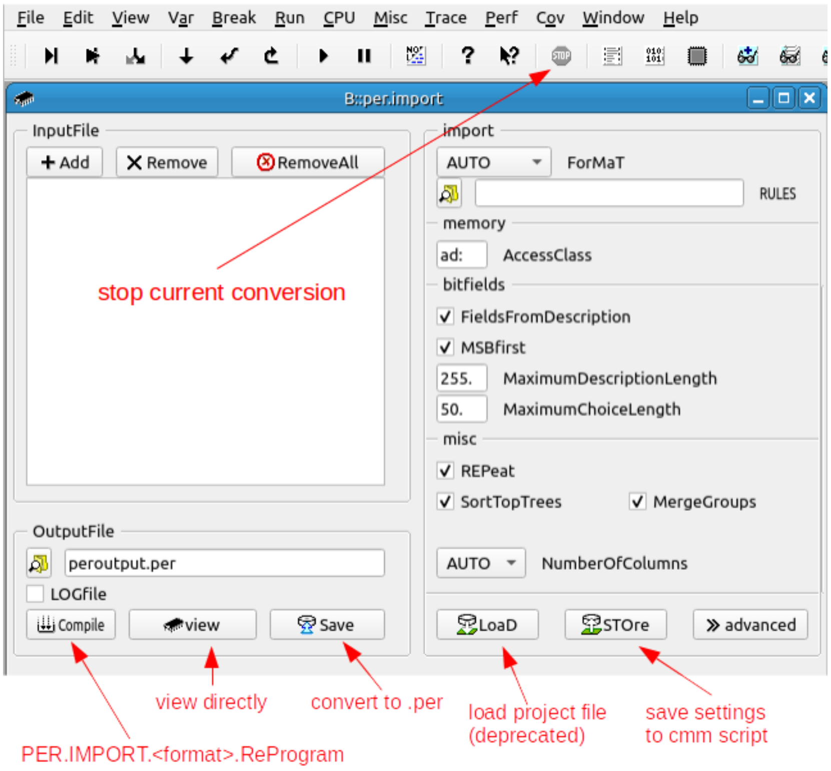


```
B::PER ~~/permpc55xx.per "Core Registers,General Registers" /SpotLight

⊟ Core Registers

  ⊟ General Registers
    CR        00000008 CR0:  LT 0  GT 0  EQ 0  SO 0
                       CR1:  LT 0  GT 0  EQ 0  SO 0
                       CR2:  LT 0  GT 0  EQ 0  SO 0
                       CR3:  LT 0  GT 0  EQ 0  SO 0
                       CR4:  LT 0  GT 0  EQ 0  SO 0
                       CR5:  LT 0  GT 0  EQ 0  SO 0
                       CR6:  LT 0  GT 0  EQ 0  SO 0
                       CR7:  LT 1        Q 0  SO 0

    XER       00000000 SO 0  OV 0      YTECOUNT 00

  ⊞ Processor Control Registers

  ⊞ Exception Handling/Control Registers
```

All microcontroller emulation probes are supported by a file which describes the internal peripherals. This file may be modified (using logical names instead of pin numbers for i/o ports) or extended to display additional peripherals outside the microcontroller.

Examples for different microcontrollers reside in the directory `~~/demo/per`.

The native peripheral file format is *.per. Though TRACE32 is able to import other file formats such as SVD or various XML derivatives. Imported files can directly we opened in a **PER.<format>.view** window or saved to native .per format using **PER.<format>.Save**.



See also

- PER.IMPORT.AccessClass
- PER.IMPORT.ForMaT
- PER.IMPORT.LoaD
- PER.IMPORT.MaximumDescriptionLength
- PER.IMPORT.ModuleFiles
- PER.IMPORT.NumberOfColumns
- PER.IMPORT.REPeat
- PER.IMPORT.RULES
- PER.IMPORT.STOre
- PER
- PER.IMPORT.FieldsFromDescription
- PER.IMPORT.InputFile
- PER.IMPORT.MaximumChoiceLength
- PER.IMPORT.MergeGroups
- PER.IMPORT.MSBfirst
- PER.IMPORT.OutputFile
- PER.IMPORT.RESet
- PER.IMPORT.SortTopTrees
- PER.IMPORT.WithValue
- PER.view

PER.<format>.ReProgram

Set default peripheral file

[build 155354 - DVD 09/2023]

Format:	**PER.<format>.ReProgram** *<file>*

Same as **PER.ReProgram** for converted peripheral files.

PER.<format>.Save

Save to file

[build 155354 - DVD 09/2023]

Format:	**PER.<format>.Save**

Convert input file(s) and save as .per file. The output file is configured by the **PER.IMPORT.OutputFile** command.

PER.<format>.TestProgram

Test mode

[build 155354 - DVD 09/2023]

Format:	**PER.<format>.TestProgram**

Same as **PER.TestProgram** for converted peripheral files.

PER.<format>.view

Display peripherals

[build 155354 - DVD 09/2023]

Format:	**PER.<format>.view** *<file>*

Same as **PER.view** for converted peripheral files.

PER.IMPORT.AccessClass TRACE32 access class

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.AccessClass** *<class>*

Specifies the TRACE32 specific **access class** to be used for the **BASE** and **GROUP** commands.

Default: :ad

See also

■ PER.IMPORT

PER.IMPORT.EnumDelimiter Delimiter for BITFLD items.

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.EnumDelimiter** *<delimiter>* [*<description>*]

BITFLD items are usually separated by a comma. In order to change the separating character, the first argument must be used. The second (optional) argument can used to provide a description (tooltip) for each item.

delimiter	Character which separates BITFLD items. Default: ,
description	Character which separates BITFLD item from corresponding description. Default: none

PER.IMPORT.FieldsFromDescription Generate BITFLDs from description

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.FieldsFromDescription** [**ON** | **OFF**]

Tries to extract choice items for **BITFLD** commands from bitfield descriptions.If no choice items can be extracted, a **HEXMASK** will be generated instead.

Default: ON

See also

■ PER.IMPORT

PER.IMPORT.ForMaT — Input file format

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.ForMaT** *<format>*
<format>:	**AUTO** **SPIRITXML** **TIXML** **SVD**

Tells TRACE32 the format of the input files.

AUTO	Detect format automatically by means of the input file(s).
SPIRITXML	XML format used by IP-XACT.
TIXML	XML format used by Texas Instruments.
SVD	System View Description format for the Common Microcontroller Software Interface Standard.

Default: AUTO

See also

■ PER.IMPORT

PER.IMPORT.INDent — Indent trees, registers and fields

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.INDent** [**ON** | **OFF**]

Indent trees, registers and fields for improved readability of the resulting .per file.

Default: OFF

PER.IMPORT.InputFile — Input files for conversion

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.InputFile** *<file_list>*

Selects input files to be converted into a single .per file.

<file_list>	List of input files separated by whitespaces.

See also

■ PER.IMPORT

PER.IMPORT.LoaD — Load external converter project

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.LoaD** *<file>*

For backward compability only.

Allows to load project files from the previous external converters. Current internal converters store project files as PRACTICE .cmm scripts. See **PER.IMPORT.STOre**.

See also

■ PER.IMPORT

PER.IMPORT.LOGfile — Create logfile of conversion

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.LOGfile** [**ON** | **OFF**]

A logfile with extended information and error messages will be created during the conversion process. The logfile will be placed in the same directory as the output file (see **PER.IMPORT.OutputFile**).

Default: OFF

PER.IMPORT.MaximumChoiceLength — Maximum choice item length

[build 155354 - DVD 09/2023]

Format: **PER.IMPORT.MaximumChoiceLength** *<length>*

Defines the maximum length of the individual choice items in **BITFLD** commands. Must be in range 1..80.

Default: 50

See also

■ PER.IMPORT

PER.IMPORT.MaximumDescriptionLength — Maximum tooltip length

[build 155354 - DVD 09/2023]

Format: **PER.IMPORT.MaximumDescriptionLength** *<length>*

Defines the maximum length of the description/tooltip. Must be in range 1..255.

Default: 255

See also

■ PER.IMPORT

PER.IMPORT.MergeGroups — Minimize number of GROUPs

[build 155354 - DVD 09/2023]

Format: **PER.IMPORT.MergeGroups** [**ON** | **OFF**]

Merges consecutive registers (**LINE**) into a single **GROUP**. Otherwise each LINE will have its own GROUP.

Default: ON

See also

■ PER.IMPORT

PER.IMPORT.ModuleFiles   Split .per file into seperate files

[build 155354 - DVD 09/2023]

Format: **PER.IMPORT.ModuleFiles** [**ON** | **OFF**]

Instead of a single .per file, a .ph file for each module will be created. This is useful if you want to build up your own peripheral file library and want to re-use module files.

Default: OFF

See also

■ PER.IMPORT

PER.IMPORT.MSBfirst   Order of bits in BITFLD command

[build 155354 - DVD 09/2023]

Format: **PER.IMPORT.MSBfirst** [**ON** | **OFF**]

If **ON**, **BITFLD** commands will output the most significant bit first. Otherwise the most significant bit will be output last.

Default: ON

See also

■ PER.IMPORT

PER.IMPORT.NumberOfColumns Number of output columns

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.NumberOfColumns** *<number>*
<number>:	**AUTO** **1** **2** **3** **4** **5** **6**

Defines the number of output columns in the PER.<format>.view window. In case of **AUTO**, the algorithm tries to find the optimal number of columns.

Default: AUTO

See also

■ PER.IMPORT

PER.IMPORT.OutputFile Name of generated peripheral file

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.OutputFile** *<file>*

Name of the resulting .per file after conversion.

<file>	Output file name. If the file already exists, its content will be replaced.If **PER.IMPORT.InputFile** specifies only one input file, the file name will be taken over and its extension replaced by .per.

See also

■ PER.IMPORT

PER.IMPORT.REPeat — Generate REPEAT commands

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.REPeat** [**ON** | **OFF**]

Tries to find repetitive elements in the input file(s) and merges them into a **REPEAT** command. **However this command only works for registers and larger elements, but not for bitfields!** Bitfield names will always be a result of the first iteration of the corresponding **REPEAT** command.

Default: OFF

See also

■ PER.IMPORT

PER.IMPORT.RESet — Reset import settings

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.RESet**

Reset all PER.IMPORT settings to their defaults.

See also

■ PER.IMPORT

PER.IMPORT.RULES — Apply rules file

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.RULES** *<file>*

Apply rules file. See **“Rules file”** in Peripheral Files Programming, page 80 (per_prog.pdf).

See also

■ PER.IMPORT

PER.IMPORT.SortSubTrees — Sort TREEs alphabetically

[build 161055 - DVD 09/2023]

Format:	**PER.IMPORT.SortSubTrees** [**ON** | **OFF**]

If **ON**, all **TREE** levels except the first will be sorted alphabetically. If **OFF**, all **TREE** levels except the first will be output in the same order as they appear in the input file(s).

Default: ON.

PER.IMPORT.SortTopTrees — Sort TREEs alphabetically

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.SortTopTrees** [**ON** | **OFF**]

If **ON**, the first level of **TREE**s will be sorted alphabetically. If **OFF**, the first level of **TREE**s will be output in the same order as they appear in the input file(s).

Default: ON.

See also

■ PER.IMPORT

PER.IMPORT.STOre — Store current project

[build 155354 - DVD 09/2023]

Format:	**PER.IMPORT.STOre** *<file.cmm>*

Write all settings to a PRACTICE .cmm file.

See also

■ PER.IMPORT

Format:	**PER.IMPORT.WithValue** [**ON** | **OFF**]

If **ON**, each choice item of a **BITFLD** command will be preceded by its corresponding value and a colon:

```
<value>:<choice_item>

e.g.
1:enable
0:disable
```

Default:OFF

See also

■ PER.IMPORT

PER.In — Read port

Format: **PER.In** *<address>* [*<count>*] [**/***<options>*]

<options>: **Byte** | **Word** | **Long** | **Quad** | **TByte** | **PByte** | **HByte** | **SByte**
BE | **LE**
Repeat | **INCrement** | **CORE** *<core_number>*

This command reads data from the specified address and prints it to the message line. Please refer to the description of the **Data.In** command for more information.

See also

- PER
- PER.view

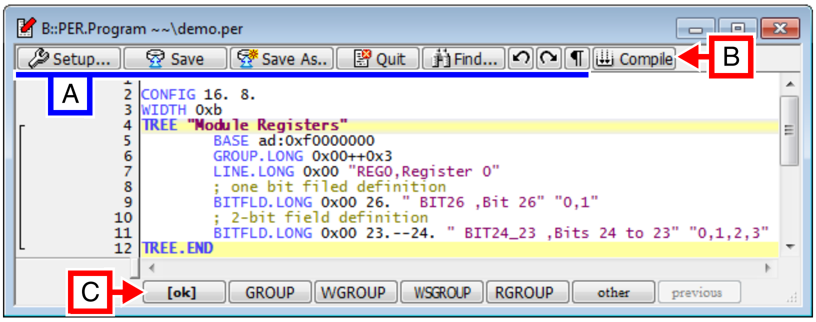
PER.Program — Interactive programming

Format: **PER.Program** [*<file>* [*<line>*]] [**/***<option>*]

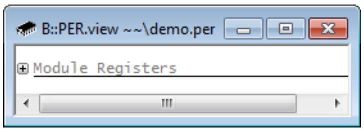
<option>: **AutoSave** | **NoSave**

Opens the **PER.Program** editor window, where you can create and edit peripheral files.

The editor provides an online syntax check. The input is guided by softkeys. For a description of the syntax for the peripheral files, refer to **"Peripheral Files Programming"** (per_prog.pdf).



```
CONFIG 16. 8.
WIDTH 0xb
TREE "Module Registers"
        BASE ad:0xf0000000
        GROUP.LONG 0x00++0x3
        LINE.LONG 0x00 "REG0,Register 0"
        ; one bit filed definition
        BITFLD.LONG 0x00 26. " BIT26 ,Bit 26" "0,1"
        ; 2-bit field definition
        BITFLD.LONG 0x00 23.--24. " BIT24_23 ,Bits 24 to 23" "0,1,2,3"
TREE.END
```



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 peripheral file (*.per) is error free, then the message “compiled successfully” is displayed in the **PER.Program** window.
To view the result, open the file in the **PER.view** window.

C Commands for programming peripheral files. For descriptions and examples, refer to **“Peripheral Files Programming Commands”** (per_prog.pdf).

<file>	The default extension for *<file>* is ***.per**.
<line>, *<option>*	For description of the arguments, see **EDIT.file**.

See also

■ PER	■ PER.ReProgram	■ PER.view	■ SETUP.EDITOR
❏ IOBASE()			

▲ ’Text Editors’ in ’PowerView User’s Guide’
▲ ’Release Information’ in ’Legacy Release History’

PER.ReProgram — Set default peripheral file

Format: **PER.ReProgram** [*<file>*]

Without command parameter <file>, the CPU specific default peripheral file (*.per) in the system directory is used (e.g. peromap35xx.per).

With command parameter <file>, the corresponding file is compiled. The file should not have any errors when using this command. This given file will be temporary used as new default peripheral file till the next **PER.ReProgram** command or a new start of TRACE32 software.

The peripherals can be displayed with the **PER.view** command without arguments.

See also

■ PER	■ PER.Program	■ PER.view	❏ IOBASE()

▲ ’Release Information’ in ’Legacy Release History’

Format:	**PER.ReProgramDECRYPT** [<*file*>]

Reprograms encrypted PER file. See **PER.ReProram** for more information.

See also

■ PER ■ PER.view

The **PER.Set** command group is used to modify peripheral registers.

See also

■ PER.Set.ByName	■ PER.Set.Field	■ PER.Set.Index	■ PER.Set.IndexField
■ PER.Set.Out	■ PER.Set.SaveIndex	■ PER.Set.SaveIndexField	■ PER.Set.SaveTIndex
■ PER.Set.SaveTIndexField	■ PER.Set.SEQuence	■ PER.Set.SEQuenceField	■ PER.Set.SHADOW
■ PER.Set.simple	■ PER.Set.TIndex	■ PER.Set.TIndexField	■ PER
■ PER.view			

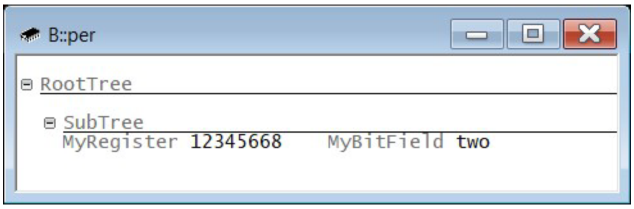
PER.Set.ByName Modify memory by name

[build 147535 - DVD 09/2022]

Format:	**PER.Set.ByName** *<path> <value>*|*<choice>*

A more convenient way to modify memory than **PER.Set.simple** and **PER.Set.Field**. The memory location can be referenced by its name rather than by its address. Also **HEXMASK** and **BITFLD** masks will be filled automatically.

<path>	(Full) path of the register(field) name. **Case sensitive!** Starting from the root tree, every subelement (**TREE**, **GROUP**, **LINE**, **BITFLD**, **HEXMASK**) must be separated by a dot. Alternatively the whole peripheral file can be searched for the register(field) name. In that case the name must be preceded by a dot. If a path elements contains spaces, it must be enclosed by quotes.
<value>	New value to be written.
<choice>	Choice item from **BITFLD**. **Case sensitive!** Only available if last item of <path> points to a **BITFLD**.



```
⊟ RootTree
  ⊟ SubTree
   MyRegister 12345668     MyBitField two
```

Example 1: Full path

```
; Select peripheral file by compiling it
PER.ReProgram permyperfile.per
; Set whole register
PER.Set.ByName RootTree.SubTree.MyRegister 0x12345668
; Set BITFLD only (Both methods do the same)
PER.Set.ByName RootTree.SubTree.MyRegister.MyBitField "two"
PER.Set.ByName RootTree.SubTree.MyRegister.MyBitField 2
```

Example 2: No path

```
; Select peripheral file by compiling it
PER.ReProgram permyperfile.per
; Set whole register
PER.Set.ByName .MyRegister 0x12345668
; Set BITFLD only (Both methods do the same)
PER.Set.ByName .MyRegister.MyBitField "two"
PER.Set.ByName .MyRegister.MyBitField 2
```

See also

■ PER.Set ❏ PER.ADDRESS() ❏ PER.VALUE() ❏ PER.VALUE.STRING()

PER.Set.Field — Modify a bit field in memory

Format:	**PER.Set.Field** *<address>* **%***<format>* *<mask>* [*<mult>* [*<summ>*]]*<value>*
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Modifies a bit field in memory. When some register content is shown in the Peripheral window by the **HEXMASK** or **BITFLD** command, it may be scaled with a multiplier and a summand. This command can be used to modify the scaled value without having to unscale it manually or taking care of the bitfield's offset.

The memory content at *<address>* is read with the access width given by *<format>*. The bits set in *<mask>* will be replaced by the corresponding bits in *<value>* and the new value is written to *<address>*. *<value>* is considered to be completely within the mask, one must not specify any offset to the mask.

```
OldData:      0x53674210  0y0101.0011.0110.0111.0100.0010.0001.0000
mask:         0x007c0000  0y0000.0000.0111.1100.0000.0000.0000.0000
                                       --- --|   <-  offset  ->   |
value:        0x5         0y           001 01
                          -----------------------------------------
NewData:      0x53174210  0y0101.0011.0001.0111.0100.0010.0001.0000
                  --                   --- --

NewData = (OldData & ~mask) | ( (value<<offset(mask)) & mask)
```

Additionally a possible multiplier *<mult>* may be specified as divisor. If the *<mult>* is omitted, the default is 1. Also a possible summand *<summ>* can be specified as subtrahend. If the *<summ>* is omitted, the default is 0. If *<summ>* and *<mult>* both specified, the division is performed before the subtraction.

```
tmpvalue = (<value> / <mult>) - <summ>;
tmpvalue = tmpvalue << (number of bits between <mask> and 0);
Memory(<address>) = (Memory(<address>) & <mask>) | tmpvalue;
```

Example 1 - the following PER file is given:

```
GROUP D:0xBF000000++3 "Cache Configuration"
LINE.LONG 0 "CACHE"
HEXMASK.LONG 0x0 8.--9. 64. 0. "Cache Size "

; Bits [9:8] are defined:  0 =   0 K Cache Size, displayed is 0x00
;                          1 =  64 K Cache Size, displayed is 0x40
;                          2 = 128 K Cache Size, displayed is 0x80
;                          3 = 172 K Cache Size, displayed is 0xC0
```

To change the cache size to 128 KB, perform the following command:

```
PER.Set.Field D:0xBF000000 %Long 0x00000300 64. 0. 128.
```

As result, the content of bits [9:8] is 0y10 (0x2).

Example 2 - Change single bit only and leave other bits untouched:

```
PER.Set.Field D:0xF0000470 %Long 0x00002000 1.    ; set bit 13
PER.Set.Field D:0xF0000470 %Long 0x01000000 0.    ; clear bit 24
```

See also

- PER.Set

Format:	**PER.Set.Index** *<idx_addr>* **%***<idx_fmt> <idx_rd> <idx_wr> <data_addr>* **%***<data_fmt> <data_value>*
<idx_fmt>, *<data_fmt>*:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Writes or modifies indirectly addressed registers.

<idx_addr>	Specifies the address register.
<data_addr>	Specifies the address of the data register of the indirect access.

PER.Set.Index can be translated into the following commands (IS_BITMASK and APPLY_BITMASK are pseudo-functions):

```
if IS_BITMASK(<data_value>)
(
  PER.Set <index_addr> %<idx_fmt>  <idx_rd>
  &read_value=DATA.<data_fmt>(<data_addr>)
  &new_value=APPLY_BITMASK(&read_value,<data_value>)
)
else
(
  &new_value=<data_value>
)
PER.Set <index_addr> %<idx_fmt>  <idx_wr>
PER.Set <data_addr>  %<data_fmt> &new_value
```

If the address register *<idx_addr>* is read/write, it is recommended to use **PER.Set.SaveIndex**, to restore the original setting after the access.

See also

■ PER.Set

PER.Set.IndexField — Set fields at indexed register

Format:	**PER.Set.IndexField** *<idx_addr>* **%***<idx_fmt> <idx_rd> <idx_wr> <data_addr>* **%***<data_fmt> <data_value>*
<idx_fmt>, *<data_fmt>*:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

See also

■ PER.Set

PER.Set.Out — Write data stream to memory

Format:	**PER.Set.Out** *<address>* **%***<format> <data> <string>* [**/***<option>*]
<options>:	**Repeat** | **CORE** *<core>*

Writes a sequence of data elements sequentially to *<address>*.

See also

■ PER.Set

Format:	**PER.Set.SaveIndex** *<idx_addr>* **%***<idx_fmt> <idx_rd> <idx_wr> <data_addr>* **%***<data_fmt> <data_value>*
<idx_fmt>, *<data_fmt>*:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Writes or modifies indirectly addressed registers.

<idx_addr>	Specifies the address register.
<data_addr>	Specifies the address if the data register of the indirect access.

The original value of the register at *<idx_addr>* is restored after the access.

PER.Set.SaveIndex can be translated into following commands (IS_BITMASK and APPLY_BITMASK are pseudo-functions):

```
&original_idx_addr=DATA.<idx_fmt>(<index_addr>)

if IS_BITMASK(<data_value>)
(
  PER.Set <index_addr> %<idx_fmt>  <idx_rd>
  &read_value=DATA.<data_fmt>(<data_addr>)
  &new_value=APPLY_BITMASK(&read_value,<data_value>)
)
else
(
  &new_value=<data_value>
)
PER.Set <index_addr> %<idx_fmt>  <idx_wr>
PER.Set <data_addr>  %<data_fmt> &new_value

PER.Set <index_addr> %<idx_fmt>  &original_idx_addr
```

If the address register *<idx_addr>* cannot be read (write only), use **“PER.Set.Index Modify indirect (indexed) register”** (general_ref_p.pdf).

See also

- PER.Set

PER.Set.SaveIndexField — Set fields at indexed register

Format:	**PER.Set.SaveIndexField** *<idx_addr>* **%***<idx_fmt>* *<idx_rd>* *<idx_wr>* *<data_addr>* **%***<data_fmt>* *<data_value>*
<idx_fmt>, *<data_fmt>*:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

See also

- PER.Set

PER.Set.SaveTIndex — Set fields at indexed registers

Format:	**PER.Set.SaveTIndex** *<address>* **%***<format>* *<value>*
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Modifies fields at indexed registers.

See also

- PER.Set

PER.Set.SaveTIndexField — Set fields at indexed registers

Format:	**PER.Set.SaveTIndexField** *<address>* **%***<format>* *<value>*
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Modifies fields at indexed registers.

See also

- PER.Set

PER.Set.SEQuence — Set SGROUP members

Format:	**PER.Set.SEQuence** *<offset>* **%***<format>* *<data>* …
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

See also

■ PER.Set

PER.Set.SEQuenceField — Set SGROUP members

Format:	**PER.Set.SEQuenceField** *<offset>* **%***<format>* *<data>* …
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

See also

■ PER.Set

PER.Set.SHADOW — Modify data based on shadow RAM

Format:	**PER.Set.SHADOW** *<address1>* *<address2>* **%***<format>* *<data>* *<string>* [**/***<option>*]
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**
<options>:	**Verify** | **ComPare** | **DIFF** | **PlusVM** | **CORE** *<core>*

Modifies data as **PER.Set**, but modifies data both on *<address1>* and on *<address2>* in shadow RAM.

See also

■ PER.Set

PER.Set.simple — Modify registers/peripherals

Format:	**PER.Set.simple** *<address>* **%***<format>* *<value>* [**/***<option>*]
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**
<options>:	**Verify** | **ComPare** | **DIFF** | **PlusVM** | **CORE** *<core>*

Modifies configuration registers/onchip peripherals. The command usually appears in the command line after a double click on a register in the **PER.view** window. See **Data.Set** for details on how to modify memories.

See also

- PER.Set

PER.Set.TIndex — Set fields at indexed registers

Format:	**PER.Set.TIndex** *<address>* **%***<format>* *<value>*
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Modifies fields at indexed registers.

See also

- PER.Set

Format:	**PER.Set.TIndexField** *<address>* **%***<format>* *<value>*
<format>:	**Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte** | **BE** | **LE**

Modifies fields at indexed registers.

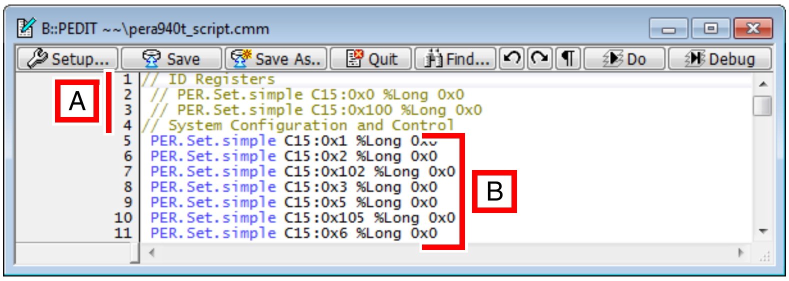
See also

■ PER.Set

[Examples]

Format:	**PER.STOre** [*<script_file>* [*<per_file>* ["*<subtree_path>*"]]] [**/CORE** *<core>*]

Stores all PER settings or all settings of a PER subtree to a PRACTICE file (*.cmm). The resulting file consists of **PER.Set.simple** commands [**B**]. If no *<script_file>* is specified, all settings are stored to the clipboard.



```
// ID Registers
 // PER.Set.simple C15:0x0 %Long 0x0
 // PER.Set.simple C15:0x100 %Long 0x0
// System Configuration and Control
 PER.Set.simple C15:0x1 %Long 0x0
 PER.Set.simple C15:0x2 %Long 0x0
 PER.Set.simple C15:0x102 %Long 0x0
 PER.Set.simple C15:0x3 %Long 0x0
 PER.Set.simple C15:0x5 %Long 0x0
 PER.Set.simple C15:0x105 %Long 0x0
 PER.Set.simple C15:0x6 %Long 0x0
```

A Headings and read-only PER file values are commented out in PRACTICE scripts generated by **PER.STOre**.

The command **PER.STOre** may result in a “bus errror” or “debug port fail” if TRACE32 has no access to a peripheral component. Possible reasons are:

- The component is disabled.
- The component has no power or clock.
- The access to the component is restricted.

The script generated by the **PER.STOre** command contains the **PER.Set** commands in the order the configuration registers appear in the **PER.view** window. If the script is used to initialize the target hardware, it is probably not possible to use the script without modifications. The configuration registers for peripheral components typically need to be initialized in a particular order, require sometimes a fixed timing, and often assume that other initializations have already been performed (e.g. clocks settings). So it is recommended to check the script and rearrange the **PER.Set** commands as required.

The script generated by the **PER.STOre** command can be directly used in the TRACE32 Instruction Set Simulator, e.g. to analyze a crash dump.

<script_file>	File name of the PRACTICE script generated upon execution of the **PER.STOre** command.
<per_file>	Name of the PER file that is used to describe the configuration registers. You can use a comma (,), if you want to use the default PER file for the core/chip under debug. The name of the default PER file is displayed in the **VERSION.SOFTWARE** window.

<subtree_path>	The optional parameter specifies the subtree to be saved. The individual components of a *<subtree_path>* are separated by comma.
CORE *<core>*	PER file values pertaining to the specified core (SMP debugging only).

Examples:

```
;generate script per_script.cmm for all settings
PER.STOre per_script.cmm
```

```
;generate script per_script.cmm for the settings
;of the subtree "Core Registers" and all its subtrees
;the name of the <per_file> is permpc564xbc.per
PER.STOre per_script.cmm permpc564xbc.per "Core Registers"
```

```
;generate script per_script.cmm for the settings
;of the subtree "Core Registers" and all its subtrees

;<per_file> can be represented by , if it is the default per file of the
;core/chip under debug
PER.STOre per_script.cmm , "Core Registers"
```

```
;generate script per_script.cmm for the settings
;of the specified subtree path
PER.STOre per_script.cmm , \
"Analog to Digital Converter,ADC0,Control Logic Registers"
```

```
;if no <script_file> is specified all settings are stored to the
;clipboard
PER.STOre
```

```
;only settings of the subtree "Core Registers" and all its subtrees
;are stored to the clipboard
PER.STOre ,, "Core Registers"
```

See also

■ PER ■ PER.view

▲ 'Release Information' in 'Legacy Release History'

PER.TestProgram — Test mode

Format:	**PER.TestProgram** [*<file>*]

Can be used to detects errors in per file.

See also

■ PER ■ PER.view

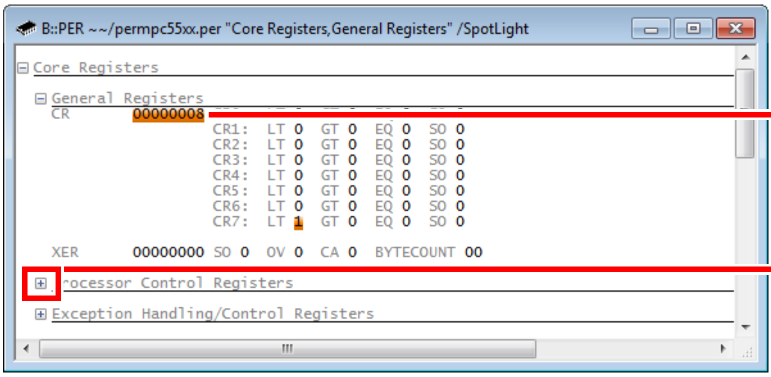
PER.view — Display peripherals

[Example]

Format:	**PER.view** [*<file>* [[*<args>*] "*<subtree_path>*"] [**/***<option>*]]
<option>:	**SpotLight** | **DualPort** | **Track** | **AlternatingBackGround** **CORE** *<core_number>*

Opens the **PER.view** window, displaying a so-called *PER file,* short for *peripheral register definition file*. PER files simplify working with peripheral registers and allow to display and modify the contents of peripheral registers. The peripheral registers in a PER file are often organized in a tree hierarchy.

Note that the **PER.view** window remains empty until the commands **SYStem.CPU** *<cpu_type>* and **SYStem.Mode Up** have been executed.



```
B::PER ~~/permpc55xx.per "Core Registers,General Registers" /SpotLight

⊟ Core Registers
  ⊟ General Registers
    CR       00000008
                      CR1:  LT 0  GT 0  EQ 0  SO 0
                      CR2:  LT 0  GT 0  EQ 0  SO 0
                      CR3:  LT 0  GT 0  EQ 0  SO 0
                      CR4:  LT 0  GT 0  EQ 0  SO 0
                      CR5:  LT 0  GT 0  EQ 0  SO 0
                      CR6:  LT 0  GT 0  EQ 0  SO 0
                      CR7:  LT 1  GT 0  EQ 0  SO 0

    XER      00000000 SO 0  OV 0  CA 0  BYTECOUNT 00
  ⊞ rocessor Control Registers
  ⊞ Exception Handling/Control Registers
```

The **SpotLight** option highlights changes.

Right-click to show/hide all subtrees.
Be sure to *show* all subtrees before searching for a specific item (e.g. with **Ctrl**+**F**).

NOTE:	For searching inside a (potentially huge) PER file, proceed as follows: • Right-click on a [-] or [+] box of the tree. • Choose **show all** from the popup menu. This will open all the subtrees. • Press **Ctrl**+**F** to open a search dialog for performing a text search in the open window and enter the term to search for.

<file>	Specifies the PER file to be displayed. If *<file>* is omitted, the default PER file for the selected CPU is displayed.
<subtree_path>	The optional parameter specifies the subtree to be opened. The individual components of a *<subtree_path>* are comma-separated. If *<subtree_path>* starts with a colon, only the selected subtree will be displayed. All others will be completely discarded.
<args>	Arguments can be passed from a PRACTICE script file (*.cmm) to a PER file. For an example, see **"Passing Arguments"** (per_prog.pdf).
SpotLight	Highlights all changes on the registers. Registers changed by the last program run/single step are marked in dark red. Registers changed by the second to the last program run/single step are marked a little bit lighter. This works up to 4 levels.
DualPort	Updates the registers while the program execution is running.
CORE *<n>*	Displays the contents of the registers for a certain core other than the currently selected core.
Track	All windows opened with the **/Track** option follow the cursor movements in the active window. For more information, see **"Window Tracking"** (ide_user.pdf).
AlternatingBack-Ground	Displays an alternating background color in the **PER.View** window. The background color display can also be toggled using the pop-up context menu entry "Toggle alternating background". This option is supported by TRACE32 release 09.2020 or newer.

Example: This script illustrates how you can use the **PER.view** command. Simply copy the script to a `test.cmm` file, and then step through the script (See "**How to...**").

```
;Displays the default PER definition file for the selected CPU, i.e.
;the peripherals for the selected CPU
PER.view

;Displays the path and the version of the PER definition file
VERSION.SOFTWARE

;The comma replaces the default PER definition file name
;and lets you use the SpotLight option.
PER.view , /SpotLight  ;This is useful to highlight changes

;Displays a specific PER definition file. The path prefix ~~ expands to
;the system directory of TRACE32
PER.view ~~/per750mm.per

;Expands all subtrees
PER.view ~~/per750mm.per "*"

;Expands just the subtree "General Registers"
PER ~~/permpc55xx.per "Core Registers,General Registers" /SpotLight
WinPAN 0. -3. ;The WinPAN command is used here for demo purposes.

;Expands all subtrees of "Core Registers"
PER.view , "Core Registers,*"
```

See also

■ PER	■ PER.IMPORT	■ PER.In	■ PER.Program
■ PER.ReProgram	■ PER.ReProgramDECRYPT	■ PER.Set	■ PER.STOre
■ PER.TestProgram	■ PER.viewDECRYPT	■ SYStem.CPU	❏ PER.ARG()

▲ 'Release Information' in 'Legacy Release History'

PER.viewDECRYPT View decrypted PER file in a PER window

Format:	**PER.viewDECRYPT** *<keystring>* *<file>* [*<string>* | *<address>*] [**/***<option>*]
<option>:	**SpotLight** | **DualPort** | **Track** | **AlternatingBackGround** **CORE** *<core_number>*

Encrypted PER files can be *executed* and viewed with the command **PER.viewDECRYPT** using the original *<keystring>*. Decrypting the PER file or viewing its original file contents in plain text is not possible.

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

See also

■ PER ■ PER.view ■ ENCRYPTPER

▲ 'Encrypt/Execute Encrypted Files' in 'PowerView User's Guide'

Programming Commands

For a description of the programming commands for peripheral files, refer to **“Peripheral Files Programming”** (per_prog.pdf).

PERF Sample-based profiling

See also

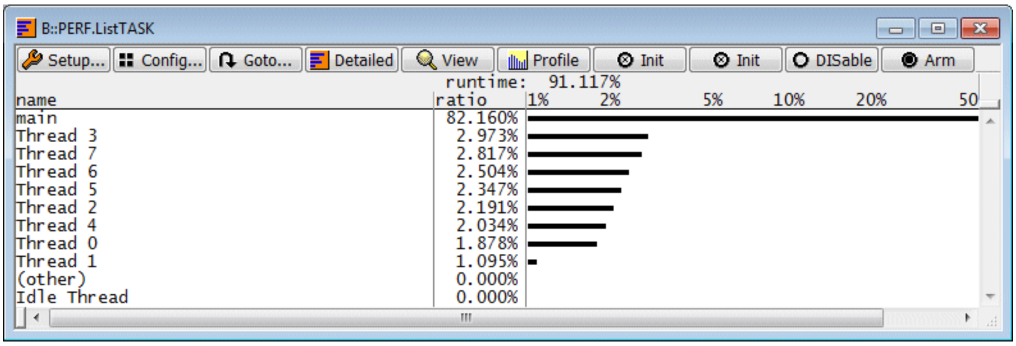
■ PERF.ADDRESS	■ PERF.Arm	■ PERF.AutoArm	■ PERF.AutoInit
■ PERF.ContextID	■ PERF.DISable	■ PERF.Init	■ PERF.List
■ PERF.ListDistriB	■ PERF.ListFunc	■ PERF.ListFuncMod	■ PERF.ListLABEL
■ PERF.ListLine	■ PERF.ListModule	■ PERF.ListProgram	■ PERF.ListRange
■ PERF.ListS10	■ PERF.ListTASK	■ PERF.ListTREE	■ PERF.ListVarState
■ PERF.LOAD	■ PERF.METHOD	■ PERF.MMUSPACES	■ PERF.Mode
■ PERF.OFF	■ PERF.PROfile	■ PERF.Program	■ PERF.ReProgram
■ PERF.RESet	■ PERF.RunTime	■ PERF.SAVE	■ PERF.SnoopAddress
■ PERF.SnoopMASK	■ PERF.SnoopSize	■ PERF.Sort	■ PERF.state
■ PERF.STREAM	■ PERF.ToProgram	■ PERF.View	❏ PERF.METHOD()
❏ PERF.MODE()	❏ PERF.RATE()	❏ PERF.RunTime()	❏ PERF.STATE()

▲ 'Release Information' in 'Legacy Release History'

Overview PERF

The TRACE32 Performance Analyzer is designed for sample-based profiling. Samples can be the actual program counter or the actual contents of a memory location. Sample-based profiling collects samples to calculate:

- The percentage of run-time used by a high-level language function.
- The percentage of run-time a variable had a certain contents.
- The percentage of run-time used by a task etc.



```
                                runtime:  91.117%
name                           ratio    1%     2%      5%     10%     20%     50
main                           82.160%
Thread 3                        2.973%
Thread 7                        2.817%
Thread 6                        2.504%
Thread 5                        2.347%
Thread 2                        2.191%
Thread 4                        2.034%
Thread 0                        1.878%
Thread 1                        1.095%
(other)                         0.000%
Idle Thread                     0.000%
```

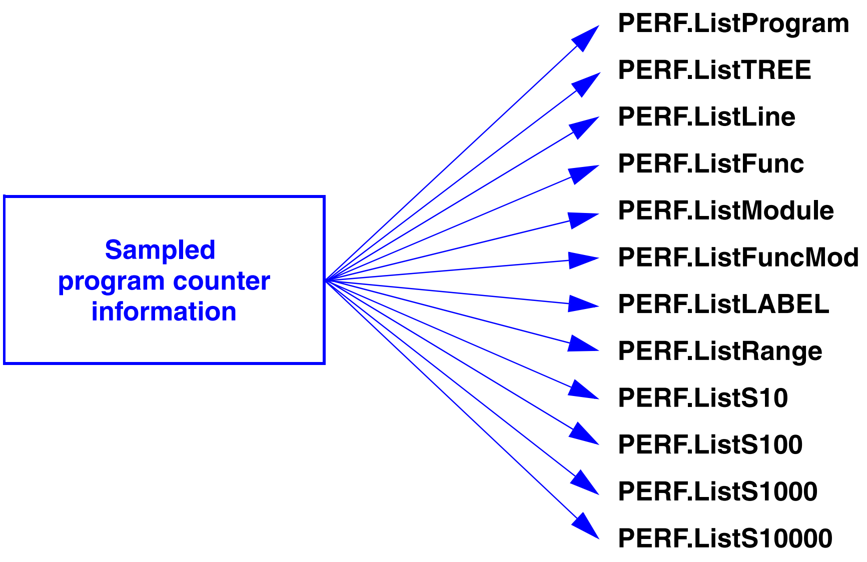
Samples are collected periodically. TRACE32 starts normally with 100 samples/s, but the sample acquisition methods of TRACE32 are auto-adaptive. They tune the sampling rate to its optimum.

TRACE32 supports several sample acquisition methods. Some have no or nearly no effect on the target's run-time behavior but require special features from the on-chip debug logic (Snoop, Trace, DCC). The acquisition method **StopAndGo** is always supported, but has some impact on the target's run-time behavior.

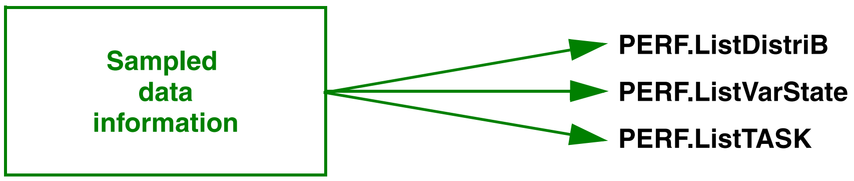
NOTE:	An unfavorable time coherence between the Performance Analyzer's sampling rate and periodic conditions on the target can distort the measurement results.

Profiling Results

The following evaluation commands can be used if the program counter is sampled:



The following evaluation commands can be used if the contents of a memory location is sampled:



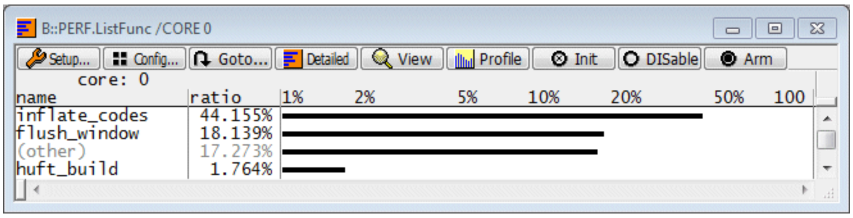
If a combi-mode is selected e.g. **PERF.Mode PCMEMory** the results can only be displayed independently.

```
PERF.state                              ; display the Performance
                                        ; Analyzer configuration window

PERF.RESet                              ; reset the Performance Analyzer
                                        ; configuration to its default
                                        ; setting

PERF.OFF                                ; enable the Performance
                                        ; Analyzer

PERF.Mode PCMEMory                      ; the Performance Analyzer
                                        ; samples the program counter
                                        ; and the contents of the
                                        ; specified memory location

;PERF.METHOD StopAndGo                  ; TRACE32 set the acquisition
                                        ; method StopAndGo

PERF.SnoopAddress Var.RANGE(flags[3])   ; specify the memory location to
                                        ; to be sampled

PERF.SnoopSize Byte                     ; specify the sampling width

PERF.ListFunc                           ; open a function profiling
                                        ; window

PERF.ListVarState                       ; and a separate variable state
                                        ; profiling window

Go                                      ; start the program execution
                                        ; and the sampling
```

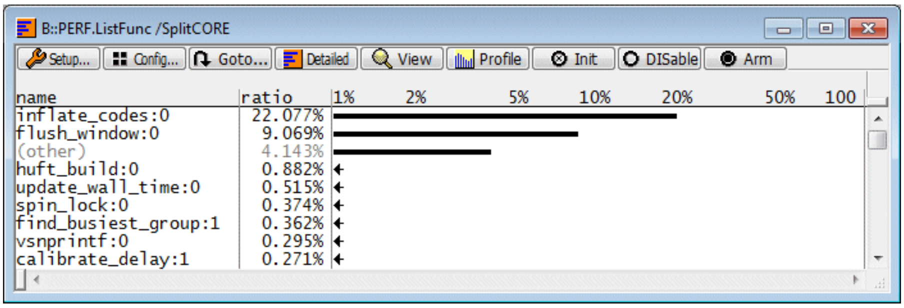
Profiling for SMP Systems

TRACE32 allows a sample-based profiling of SMP systems by supporting the **methods** **Snoop** and **StopAndGo**.

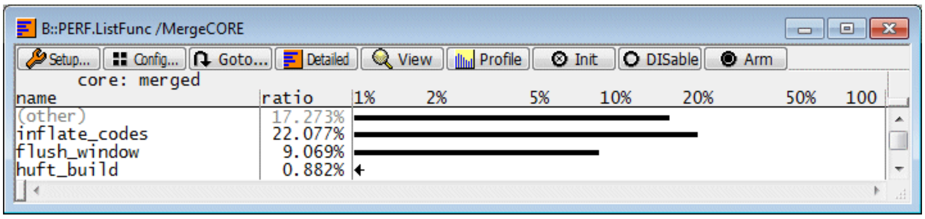
Function Profiling



```
      core: 0
name           ratio    1%     2%      5%     10%     20%      50%   100
inflate_codes   44.155%
flush_window    18.139%
(other)         17.273%
huft_build       1.764%
```



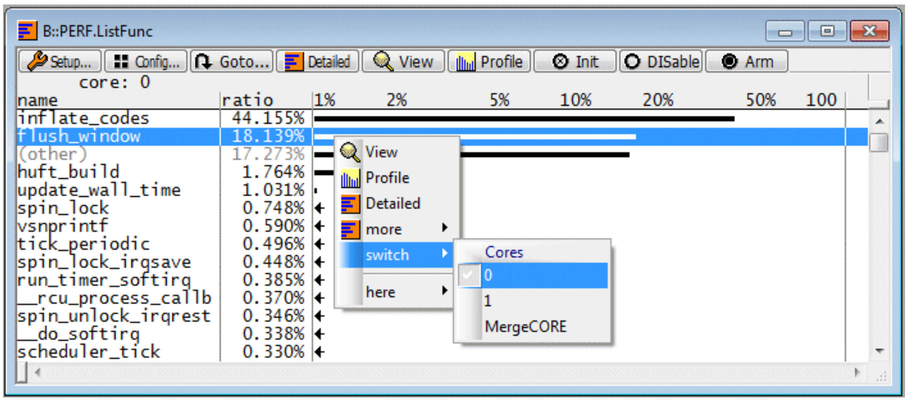
```
name                    ratio    1%     2%      5%     10%     20%      50%   100
inflate_codes:0         22.077%
flush_window:0           9.069%
(other)                  4.143%
huft_build:0             0.882%
update_wall_time:0       0.515%
spin_lock:0              0.374%
find_busiest_group:1     0.362%
vsnprintf:0              0.295%
calibrate_delay:1        0.271%
```



```
      core: merged
name                ratio    1%     2%      5%     10%     20%      50%   100
(other)             17.273%
inflate_codes       22.077%
flush_window         9.069%
huft_build           0.882%
```

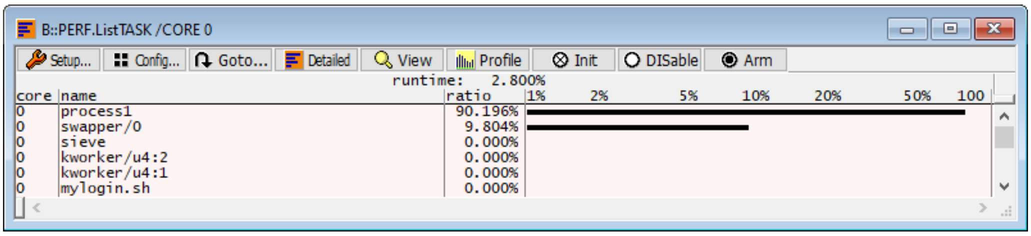
```
PERF.state                                ; display the Performance Analyzer
                                          ; configuration window

PERF.RESet                                ; reset the Performance Analyzer
                                          ; configuration window to its
                                          ; default settings

PERF.OFF                                  ; enable the Performance Analyzer

PERF.Mode PC                              ; the Performance Analyzer sample
                                          ; the actual program counter

;PERF.METHOD Snoop                        ; TRACE32 set the METHOD Snoop if
                                          ; the program counter can be read
                                          ; while the program execution is
                                          ; running

PERF.ListFunc /CORE 0                     ; open window for function
                                          ; profiling for core 0

…

PERF.ListFunc /SplitCORE                  ; open window for function
                                          ; profiling for all cores

                                          ; display results for each
                                          ; individual core

PERF.ListFunc /MergeCORE                  ; open window for function
                                          ; profiling for all cores

                                          ; results are added up for all
                                          ; cores
```

The result display can also be configured by the local pull-down menu.

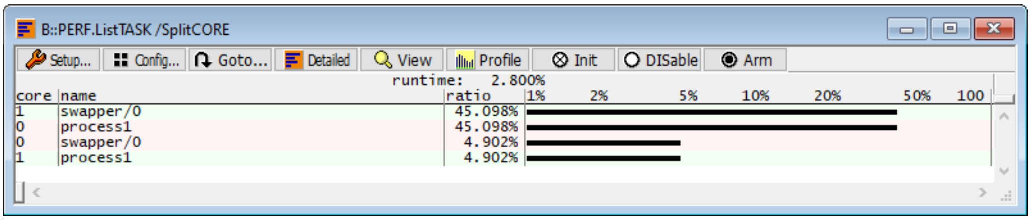


```
     core: 0
name              ratio    1%   2%    5%   10%   20%    50%  100
inflate_codes     44.155%
flush_window      18.139%
(other)           17.273%
huft_build         1.764%
update_wall_time   1.031%
spin_lock          0.748%
vsnprintf          0.590%
tick_periodic      0.496%
spin_lock_irqsave  0.448%
run_timer_softirq  0.385%
__rcu_process_callb 0.370%
spin_unlock_irqrest 0.346%
__do_softirq       0.338%
scheduler_tick     0.330%
```

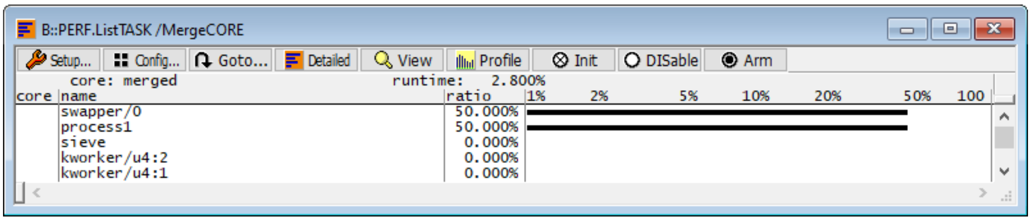
Task Profiling



```
                          runtime:   2.800%
core name                 ratio    1%    2%    5%   10%   20%   50%  100
0    process1             90.196%
0    swapper/0             9.804%
0    sieve                 0.000%
0    kworker/u4:2          0.000%
0    kworker/u4:1          0.000%
0    mylogin.sh            0.000%
```



```
                          runtime:   2.800%
core name                 ratio    1%    2%    5%   10%   20%   50%  100
1    swapper/0            45.098%
0    process1             45.098%
0    swapper/0             4.902%
1    process1              4.902%
```



```
     core: merged         runtime:   2.800%
core name                 ratio    1%    2%    5%   10%   20%   50%  100
     swapper/0            50.000%
     process1             50.000%
     sieve                 0.000%
     kworker/u4:2          0.000%
     kworker/u4:1          0.000%
```

```
PERF.state                          ; display the Performance Analyzer
                                    ; configuration window

PERF.RESet                          ; reset the Performance Analyzer
                                    ; configuration window to its
                                    ; default settings

PERF.OFF                            ; enable the Performance Analyzer

PERF.Mode TASK                      ; the Performance Analyzer sample
                                    ; the actual program counter
```

```
;PERF.METHOD Snoop                     ; TRACE32 set the METHOD Snoop if
                                       ; the memory can be read
                                       ; while the program execution is
                                       ; running

;TASK.CONFIG ....                      Setup OS-aware debugging

PERF.ListTASK /CORE 0                  ; open window for task profiling
                                       ; for core 0

…

PERF.ListTASK /SplitCORE               ; open window for TASK
                                       ; profiling for all cores

                                       ; display results for each
                                       ; individual core

PERF.ListTASK /MergeCORE               ; open window for TASK
                                       ; profiling for all cores

                                       ; results are added up for all
                                       ; cores
```

Format:	**PERF.ADDRESS** *<address>* | *<address_range>* (program counter sampling only)

Restricts the evaluation of the program counter sampling to *<address_range>*. A given *<address>* is expanded to an address range that ends at the next label. The default *<address_range>* is the whole address space of the processor.

The following commands are equivalent:

```
PERF.ADDRESS Var.RANGE(sieve)     PERF.ListFunc /Address Var.RANGE(sieve)
PERF.ListFunc
```

Example: In this script, the sample-based profiling is restricted to the function `sieve`.

```
PERF.state                        ; display the Performance Analyzer
                                  ; configuration window

PERF.RESet                        ; reset the Performance Analyzer
                                  ; configuration to its default settings

PERF.OFF                          ; enable the Performance Analyzer

PERF.Mode PC                      ; sample the program counter
                                  ; information

PERF.METHOD Trace                 ; set the acquisition method Trace

PERF.ADDRESS Var.RANGE(sieve)     ; restrict the evaluation of the
                                  ; result to the program range of the
                                  ; function sieve

PERF.ListLine                     ; open a window for the profiling of
                                  ; high-level language lines

Go                                ; start the program execution and the
                                  ; sampling
```

See also

- PERF
- PERF.state

PERF.Arm — Activate the performance analyzer manually

Format:	**PERF.Arm**

The Performance Analyzer is coupled to the program execution if **PERF.AutoArm** is **ON** (default).

If **PERF.AutoArm** is **OFF**, the Performance Analyzer can be controlled manually. **PERF.Arm** activates the Performance Analyzer, **PERF.OFF** stops the Performance Analyzer.

See also

■ PERF ■ PERF.state

PERF.AutoArm — Couple performance analyzer to program execution

Format:	**PERF.AutoArm** [**ON** | **OFF**]

The Performance Analyzer is coupled to the program execution.

ON (default)	The Performance Analyzer starts sampling when the program execution is started and stops when the program execution is stopped.
OFF	The Performance Analyzer has to be started and stopped manually by the commands **PERF.Arm** and **PERF.OFF**.

See also

■ PERF ■ PERF.state

PERF.AutoInit — Automatic initialization

Format:	**PERF.AutoInit** [**ON** | **OFF**]

The **PERF.Init** command will be executed automatically, when the user program is started.

See also

■ PERF ■ PERF.state

PERF.ContextID — Enable sampling the context ID register

Format:	**PERF.ContextID** [**ON** | **OFF**]

When this option is enabled, the ARM ContextID register will be sampled with the program counter and used in the analysis for task identification. This option is only available for some ARM cores.

See also

■ PERF ■ PERF.state

PERF.DISable — Disable the performance analyzer

Format:	**PERF.DISable**

The Performance Analyzer is disabled. Enabling can be done by entering the commands **PERF.Arm** or **PERF.OFF**.

The measurement data are preserved until the Performance Analyzer is re-enabled.

See also

■ PERF ■ PERF.state

PERF.Init — Reset current measurement

Format:	**PERF.Init**

Resets the current measurement. **PERF.Init** does not affect the Performance Analyzer configuration.

See also

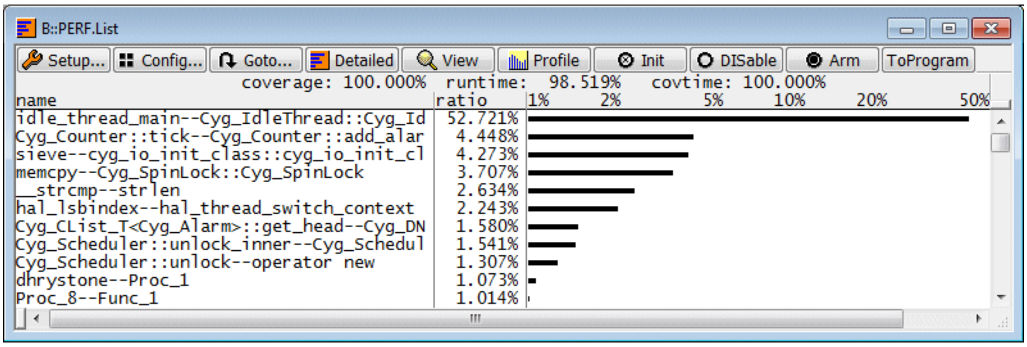
■ PERF ■ PERF.state

Format:	**PERF.List** [*<column>* …] [**/***<option>*]
<column>:	**DEFault** **DYNamic** **ALL** **Name** **TIme** **WatchTIme** **Ratio** **DRatio** **BAR** [**.log** | **.LIN**] **DBAR** [**.log** | **.LIN**] **Hits** **Address**
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Default profiling displays:

PERF.ListLabel	for **PERF.Mode PC** | **PCTASK** | **PCMEMory**
PERF.ListTASK	for **PERF.Mode TASK**
PERF.ListDistriB	for **PERF.Mode MEMory**

CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.



```
                              coverage: 100.000%  runtime:   98.519%   covtime: 100.000%
name                                     ratio    1%      2%         5%       10%       20%       50%
idle_thread_main--Cyg_IdleThread::Cyg_Id  52.721%
Cyg_Counter::tick--Cyg_Counter::add_alar   4.448%
sieve--cyg_io_init_class::cyg_io_init_cl   4.273%
memcpy--Cyg_SpinLock::Cyg_SpinLock         3.707%
__strcmp--strlen                           2.634%
hal_lsbindex--hal_thread_switch_context    2.243%
Cyg_CList_T<Cyg_Alarm>::get_head--Cyg_DN   1.580%
Cyg_Scheduler::unlock_inner--Cyg_Schedul   1.541%
Cyg_Scheduler::unlock--operator new        1.307%
dhrystone--Proc_1                          1.073%
Proc_8--Func_1                             1.014%
```

Interpretation of the result:

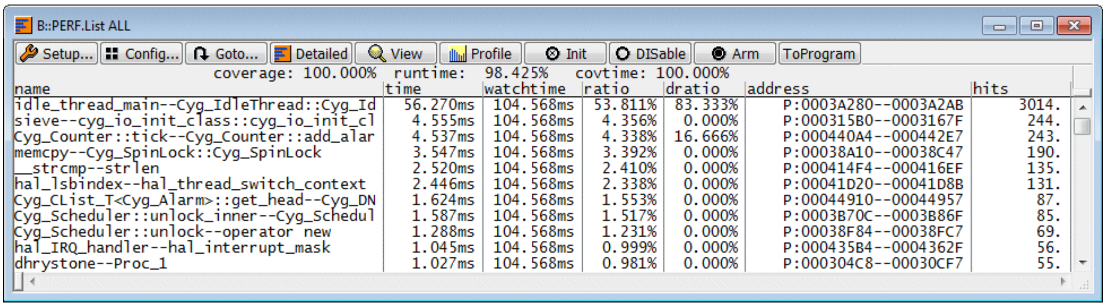
runtime	**PERF.METHOD StopAndGo** only: Percentage of time taken by the actual program run in the last second, the rest of the time was consumed by the measurement.

Columns sets:

DEFault	Select the standard set (columns: **Name**, **Ratio** and **BAR.log**). The DEFault configuration is also used if no display items are specified.
DYNamic	Displays the results of the last second (columns: **Name**, **DRatio**, **DBAR.log**). Dynamic displays are continually updated with the results of the previous second of performance data. They will not reflect any performance data outside the previous second.
ALL	Display all possible numeric fields in the **PERF.List** window (columns: **Name**, **Time**, **WatchTime**, **Ratio**, **DRatio**, **Address**, **Hits**).

```
PERF.List Hits DEFault           ; Open a PERF.List window starting with
                                 ; the column Hits followed by the
                                 ; default columns
```

```
PERF.List ALL
```



```
                      coverage: 100.000%  runtime:  98.425%  covtime: 100.000%
name                                  time      watchtime  ratio    dratio   address                       hits
idle_thread_main--Cyg_IdleThread::Cyg_Id  56.270ms  104.568ms  53.811%  83.333%  P:0003A280--0003A2AB  3014.
sieve--cyg_io_init_class::cyg_io_init_cl   4.555ms  104.568ms   4.356%   0.000%  P:000315B0--0003167F   244.
Cyg_Counter::tick--Cyg_Counter::add_alar   4.537ms  104.568ms   4.338%  16.666%  P:000440A4--000442E7   243.
memcpy--Cyg_SpinLock::Cyg_SpinLock         3.547ms  104.568ms   3.392%   0.000%  P:00038A10--00038C47   190.
__strcmp--strlen                           2.520ms  104.568ms   2.410%   0.000%  P:000414F4--000416EF   135.
hal_lsbindex--hal_thread_switch_context    2.446ms  104.568ms   2.338%   0.000%  P:00041D20--00041D8B   131.
Cyg_CList_T<Cyg_Alarm>::get_head--Cyg_DN   1.624ms  104.568ms   1.553%   0.000%  P:00044910--00044957    87.
Cyg_Scheduler::unlock_inner--Cyg_Schedul   1.587ms  104.568ms   1.517%   0.000%  P:0003B70C--0003B86F    85.
Cyg_Scheduler::unlock--operator new        1.288ms  104.568ms   1.231%   0.000%  P:00038F84--00038FC7    69.
hal_IRQ_handler--hal_interrupt_mask        1.045ms  104.568ms   0.999%   0.000%  P:000435B4--0004362F    56.
dhrystone--Proc_1                          1.027ms  104.568ms   0.981%   0.000%  P:000304C8--00030CF7    55.
```

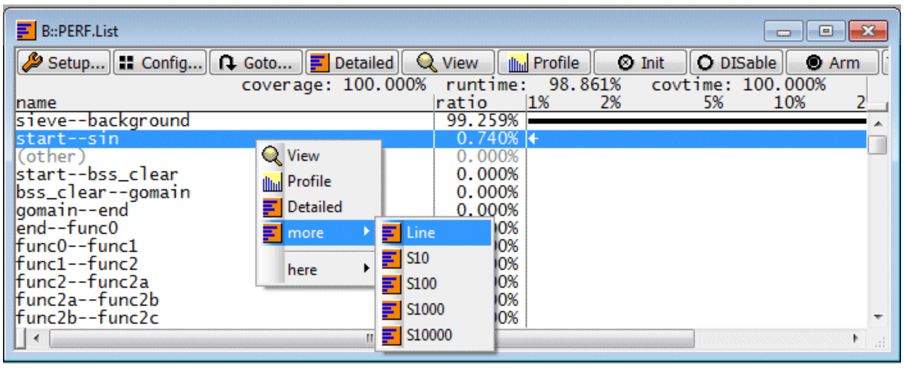
columns	
name	Name of the item (here label range)
time	Total run-time spent in item
watchtime	Observation time of item
ratio	Ratio of time spent by the item in percent
dratio	Ratio of time spent by item in the last second in percent. Please refer to **DYNamic** for more information.
address	Item´s address range or contents of the memory location
hits	Number of samples taken for the item
bar	Logarithmic bar for the ratio
dbar	Logarithmic bar for the ratio of time spent by item in the last second. Please refer to **DYNamic** for more information.

Column description:

Name	Display the names/contents of the listed items. Command **PERF.ListFunc**: If the sampled program counter can’t be assigned to a high-level language function (e.g. assembler code, library code) it is assigned to **(other)**. Command **PERF.ListLine**: If the sampled program counter can not be assigned to the address range of an high-level language line, it is assigned to **(other)** Command **PERF.ListTASK**: If task ID 0x0 is sampled or if the sampled task ID is unknown it is assigned to **(other)**.
TIme	Total runtime spent in listed item.

WatchTIme	Time the item is observed. This time will be the same for all ranges if the program counter is sampled. When the contents of a memory location is sampled, **WatchTime** starts when the listed value is detected the first time.
Ratio	Ratio of time spent by the listed item in percent. This value is calculated by dividing the field **TIme** by **WatchTIme.**
DRatio	Similar to **Ratio**, but only for the last second. Please refer to **DYNamic** for more information.
BAR	Display the profiling values in a graphical way as horizontal bars. The default display is logarithmic. The keyword **.LIN** changes to a linear display.
DBAR	Similar to **BAR**, but only for the last second. Please refer to **DYNamic** for more information.
Hits	Number of samples taken for the item.
Address	Item´s address range or contents of the memory location.

Buttons and Context Menu in the PERF.List window



```
                         coverage: 100.000%  runtime:  98.861%   covtime: 100.000%
name                                       ratio  1%     2%        5%      10%    2
sieve--background                        99.259%
start--sin                                0.740%
(other)                                   0.000%
start--bss_clear                          0.000%
bss_clear--gomain                         0.000%
gomain--end                               0.000%
end--func0                                   0%
func0--func1                                 0%
func1--func2                                 0%
func2--func2a                                0%
func2a--func2b                               0%
func2b--func2c                               0%
```

Buttons	
Setup …	Opens a **PERF.state** window that allows the configuration of the Performance Analyzer.
Config …	Opens a configuration dialog that allows to rearrange the column display in the **PERF.List** window.
Goto …	Opens a **Perf Goto** dialog which allows to bring the specified item in display (command line equivalent **Data.GOTO**).

Detailed	Opens a **PERF.List** window, which lists all numerical items (command line equivalent **PERF.List<item> ALL)**. Only supported for program counter sampling.
View	Opens a window to display all performance data of a selected item (command line equivalent **PERF.View /Track**).
Profile	Opens a **PERF.PROfile** window that displays a graphical profiling for the first three listed items, (other) is ignored.
Init	Execute the command **PERF.Init**. This command resets the current measurement. The Performance Analyzer configuration is not touched.
DISable	Disable the Performance Analyzer (command line equivalent **PERF.DISable**).
Arm	Activates the Performance Analyzer manually (command line equivalent **PERF.Arm**)
ToProgram	A Performance Analyzer program is generated out of the currently shown address ranges (program counter sampling only). The command line equivalent is **PERF.ToProgram**.

Context menu items	
View	This window displays all performance data for the selected line (command line equivalent **PERF.View** *<address>*).
Profile	Opens a **PERF.PROfile** window that displays a graphical profiling for the selected line.
Detailed	Opens a **PERF.List** window, which lists all numerical items (command line equivalent **PERF.List<item> ALL)**. Only supported for program counter sampling.
Line	Opens a **PERF.ListLine** window for the selected item (command line equivalent **PERF.ListLine /Address** *<range>*). Only supported for program counter sampling.
S10/S100/S1000/S10000	Opens a **PERF.ListSn** window for the selected item (command line equivalent **PERF.ListSn /Address** *<range>*). Only supported for program counter sampling.

Options	
Track	Tracks the window to the reference position of other windows.
Address *<range>* | *<address>*	Restricts the evaluation of the profiling results to the specified address range. If only an *<address>* is given it is expanded to an address range that ends at the next label. Only supported for program counter sampling.

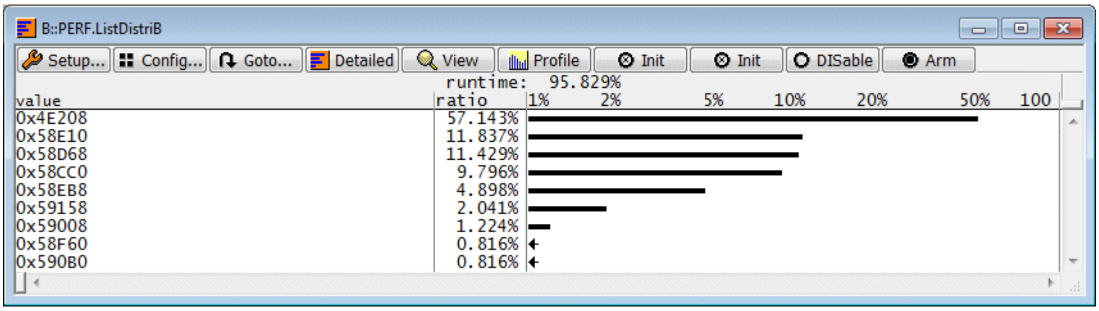
See also

■ PERF ■ PERF.state

▲ 'Release Information' in 'Legacy Release History'

Format: **PERF.ListDistriB** [*<column>* …] [**/Track**]
(memory contents sampling)

Reports the percentage of run-time a memory location had a certain value.



```
                         runtime:   95.829%
value                    ratio    1%   2%    5%   10%   20%   50%  100
0x4E208                  57.143%
0x58E10                  11.837%
0x58D68                  11.429%
0x58CC0                   9.796%
0x58EB8                   4.898%
0x59158                   2.041%
0x59008                   1.224%
0x58F60                   0.816%
0x590B0                   0.816%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

Example for ARM9:

```
PERF.state                          ; display the Performance Analyzer
                                    ; configuration window

PERF.RESet                          ; reset the Performance Analyzer
                                    ; configuration to its default
                                    ; setting

PERF.OFF                            ; enable the Performance Analyzer

PERF.Mode MEMory                    ; the Performance Analyzer samples
                                    ; the contents of a memory location

;PERF.METHOD StopAndGo              ; TRACE32 sets the acquisition
                                    ; method StopAndGo

PERF.SnoopAddress 0x4BD60           ; specify the memory location

PERF.SnoopSize Long                 ; specifies the sampling width

PERF.ListDistriB                    ; open a memory contents
                                    ; profiling window

Go                                  ; start the program execution and
                                    ; sampling
```

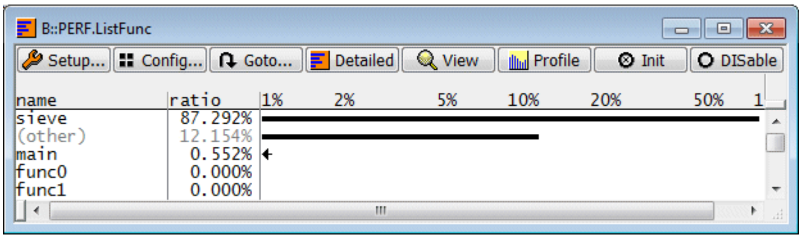
See also

■ PERF ■ PERF.state

Format:	**PERF.ListFunc** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time used by high-level language functions.

If the sample program counter can not be assigned to the address range of an HLL function, it is assigned to (other). The command **PERF.ListLABEL** can be used to get more information on what is assigned to (other).



```
name      ratio    1%   2%   5%   10%   20%   50%   1
sieve     87.292%
(other)   12.154%
main       0.552%
func0      0.000%
func1      0.000%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

CORE, **MergeCORE**, **SplitCORE**	For details, refer to **“Profiling for SMP Systems”**, page 48.

Example for ARM9:

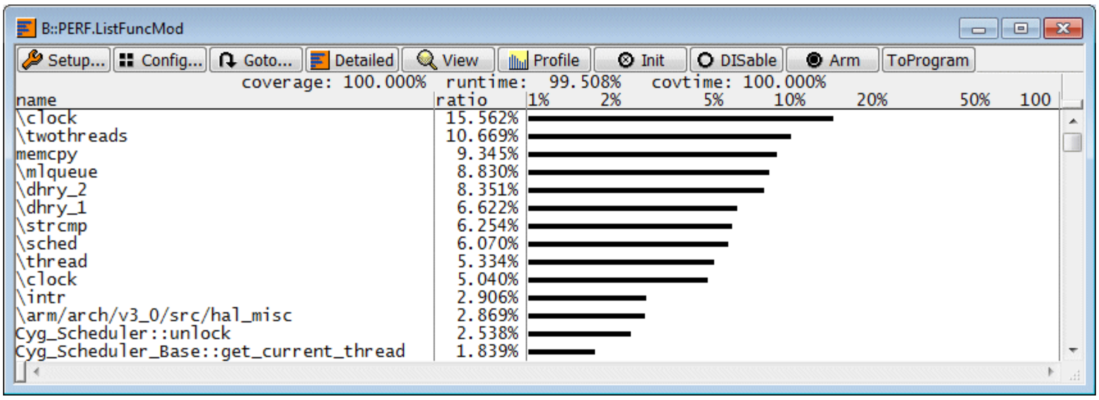
```
; example for ARM9

PERF.state                              ; display the Performance Analyzer
                                        ; configuration window

PERF.RESet                              ; reset the Performance Analyzer
                                        ; configuration to its default
                                        ; settings

PERF.OFF                                ; enable Performance Analyzer

PERF.Mode PC                            ; the Performance Analyzer samples
                                        ; the actual program counter

PERF.METHOD Trace                       ; set the acquisition method Trace

PERF.ListFunc                           ; open a window for function
                                        ; profiling

Go                                      ; start the program execution and
                                        ; sampling
```

See also

■ PERF ■ PERF.state

Format:	**PERF.ListFuncMod** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Report the percentage of run-time spent in high-level language functions inside the address range specified by the **PERF.ADDRESS** command. Outside the specified address range the percentage is reported on module base.



```
                     coverage: 100.000%  runtime:   99.508%   covtime: 100.000%
name                                      ratio    1%     2%       5%     10%      20%        50%   100
\clock                                    15.562%
\twothreads                               10.669%
memcpy                                     9.345%
\mlqueue                                   8.830%
\dhry_2                                    8.351%
\dhry_1                                    6.622%
\strcmp                                    6.254%
\sched                                     6.070%
\thread                                    5.334%
\clock                                     5.040%
\intr                                      2.906%
\arm/arch/v3_0/src/hal_misc                2.869%
Cyg_Scheduler::unlock                      2.538%
Cyg_Scheduler_Base::get_current_thread     1.839%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

CORE, **MergeCORE**, **SplitCORE**	For details, refer to **“Profiling for SMP Systems”**, page 48.

Example for ARM9:

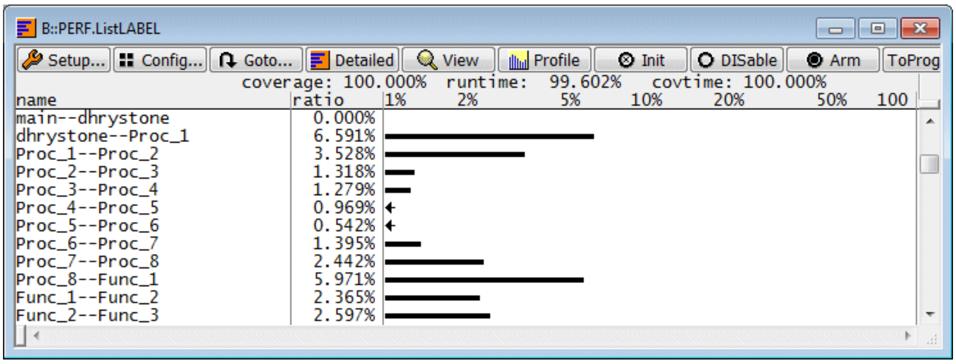
```
PERF.state                              ; display the Performance Analyzer
                                        ; configuration window

PERF.RESet                              ; reset the Performance Analyzer
                                        ; configuration to its default
                                        ; settings

PERF.OFF                                ; enable Performance Analyzer

PERF.Mode PC                            ; the Performance Analyzer samples
                                        ; the actual program counter

; PERF.METHOD StopAndGo                 ; TRACE32 sets the acquisition
                                        ; method StopAndGo

PERF.Mode PC                            ; the Performance Analyzer samples
                                        ; the actual program counter

PERF.ADDRESS 0x38000--0x38fff           ; specify address range

PERF.ListFuncMod                        ; display a function profiling
                                        ; inside the specified address
                                        ; range and module profiling
                                        ; outside the specified address
                                        ; range

Go                                      ; start the program execution and
                                        ; sampling
```

See also

■ PERF ■ PERF.state

Format:	**PERF.ListLABEL** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>*
	CORE *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in the address range between two labels.


B::PERF.ListLABEL
Setup... Config... Goto... Detailed View Profile Init DISable Arm ToProg

```
                              coverage: 100.000%  runtime:   99.602%   covtime: 100.000%
name                      ratio    1%     2%        5%     10%      20%        50%    100
main--dhrystone            0.000%
dhrystone--Proc_1          6.591%
Proc_1--Proc_2             3.528%
Proc_2--Proc_3             1.318%
Proc_3--Proc_4             1.279%
Proc_4--Proc_5             0.969%
Proc_5--Proc_6             0.542%
Proc_6--Proc_7             1.395%
Proc_7--Proc_8             2.442%
Proc_8--Func_1             5.971%
Func_1--Func_2             2.365%
Func_2--Func_3             2.597%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.

Example for ARM9:

```
PERF.state                              ; display the Performance Analyzer
                                        ; configuration window

PERF.RESet                              ; reset the Performance Analyzer
                                        ; configuration to its default
                                        ; settings

PERF.OFF                                ; enable Performance Analyzer

PERF.Mode PC                            ; the Performance Analyzer samples
                                        ; the actual program counter

; PERF.METHOD StopAndGo                 ; TRACE32 sets the acquisition
                                        ; method StopAndGo

PERF.Sort OFF                           ; the result is sorted by the
                                        ; succession of the labels in the
                                        ; symbol database

PERF.ListLABEL                          ; open a window for label-based
                                        ; profiling

Go                                      ; start the program execution and
                                        ; sampling
```

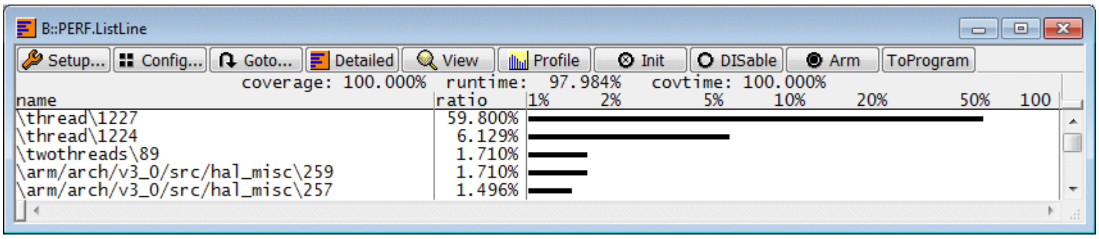
See also

- PERF
- PERF.state

Format:	**PERF.ListLine** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in high-level language lines.

If the sampled program counter cannot be assigned to the address range of an HLL line, it is assigned to (other). If the time spent in (others) is high the command **PERF.ListLABEL** can be used to get more information.



```
                              coverage: 100.000%  runtime:   97.984%   covtime: 100.000%
name                                          ratio     1%      2%         5%      10%       20%          50%    100
\thread\1227                                   59.800%
\thread\1224                                    6.129%
\twothreads\89                                  1.710%
\arm/arch/v3_0/src/hal_misc\259                 1.710%
\arm/arch/v3_0/src/hal_misc\257                 1.496%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

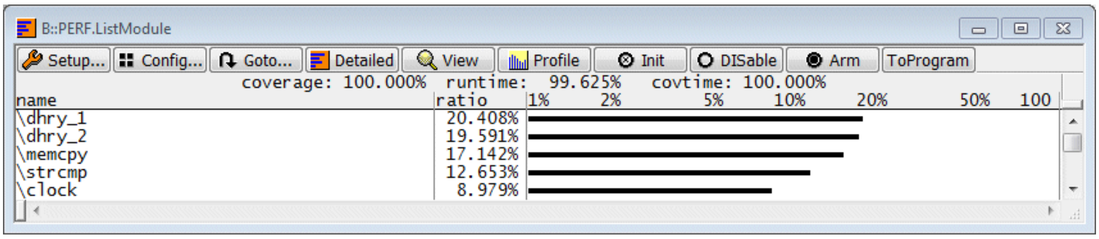
CORE, **MergeCORE**, **SplitCORE**	For details, refer to **“Profiling for SMP Systems”**, page 48.

See also

- PERF
- PERF.state

Format:	**PERF.ListModule** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in program modules.



```
                    coverage: 100.000%  runtime:   99.625%   covtime: 100.000%
name                                   ratio     1%      2%        5%     10%      20%      50%   100
\dhry_1                                 20.408%
\dhry_2                                 19.591%
\memcpy                                 17.142%
\strcmp                                 12.653%
\clock                                   8.979%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.

See also

- PERF
- PERF.state

Format:	**PERF.ListProgram** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in the address ranges specified by the Performance Analyzer program. A complete example of how to work with a Performance Analyzer program is given in the description of the **PERF.Program** command.

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.

See also

- PERF
- PERF.state

PERF.ListRange

Profiling by ranges

Format:	**PERF.ListRange** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in all ranges specified in the symbol database.

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

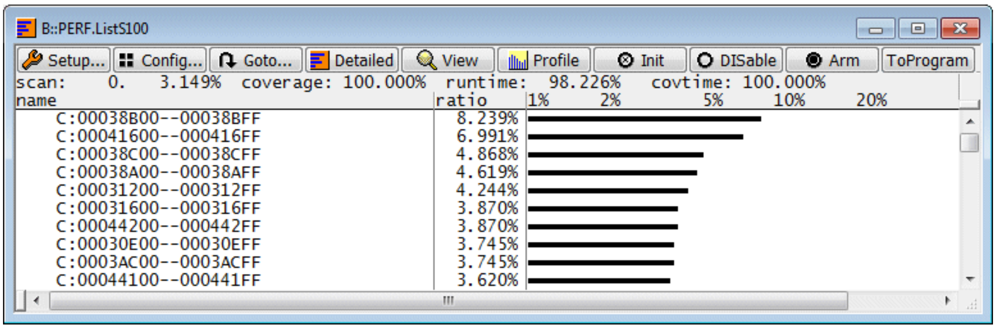
CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.

See also

- PERF
- PERF.state

Format:	**PERF.ListS10** [*<column>* …] [**/***<option>*] **PERF.ListS100** [*<column>* …] [**/***<option>*] **PERF.ListS1000** [*<column>* …] [**/***<option>*] **PERF.ListS10000** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in 16/256/4096/65536 byte segments.



```
scan:     0.    3.149%  coverage: 100.000%  runtime:   98.226%   covtime: 100.000%
name                                        ratio    1%      2%          5%     10%      20%
    C:00038B00--00038BFF                      8.239%
    C:00041600--000416FF                      6.991%
    C:00038C00--00038CFF                      4.868%
    C:00038A00--00038AFF                      4.619%
    C:00031200--000312FF                      4.244%
    C:00031600--000316FF                      3.870%
    C:00044200--000442FF                      3.870%
    C:00030E00--00030EFF                      3.745%
    C:0003AC00--0003ACFF                      3.745%
    C:00044100--000441FF                      3.620%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

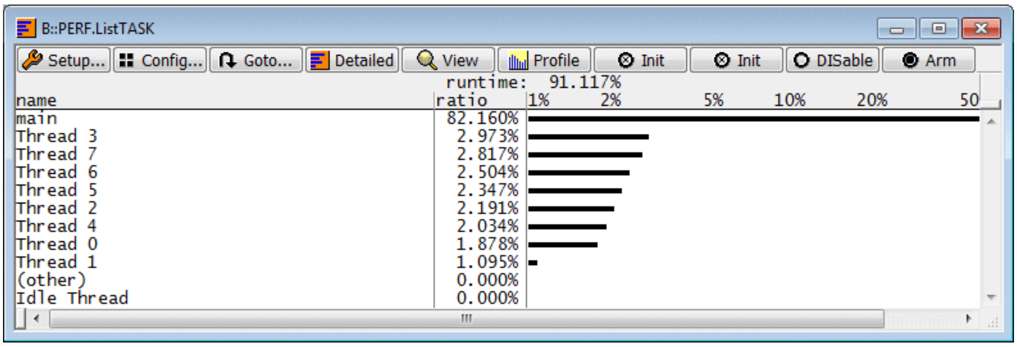
CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.

See also

- PERF
- PERF.state

Format:	**PERF.ListTASK** [*<column>* …] **[/Track]** (memory contents sampling)

Reports the percentage of run-time spent in different tasks/threads based on the sampling of the contents of the OS-variable that contains the identifier for the current task/tread.



```
                      runtime:  91.117%
name                  ratio    1%   2%   5%   10%   20%   50
main                  82.160%
Thread 3               2.973%
Thread 7               2.817%
Thread 6               2.504%
Thread 5               2.347%
Thread 2               2.191%
Thread 4               2.034%
Thread 0               1.878%
Thread 1               1.095%
(other)                0.000%
Idle Thread            0.000%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

Example for ARM9 and RTOS ECOS:

```
TASK.CONFIG ecos                ; enable ECOS-aware debugging

PERF.state                      ; display the Performance Analyzer
                                ; configuration window

PERF.RESet                      ; reset the Performance Analyzer
                                ; configuration to its default
                                ; settings

PERF.OFF                        ; enable Performance Analyzer

PERF.Mode TASK                  ; the Performance Analyzer samples
                                ; the contents of the variable that
                                ; contains the identifier for the
                                ; current task

; PERF.METHOD StopAndGo         ; TRACE32 sets the acquisition
                                ; method StopAndGo

PERF.Mode TASK                  ; the Performance Analyzer samples
                                ; data information from
                                ; TASK.CONFIG(magic)

PERF.ListTASK                   ; open a window to display a
                                ; a task profiling

Go                              ; start the program execution and
                                ; the sampling
```

Example for ARM9 and proprietary target-OS:

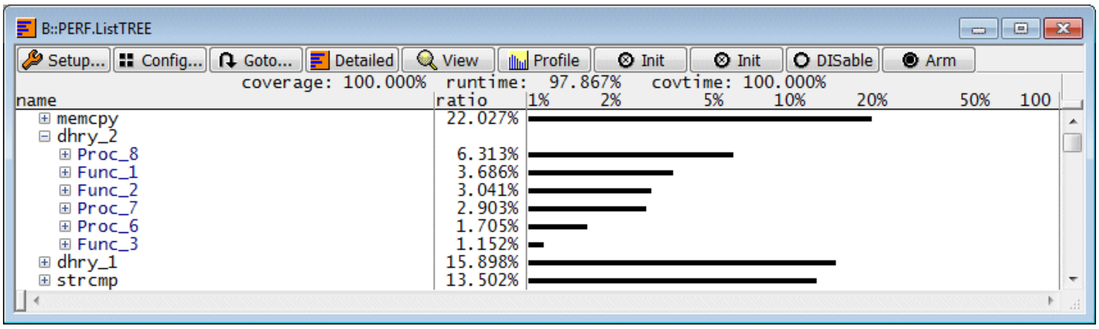
```
; inform TRACE32 which variable contains the identifier for the
; current task
; ~~ represents the TRACE32 installation directory
TASK.CONFIG ~~/demo/kernel/simple/simple.t32 current_task

; specify names for the individual tasks
Task.NAME.Set 0x4bca "Idle Task"
TASK.NAME.Set 0x58cc0 "Thread 1"

; list specified task names
TASK.NAME.view

; display the Performance Analyzer configuration window
PERF.state

; reset the Performance Analyzer configuration to its default settings
PERF.RESet

; enable Performance Analyzer
PERF.OFF

; the Performance Analyzer samples the contents of the variable that
; contains the identifier for the current task
PERF.Mode TASK

; TRACE32 sets the acquisition method StopAndGo
; PERF.METHOD StopAndGo

; open a window to display a task profiling
PERF.ListTASK

; start the program execution and the sampling
Go
```

See also

- PERF
- PERF.state

Format:	**PERF.ListTREE** [*<column>* …] [**/***<option>*] (program counter sampling)
<option>:	**Track** | **Address** *<range>* | *<address>* **CORE** *<core_number>* | **MergeCORE** | **SplitCORE**

Reports the percentage of run-time spent in modules/functions as a tree display. The tree is based on the module/function information provided by the symbol database.



```
                           coverage: 100.000%  runtime:   97.867%   covtime: 100.000%
name                                   ratio    1%     2%       5%     10%     20%      50%   100
  ⊞ memcpy                              22.027%
  ⊟ dhry_2
    ⊞ Proc_8                             6.313%
    ⊞ Func_1                             3.686%
    ⊞ Func_2                             3.041%
    ⊞ Proc_7                             2.903%
    ⊞ Proc_6                             1.705%
    ⊞ Func_3                             1.152%
  ⊞ dhry_1                              15.898%
  ⊞ strcmp                              13.502%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

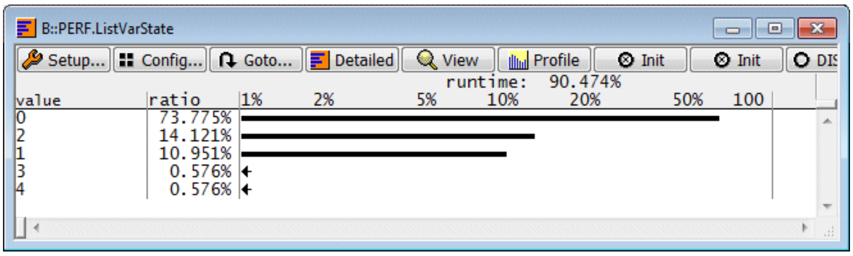
CORE, **MergeCORE**, **SplitCORE**	For details, refer to **"Profiling for SMP Systems"**, page 48.

See also

■ PERF ■ PERF.state

Format: **PERF.ListVarState** [*<column>* …] [**/Track**]
(memory contents sampling)

Reports the percentage of run-time a variable had a certain contents.



```
                          runtime:   90.474%
value        ratio    1%     2%       5%     10%     20%        50%   100
0            73.775%
2            14.121%
1            10.951%
3             0.576%
4             0.576%
```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down is given in the description of the **PERF.List** command.

Example for ARM9:

```
PERF.state                                  ; display the Performance
                                            ; Analyzer configuration
                                            ; window

PERF.RESet                                  ; reset the Performance
                                            ; Analyzer configuration to
                                            ; its default settings

PERF.OFF                                    ; enable Performance Analyzer

PERF.Mode MEMory                            ; the Performance Analyzer
                                            ; samples the contents of
                                            ; a memory location

; PERF.METHOD StopAndGo                     ; TRACE32 set the acquisition
                                            ; method StopAndGo

PERF.SnoopAddress Var.RANGE(sched_Lock)     ; specifies the address range
                                            ; of the variable

PERF.SnoopSize Var.SIZEOF(sched_Lock)       ; specifies the sampling width

PERF.ListVarState                           ; open a window for variable
                                            ; profiling

Go                                          ; start the program execution
                                            ; and sampling
```

See also

- PERF
- PERF.state

PERF.LOAD — Load previously stored PERF results

Format:	**PERF.LOAD** *<file>*

Loads the PERF results previously stored with the **PERF.SAVE** command for postprocessing.

See also

■ PERF ■ PERF.SAVE ■ PERF.state

PERF.METHOD — Specify acquisition method

Format:	**PERF.METHOD** *<mode>*
<mode>:	**StopAndGo** **Trace** **Snoop** **DCC** (only if JTAG interface provides Data Communications Channel)

The TRACE32 software sets automatically the acquisition method **Snoop**:

- If the processor allows to read the program counter while the program execution is running and **PERF.Mode PC** is selected.
- If the processor allows to read the contents of a memory locations while the program execution is running and **PERF.Mode MEMory** or **TASK** is selected.

Otherwise the default method is set to **StopAndGo**.

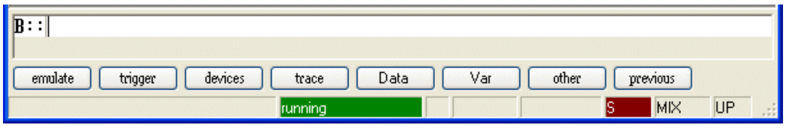
Performance Analyzer Methods	
StopAndGo	The target processor is stopped periodically in order to get the actual program counter or in order to read the data information of interest (intrusive). For details refer to **"The Method StopAndGo"** in General Commands Reference Guide P, page 78 (general_ref_p.pdf).

Snoop	The actual program counter or the data information of interest is read while the program execution is running (non-intrusive). Sampling is done as fast as possible (no snoop fails). The minimum rate is 10 samples per second. The sampling rate is set slightly varied to avoid any side effects with the timing of the application / target. For details, refer to **“The Method Snoop”** in General Commands Reference Guide P, page 79 (general_ref_p.pdf).
Trace	This method requires an off-chip trace port. In order to get the actual program counter or the data information of interest, the trace recording is stopped shorty to get a big enough section of the most recent trace information (non-intrusive). Sampling is done as fast as possible (no snoop fails). The minimum rate is 10 samples per second. The sampling rate is set slightly varied to avoid any side effects with the timing of the application / target. For details, refer to **“The Method Trace”** in General Commands Reference Guide P, page 83 (general_ref_p.pdf).
DCC	The Performance Analyzer sample the data provided via the DCC (intrusive due to code instrumentation in the target application). For details, refer to **“The Method DCC”** in General Commands Reference Guide P, page 87 (general_ref_p.pdf).

The Method StopAndGo

The method StopAndGo is available for all processors.

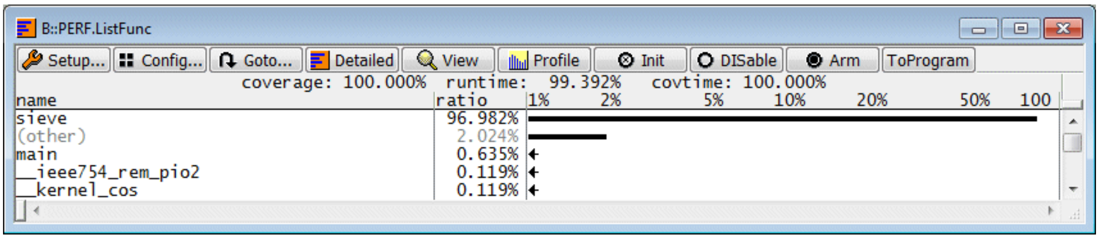
The target processor is stopped periodically in order to get the actual program counter or in order to read the data information of interest. The target processor is restarted afterwards. A stop and restart of the target processor can take more than 1 ms in a worst case scenario.



The display of a red **S** in the TRACE32 state line indicates that the program execution is periodically interrupted by the Performance Analyzer.

The field **snoops/s** in the **PERF.state** window shows how much stops have been performed in the last second.

The field **runtime** in the **PERF.List<item>** window shows the percentage of time taken by the actual program run in the last second.



```
                      coverage: 100.000%  runtime:   99.392%   covtime: 100.000%
name                          ratio       1%     2%        5%      10%      20%       50%    100
sieve                         96.982%
(other)                        2.024%
main                           0.635%
__ieee754_rem_pio2             0.119%
__kernel_cos                   0.119%
```

TRACE32 starts the sampling with 100 stops per second, but then tunes the sampling rate so that more the 99% of the run-time is retained for the actual program run. The smallest possible sampling rate is nevertheless 10.

A fixed percentage of time can be retained for the actual program run by the command **PERF.RunTime**.

The actual program counter or the data information of interest is read while the program execution is running (non-intrusive).

Non intrusive sample-based profiling can be done, if the target processor supports

- reading the program counter while the target program is running.
- reading memory (never cache) while the target program is running.

TRACE32 is optimizing the sampling rate. The achieved sampling rate of the last second is displayed in the field **snoops/s** in the **PERF.state** window.

Combi-modes e.g. **PERF.Mode PCMEMory** operate only if both, reading the program counter and reading memory is supported while the target program is running.

Processor architecture that allow to read the program counter while the program execution is runing	
ARC600 **ARC700**	
ARM1136 **Cortex-M0** **Cortex-M1** **Cortex-M3** **Cortex-A5** **Cortex-A9** **ARMV8**	If Program Counter Sampling Register (PCSR) is implemented
Blackfin	
CEVA-X1622 **TeakLite-III**	
DSP56300 **DSP56800E**	
GTM	
M8051EW	
MIPS32 **MIPS64**	Starting from eJTAG 3.1
R8051XC	
RH850	

Processor architecture that allow to read the program counter while the program execution is runing	
RX	
SH	only SH4A cores
TMS320C28xx **TMS320C54xx** **TMS320C55xx** **TMS320C62xx** **TMS320C64xx** **TMS320C67xx**	
TriCore	
V850	only via quick access

Processor architectures that allow to read memory (not cache) while the program execution is running	
78K0R	
ARC600 **ARC700**	
Blackfin	Only via Background Telemetric Channel
ColdFire	
Cortex-A/R **other ARM cores**	If the DAP is connected to the AHB bus
Cortex-M	
GTM	
MPC55xx/56xx	Via NEXUS block
QORIQ	
RH850	
S12X, MCS12, 68HC12	
SH2/SH2A	

Processor architectures that allow to read memory (not cache) while the program execution is running	
TMS320C28xx **TMS320C54xx** **TMS320C55xx** **TMS320C62xx** **TMS320C64xx** **TMS320C67xx**	
TriCore	
V850 E1 core	by QUICK access
V850 E2 core	
XC2000/C166S V2	
ZSP500	Debug Emulation Unit only

Example: Program counter sampling

```
PERF.state                                    ; display the Performance
                                              ; Analyzer configuration
                                              ; window

PERF.RESet                                    ; reset the Performance
                                              ; Analyzer configuration to
                                              ; its default settings

PERF.OFF                                      ; enable Performance Analyzer

PERF.Mode PC                                  ; the Performance Analyzer samples
                                              ; the program counter

;PERF.METHOD Snoop                            ; TRACE32 detects automatically
                                              ; that reading the program counter
                                              ; is possible while the program
                                              ; execution is running

PERF.ListFunc                                 ; open a window for function
                                              ; profiling

Go                                            ; start the program execution and
                                              ; the measurement
```

Example: Memory contents sampling

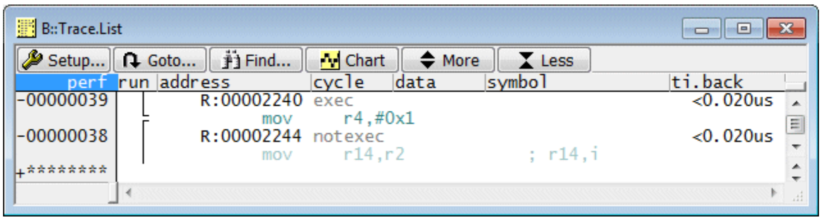
```
PERF.state                          ; display the Performance
                                    ; Analyzer configuration
                                    ; window

PERF.RESet                          ; reset the Performance
                                    ; Analyzer configuration to
                                    ; its default settings

PERF.OFF                            ; enable Performance Analyzer

PERF.Mode MEMory                    ; the Performance Analyzer samples
                                    ; the contents of a memory location

;PERF.METHOD Snoop                  ; TRACE32 detects automatically
                                    ; that reading memory is possible
                                    ; while the program execution is
                                    ; running

PERF.SnoopAddress 0xA108002F        ; specifies the memory address

PERF.SnoopSize Word                 ; specifies the sampling width

PERF.ListDistriB                    ; open a window for memory contents
                                    ; profiling

Go                                  ; start the program execution and
                                    ; the measurement
```

NOTE:	The sampling rate of **PERF.METHOD Trace** is much slower than the sampling rate of **PERF.METHOD Snoop**. Use **PERF.METHOD Trace** only if: • You do not want to stop the application. • The option **Snoop** (= **PERF.METHOD Snoop**) is disabled in the **PERF.state** window. • The architecture supports a trace that can be read without stopping the application.

This non-intrusive method is only available if the processor provides an off-chip trace port. Please make sure, that the trace recording is working correctly before you use the **PERF.METHOD Trace**.

In order to get the actual program counter or the data information of interest, the trace recording is stopped shortly to get a big enough section of the most recent trace information.

The field **snoop fails** in the **PERF.state** window shows how often TRACE32 failed to get the requested information out of the captured section.



```
     perf run address        cycle    data      symbol               ti.back
-00000039         R:00002240 exec                                   <0.020us
                      mov       r4,#0x1
-00000038         R:00002244 notexec                                <0.020us
                      mov       r14,r2                ; r14,i
+********
```

The display of **perf** in blue in any Trace display window indicates that the trace recording was periodically interrupted by the Performance Analyzer. In this case the trace information is inappropriate for any trace analysis.

Sampling the actual program counter (PERF.Mode PC)

If the actual program counter is sampled the source code is required to decompress the trace information. If the target processor doesn’t allow to read memory while the program execution is running, the source code has to be loaded to the TRACE32 virtual memory.

Sampling data information (PERF.Mode MEMory/TASK)

If data information is sampled it is recommended to set a filter on the data of interest. Otherwise the number of **snoop fails** will be too high.

Example for MPC5554: NEXUS block allows to read source code from memory while the program execution is running.

```
...

TRANSlation.Create 0x0--0xffffffff 0x0     ; specify 1:1 translation of
                                           ; effective to real addresses
                                           ; for debugger MMU

TRANSlation.ON                             ; activate translation via
                                           ; debugger MMU

...

NEXUS.DTM OFF                              ; switch data trace off in
                                           ; order to reduce load on the
                                           ; NEXUS port

PERF.state                                 ; display the Performance
                                           ; Analyzer configuration
                                           ; window

PERF.RESet                                 ; reset the Performance
                                           ; Analyzer configuration to
                                           ; its default settings

PERF.OFF                                   ; enable Performance Analyzer

PERF.METHOD Trace                          ; set acquisition method Trace

PERF.Mode PC                               ; the Performance Analyzer
                                           ; samples the program counter

PERF.ListFunc                              ; open a window for
                                           ; function profiling

Go                                         ; start the program execution
                                           ; and the sampling
```

Example for ARM920: Load the source code to the virtual memory of TRACE32 because it is not possible to read the source code from memory while the program execution is running.

```
Data.LOAD.Elf armle.axf /VM           ; load source code to virtual
                                      ; memory of TRACE32

ETM.DataTrace off                     ; switch data trace off in order to
                                      ; reduce load on ETM trace port

PERF.state                            ; display the Performance
                                      ; Analyzer configuration
                                      ; window

PERF.RESet                            ; reset the Performance
                                      ; Analyzer configuration to
                                      ; its default settings

PERF.OFF                              ; enable Performance Analyzer

PERF.METHOD Trace                     ; set acquisition method Trace

PERF.Mode PC                          ; the Performance Analyzer samples
                                      ; the program counter

PERF.ListLABEL                        ; open a window for label-based
                                      ; profiling

Go                                    ; start the program execution and
                                      ; the sampling
```

Example for ARM920: A filter is set to advise the ETM to only broadcast trace information if a write access to the variable flags[3] occurs.

```
Var.Break.Set flags[3] /TraceEnable /Write     ; configure the ETM so
                                               ; that only write
                                               ; accesses to the
                                               ; variable flags[3] are
                                               ; broadcast

PERF.state                                     ; display the Performance
                                               ; Analyzer configuration
                                               ; window

PERF.RESet                                     ; reset the Performance
                                               ; Analyzer configuration
                                               ; to its default settings

PERF.OFF                                       ; enable Performance
                                               ; Analyzer

PERF.METHOD Trace                              ; set acquisition method
                                               ; Trace

PERF.Mode MEMory                               ; the Performance
                                               ; Analyzer samples
                                               ; memory contents

PERF.SnoopAddress Var.RANGE(flags[3])          ; specifies the sampling
                                               ; address

PERF.SnoopSize Byte                            ; specifies the sampling
                                               ; width

PERF.ListVarState                              ; open a variable state
                                               ; profiling window

Go                                             ; start the program
                                               ; execution and
                                               ; the sampling
```

The Method DCC

DCC (Debug Communications Channel) is a feature of the on-chip debugging logic currently available for all ARM/Cortex cores (not Cortex-M) and the StarCore architecture. DCC allows the target program to provide data of interest to the TRACE32 debugger. For details on DCC, refer to the manual of your target CPU.

Examples of how to use the DCC with TRACE32 are given in the TRACE32 demo folder:

```
~~/demo/arm/etc/semihosting_arm_dcc
```

The Performance Analyzer sample the data provided via the DCC. The DCC method is recommended mainly for **PERF.Mode MEMory and TASK**.

TRACE32 is optimizing the sampling rate. The achieved sampling rate of the last second is displayed in the field **snoops/s** in the **PERF.state** window.

Example for ARM920: The contents of a variable is sent via DCC to TRACE32.

```
...

PERF.state                                    ; display the Performance
                                              ; Analyzer configuration
                                              ; window

PERF.RESet                                    ; reset the Performance
                                              ; Analyzer configuration to
                                              ; its default settings

PERF.OFF                                      ; enable Performance Analyzer

PERF.METHOD DCC                               ; set acquisition method DCC

PERF.Mode MEMory                              ; the Performance Analyzer samples
                                              ; data information

PERF.ListVarState                             ; open a variable state profiling
                                              ; window

Go                                            ; start the program execution and
                                              ; the sampling
```

See also

■ PERF ■ PERF.state ❏ PERF.METHOD()

PERF.MMUSPACES Include space IDs for addresses in the sampling

Format:	**PERF.MMUSPACES** [**ON** | **OFF**]

If a target operating system (e.g. Linux, Windows CE) is used, several processes/tasks can run at the same logical addresses. In this scenario, the logical address sampled by the Performance Analyzer is not sufficient to assign the address to a function or variable. For a clear assignment the space ID is also required.

OFF (default)	The Performance Analyzer does standard sampling.
ON	The Performance Analyzer includes the space ID in the sampling.

See also

■ PERF ■ PERF.state

▲ 'Release Information' in 'Legacy Release History'

PERF.Mode Specify sampling object

Format:	**PERF.Mode** *<mode>*
<mode>:	**PC** **TASK** **MEMory** **PCTASK** **PCMEMory**

Selects the sampling object for the sample-based profiling.

TRACE32 samples in essence either:

- The actual program counter (PC)
- The contents of a memory location (MEMory, TASK)
- Or both simultaneously (PCMEMory, PCTASK)

The sampled program counter information and the sampled data information can only be profiled independently of each other.

PC	The actual program counter is sampled.
TASK	The contents of the variable that contains the identifier for the actual task is sampled. If **OS-aware debugging** is configured, TRACE32 knows the address of this variable (TASK.CONFIG(magic)). Context ID packets are not supported.
MEMory	The memory address specified by the command **PERF.SnoopAddress** is sampled in the size specified by the command **PERF.SnoopSize**.
PCTASK	The actual program counter and the contents of the variable that contains the identifier for the actual task are sampled. The information is sampled simultaneous, but can only be evaluated separately.
PCMEMory	The actual program counter and the memory address specified by the command **PERF.SnoopAddress** is sampled in the size specified by the command **PERF.SnoopSize.** The information is sampled simultaneous, but can only be evaluated separately.

Not all **PERF Modes** are suitable for all **PERF METHODs**. The table below provides a summary.

	Mode PC	**Mode MEMory/TASK**	**Mode PCMEMory/PCTASK**
METHOD StopAndGo	yes	yes	yes
METHOD Trace	yes	yes, but requires appropriate filter	no
METHOD Snoop	yes, if the program counter can be read during program run	yes, if memory can be read during program run	yes, if program counter and memory can be read during program run
METHOD DCC	no	yes	no

See also

■ PERF ■ PERF.state ❑ PERF.MODE()

▲ 'Release Information' in 'Legacy Release History'

PERF.OFF — Stop the performance analyzer manually

Format:	**PERF.OFF**

The Performance Analyzer is coupled to the program execution if **PERF.AutoArm** is ON (default).

If **PERF.AutoArm** is OFF, the Performance Analyzer can be controlled manually. **PERF.Arm** activates the Performance Analyzer, **PERF.OFF** stops the Performance Analyzer.

If the Performance Analyzer is disabled (state disable) it can be enable by **PERF.OFF**.

See also

■ PERF ■ PERF.state ❏ PERF.STATE()

PERF.PROfile — Graphic profiling display

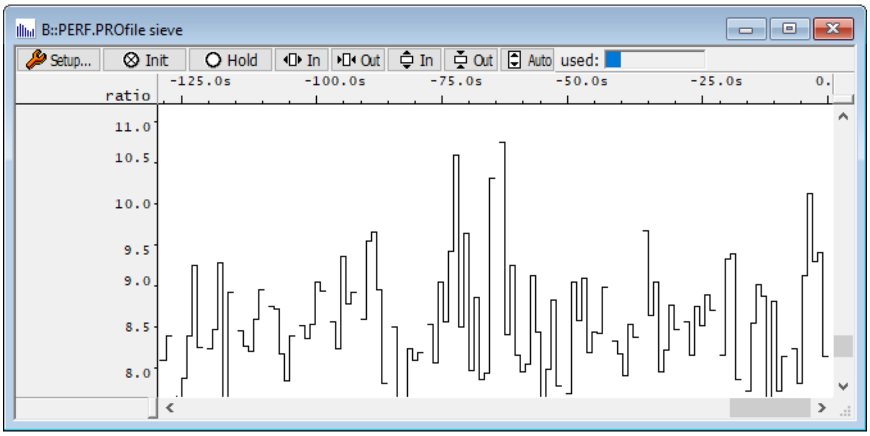
Format:	**PERF.PROfile** *<channel>* [*<channel>* [*<channel>*]] [*<gate>* *<scale>*]
<channel>:	*<range>* | *<address>* | *<value>*
<gate>:	**0.1s** | **1.0s** | **10.0s**
<scale>:	**1.** … **32768.**

The Performance Analyzer charts the percentage of time spent in the specified item over the time axis.

By default the display is updated once per second while the minimum update period is 100 ms. Within the update period a large number of PC samples is required to calculate a statistically relevant distribution of the runtime. Therefore using slow sample methods like *StopAndGo* with short update periods will give imprecise results.

Up to three channels may be displayed in one window. Channels correspond to a code areas like functions, address ranges, addresses, tasks or memory/variable contents.

```
PERF.METHOD StopAndGo                 ; take the samples for the profiling
                                      ; from the recorded trace information

PERF.Mode PC                          ; sample the program counter
                                      ; information

PERF.Arm                              ; arm the Performance Analyzer

PERF.PROfile sieve                    ; restrict the evaluation of the
                                      ; result to the program range of the
                                      ; function sieve
```



See also

- PERF
- PERF.state

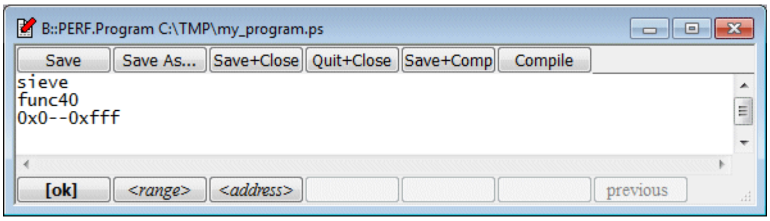
Format:	**PERF.Program** [*<file>*] (program counter sampling only)

PERF.Program opens a Performance Analyzer programming window that allows to restrict the evaluation of the program counter sampling to address ranges of interest.

A programming file consists of a text file containing one or more address ranges, each on a separate line. The address ranges can be specified using a variety of methods:

Direct Address	AHB:08000000-AHB08FFFFFF
Address Symbols	main
Range Symbols	localArray++0xFF

See *<address ranges>* for more details on specifying address ranges.



```
sieve
func40
0x0--0xfff
```

Buttons in the PERF.Program window	
Save	Save the Performance Analyzer program. If no name is specified the default name t32.ps is used.
Save As …	Save the Performance Analyzer program under a different name.
Save + Close	Save the Performance Analyzer program and close the Performance Analyzer programming window.
Quit + Close	Quit editing and close the Performance Analyzer programming window.
Save + Comp	Save the Performance Analyzer program and activate it as done by **Compile**.
Compile	Compiles the Performance Analyzer program. The evaluation of the profiling is restricted to the specified address ranges in all **PERF.List<item>** windows that evaluate sampled program counter information.

Example:

```
PERF.state                           ; display the Performance Analyzer
                                     ; configuration window

PERF.RESet                           ; reset the Performance Analyzer
                                     ; configuration to its default
                                     ; settings

PERF.OFF                             ; enable the Performance Analyzer

; PERF.METHOD StopAndGo              ; the acquisition method StopAndGo
                                     ; is set by TRACE32

PERF.ReProgram my_program.ps         ; load a existing, error-free
                                     ; Performance Analyzer program

PERF.ListProgram                     ; open a window for Performance
                                     ; Analyzer program based profiling

Go                                   ; start the program execution and
                                     ; the sampling
```

See also

■ PERF ■ PERF.state

▲ 'Release Information' in 'Legacy Release History'

PERF.ReProgram Load an existing performance analyzer program

Format:	**PERF.ReProgram** [<*file*>] (program counter sampling only)

Loads an existing, error-free Performance Analyzer program to the Performance Analyzer.

See also

■ PERF ■ PERF.state

▲ 'Release Information' in 'Legacy Release History'

PERF.RESet

Reset analyzer

Format:	**PERF.RESet**

All settings of the performance analyzer and all marked breakpoints will be destroyed. The windows of the performance analyzer will be changed to the freeze mode and the performance analyzer will be disabled.

See also

■ PERF ■ PERF.state

PERF.RunTime

Retain time for program run

Format:	**PERF.RunTime** *<value>*

If **PERF.METHOD StopAndGo** is used a fraction of time is taken by the sample-based performance measurement, the rest is used by the actual program run. The command **PERF.RunTime** allows to specify the percentage of time that should be retained for the actual program run.

Examples:

```
PERF.RunTime 90.                        ; 90% of time is retained for the
                                        ; actual program run, the sample-
                                        ; based performance measurement can
                                        ; take 10% of the time

PERF.RunTime 90%                        ; alternative input format
```

The adjustment of the snoops/s is done gradually (see the **snoops/s** field in the **PERF.state** window).

See also

■ PERF ■ PERF.state

PERF.SAVE Save the PERF results for postprocessing

Format: **PERF.SAVE** *<file>*

The PERF results are stored to the selected file. The file can be then loaded for postprocessing with the **PERF.LOAD** command.

See also

- ■ PERF
- ■ PERF.LOAD
- ■ PERF.state

PERF.SnoopAddress Address for memory sample

Format: **PERF.SnoopAddress** *<address>* | *<range>*
(memory contents sampling only)

Defines the memory address for snoop modes (**DistriBution**, **VarState**). Supplying an address range defines also the size of the memory operation (**PERF.SnoopSize**).

See also

- ■ PERF
- ■ PERF.state
- ❏ PERF.MEMORY.SnoopAddress()

PERF.SnoopMASK Mask for memory sample

Format: **PERF.SnoopMASK** *<value>*
(memory contents sampling only)

Defines the sample mask for snoop modes (**DistriBution**, **VarState**).

See also

- ■ PERF
- ■ PERF.state

PERF.SnoopSize — Size for memory sample

Format:	**PERF.SnoopSize Byte** | **Word** | **Long** (memory contents sampling only)

Defines the memory access size for snoop modes (**DistriBution**, **VarState**).

See also

- ■ PERF
- ❏ PERF.MEMORY.SnoopSize()
- ■ PERF.state

PERF.Sort — Specify sorting of evaluation results

Format:	**PERF.Sort** *<mode>*
<mode>:	**OFF** **Address** **sYmbol** **Ratio**

As a default the results are sorted by ratio.

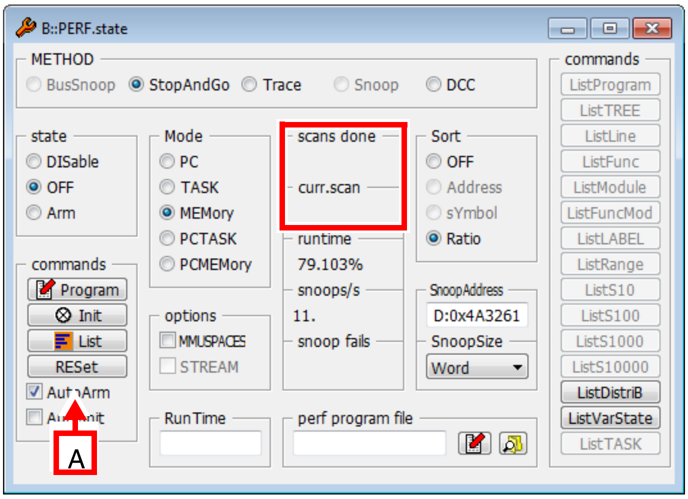
OFF	Don’t sort. Results of the program counter sampling are sorted by address, results of memory contents sampling are sorted by occurrence.
Address	Sort evaluation result by addresses (program counter sampling only).
sYmbol	Sort evaluation result by symbol names (program counter sampling only).
Ratio	Sort evaluation result by the ratio of time used by the items.

See also

- ■ PERF
- ■ PERF.state

Format:	**PERF.state**

Displays the control window for the Performance Analyzer.



METHOD: BusSnoop [✓ StopAndGo] Trace Snoop DCC
state: DISable [✓ OFF] Arm
Mode: PC TASK [✓ MEMory] PCTASK PCMEMory
Sort: OFF Address sYmbol [✓ Ratio]
[✓ AutoArm] AutoInit

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

scan done	Displays the number of scans already completed. The field will be displayed only, if the scanning mode is active, i.e. **Ratio** is active and more ranges than available counters are covered.
curr.scan	The 'current scan' field displays the ratio of the scanned ranges to total the number of ranges.
covered time	The 'covered time' field gives the time covered by the current set of ranges. (not shown in the above **PERF.state** window.)

See also

■ PERF	■ PERF.ADDRESS	■ PERF.Arm	■ PERF.AutoArm
■ PERF.AutoInit	■ PERF.ContextID	■ PERF.DISable	■ PERF.Init
■ PERF.List	■ PERF.ListDistriB	■ PERF.ListFunc	■ PERF.ListFuncMod
■ PERF.ListLABEL	■ PERF.ListLine	■ PERF.ListModule	■ PERF.ListProgram
■ PERF.ListRange	■ PERF.ListS10	■ PERF.ListTASK	■ PERF.ListTREE
■ PERF.ListVarState	■ PERF.LOAD	■ PERF.METHOD	■ PERF.MMUSPACES
■ PERF.Mode	■ PERF.OFF	■ PERF.PROfile	■ PERF.Program
■ PERF.ReProgram	■ PERF.RESet	■ PERF.RunTime	■ PERF.SAVE
■ PERF.SnoopAddress	■ PERF.SnoopMASK	■ PERF.SnoopSize	■ PERF.Sort
■ PERF.STREAM	■ PERF.ToProgram	■ PERF.View	❏ PERF.METHOD()
❏ PERF.MODE()	❏ PERF.RATE()	❏ PERF.RunTime()	❏ PERF.STATE()

▲ 'Release Information' in 'Legacy Release History'

PERF.STREAM PERF stream mode

Format:	**PERF.STREAM** [**ON** | **OFF**] (program counter sampling and StopAndGo method only)

Default: OFF

Enable/disable STREAM mode for program counter sampling when **PERF.METHOD** is set to StopAndGo.

When STREAM mode is enabled, the sampling is performed by the software running on the PowerDebug module instead of the PowerView host software which leads to higher sampling rates.

The STREAM mode cannot be used together with **PERF.MMUSPACES**.

See also

- PERF
- PERF.state

PERF.ToProgram Automatic generation of performance analyzer program

Format:	**PERF.ToProgram** (program counter sampling only)

The different **PERF.List<item>** commands partition the address spaces into address ranges in order to evaluate the sampled program counter information. Examples:

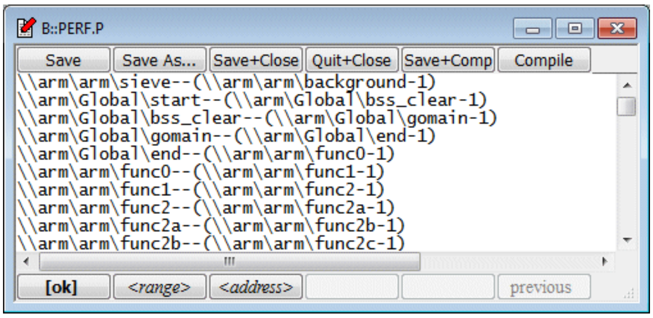
PERF.ListFunc	Partitions the address space in function ranges
PERF.ListLine	Partitions the address space in high-level language line ranges
PERF.ListModule	Partitions the address space in module ranges

The command **PERF.ToProgram** converts the current segmentation into a Performance Analyzer program.

TRACE32 allows up to 1024 address ranges in a Performance Analyzer program.

Example for ARM9:

```
PERF.state                            ; display the Performance Analyzer
                                      ; configuration window

PERF.RESet                            ; reset the Performance Analyzer
                                      ; configuration to its default
                                      ; settings

PERF.OFF                              ; enable Performance Analyzer

PERF.Mode PC                          ; the Performance Analyzer samples
                                      ; the actual program counter

; PERF.METHOD StopAndGo               ; acquisition method StopAndGo
                                      ; is set by TRACE32

PERF.ListLABEL                        ; open a window for label-based
                                      ; profiling

Go                                    ; start the program execution and
                                      ; sampling

Break                                 ; stop the program execution and
                                      ; the sampling

PERF.ToProgram                        ; convert the listed label ranges
                                      ; to a Performance Analyzer program
```



```
\\arm\arm\sieve--(\\arm\arm\background-1)
\\arm\Global\start--(\\arm\Global\bss_clear-1)
\\arm\Global\bss_clear--(\\arm\Global\gomain-1)
\\arm\Global\gomain--(\\arm\Global\end-1)
\\arm\Global\end--(\\arm\arm\func0-1)
\\arm\arm\func0--(\\arm\arm\func1-1)
\\arm\arm\func1--(\\arm\arm\func2-1)
\\arm\arm\func2--(\\arm\arm\func2a-1)
\\arm\arm\func2a--(\\arm\arm\func2b-1)
\\arm\arm\func2b--(\\arm\arm\func2c-1)
```

See also

■ PERF ■ PERF.state

PERF.View Detailed view

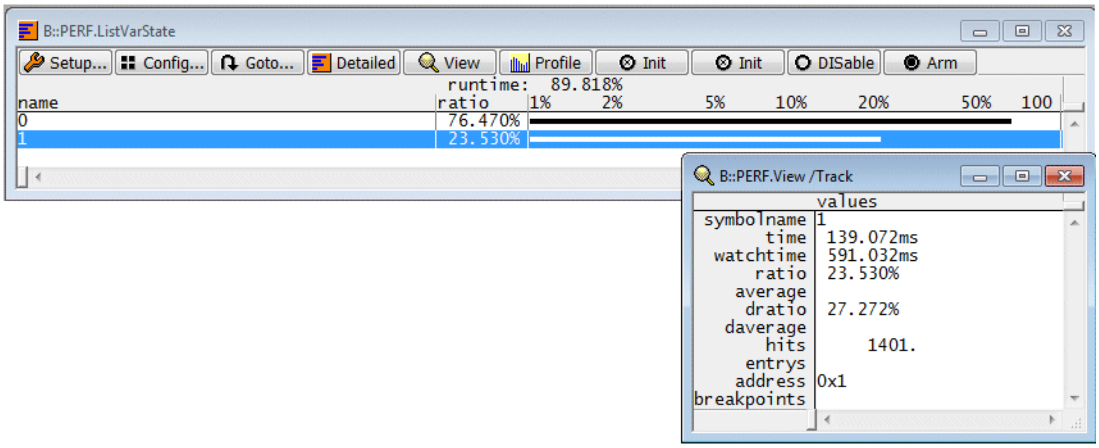
Format: **PERF.View** *<address>* | **/Track**

Displays all numerical results of a symbol or an area.

Examples:

```
PERF.View sieve                          ; list all numerical results for
                                         ; the function sieve
```

```
PERF.state                               ; display the Performance
                                         ; Analyzer configuration window

PERF.RESet                               ; reset the Performance Analyzer
                                         ; to its default settings

PERF.OFF                                 ; enable the Performance
                                         ; Analyzer

PERF.Mode MEMory                         ; the Performance Analyzer
                                         ; samples the contents of a
                                         ; memory location

; PERF.Mode StopAndGo                    ; the Performance Analyzer sets
                                         ; the acquisition method
                                         ; StopAndGo

PERF.SnoopAddress Var.RANGE(flags[3])    ; specify the memory address

PERF.SnoopSize Byte                      ; specify the sampling width

PERF.ListVarState                        ; open a window for variable
                                         ; state profiling

Go                                       ; start the program execution
                                         ; and the sampling

PERF.View /Track                         ; list all numerical results for
                                         ; the item selected in
                                         ; PERF.List<item>
```



```
                              runtime:  89.818%
name                          ratio     1%    2%     5%   10%   20%    50%  100
0                             76.470%
1                             23.530%
```

```
             values
 symbolname 1
       time  139.072ms
  watchtime  591.032ms
      ratio  23.530%
    average
     dratio  27.272%
   daverage
       hits       1401.
     entrys
    address 0x1
breakpoints
```

See also

- PERF
- PERF.state

PERSVD

PERSVD — Built-in converter for peripheral files in CMSIS-SVD format

Allows you to display peripheral files written in CMSIS-SVD format. Furthermore you can export an SVD file to Lauterbach's native peripheral file format.

PERSVD.Save — Save converted file

Format:	**PERSVD.Save** *<svd_file>* *<per_file>* [**/***<option>*]
<option>:	See **PERSVD.view**

Converts the given **svd_file** to native Lauterbach peripheral file format and saves it to a file named **per_file**.

svd_file	Source file in CMSIS-SVD format.
per_file	Destination file name. Will be overwritten if the file already exists.

PERSVD.view — Display peripherals

Format:	**PERSVD.view** *<file>* [**/***<option>*]
<option>:	**WithValue** **Description** **AccessClass** *<class>* For additional options see **PER.view**

Converts a CMSIS-SVD file to Lauterbach's native peripheral file format and displays its peripherals. See **PER.view**.

WithValue	Precedes bitfield names or descriptions with the value followed by a colon: “<value>: <name>”.
Description	In case of bitfields, the description instead of the name will be taken from the SVD file.
AccessClass	Since SVD files don’t know about TRACE32 access classes, the default access class is “AD:”. With this option you can change the default, e.g. **PERSVD.view** *<file>* **/AccessClass d:**

In case you encounter any errors during conversion, it might be helpful to save the converted intermediate to a file (**PERSVD.Save**) first and to process the result via **PER.Program** afterwards.

PMI

Power management interface

For a description of the **PMI** commands, see **“System Trace User’s Guide”** (trace_stm.pdf).

POD — Configure input behavior of digital and analog probe

See also

- POD.ADC
- POD.Level
- POD.RESet
- POD.state
- POD.USB

POD.ADC — Probe configuration

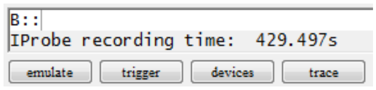
[Example]

Format:	**POD.ADC** *<probe>*.*<voltage>* [**ON** | **OFF**] [*<comp>*] [*<sample>*] **POD.ADC** *<probe>*.*<current>* [**ON** | **OFF**] [*<comp>*] [*<sample>*] [*<shunt>*] **POD.ADC** *<probe>*.*<power>* [**ON** | **OFF**] [*<vref>*] **CIProbe.CONFIG.CHANNEL** <...> (deprecated)
<probe>:	**A** | **IP** | **CIP**
<voltage>:	**V0** | **V1** | **V2** | **V3**
<current>:	**I0** | **I1** | **I2**
<power>:	**P0** | **P1** | **P2**
<comp>:	**1/1** | **2/1** | **4/1** | **8/1** | **16/1** | **32/1** | **64/1** | **128/1** | **256/1**
<sample>:	**ALways** | **Track** | **BusA** | **Filter**
<shunt>:	*<float>*
<vref>:	**V0** | **V1** | **V2** | *<float>*

ADC stands for analog-digital converter. The **POD.ADC** command allows you to programmatically configure the Analog Probe together with the PowerIntegrator, PowerIntegrator II, IProbe or CIProbe. Alternatively, you can manually configure the hardware via the **POD.state A**, **POD.state IP** or the **POD.state CIP** window.

Note that all parameters after the channel are optional, but have to be specified in the correct order. If a parameter is not given, that setting remains unchanged.

<probe>	**A** stands for port A of the PowerIntegrator or PowerIntegrator II. **IP** stands for the IProbe. **CIP** stands for the CombiProbe.
<voltage> *<current>* *<power>*	The following channels are available: • Four voltage channels (V0, V1, V2, and V3) • Three current channels (I0, I1, and I2) • Three virtual power channels (P0, P1, and P2).

<comp>	Changing the compression changes the recording time: The higher the compression factor, the longer the recording time.
	For the IProbe, the resulting recording time is displayed in the message bar below the command line and in the **AREA** window.
	
	Example: A compression factor of 256/1 for all channels results in a recording time of 429 seconds. A compression factor of 1/1 for all channels results in a recording time of 1.678 seconds.
	
	`B::`
	`IProbe recording time:  429.497s`
	
	A high compression factor reduces the noise, which results in a smoother line chart, e.g. in an **ETA.DRAW** or **IProbe.DRAW** window, and allows for a better interpretation of the line chart.
	
	This setting is not available for the virtual power channels. The setting from the corresponding current channel is used instead.

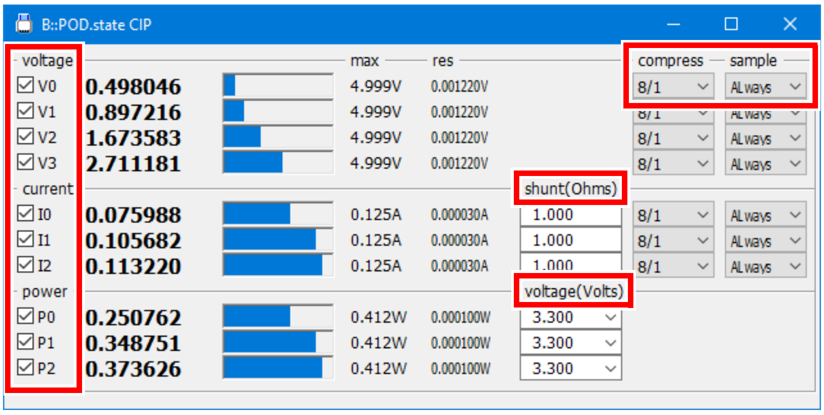
<sample>	• (Default) **ALways** for continuous recording of analog trace data. Use the option, for example, if you want to focus on power consumption even during the sleep mode of the CPU. • (IProbe only) **Track** for intermittent recording of analog trace data. Analog trace data is recorded only if a user-defined trigger event occurs in the program flow. Use this option, for example, if you want to record analog trace data when the CPU is active, i.e. not in sleep mode. • (IProbe only) **BusA**: Data is recorded if a PodBus trigger signal is detected on the bus trigger line BUSA. • **Filter**: Use the trigger logic to only record samples that are in a specified range. For the CIProbe, this condition can be configured using the command **CIProbe.ATrigMODE**. This setting is not available for the virtual power channels. The setting from the corresponding current channel is used instead. This settting is also not available for the PowerIntegrator or PowerIntegrator II.

<shunt>	To measure current, you have to use an appropriate shunt resistor and configure TRACE32 with the shunt resistance in Ohms. **Shunt formula**: $R_s = 0.125V / I_{max}$ • To achieve a maximum resolution of the analog-digital converter, the voltage drop permissible at the shunt must *not* exceed 0.125V. • I_{max} is the maximum current that you expect: The more accurate your estimate, the better the measurement accuracy. **Example**: $R_s = 0.125V / 4A = 0.031\Omega$ If a voltage drop of 0.125V is not acceptable in your case, then you may lower the voltage value from 0.125V to e.g. 0.05V. Note that this reduces the resolution of the analog-digital converter. **Example**: $R_s = 0.05V / 4A = 0.012\Omega$
<vref>	If you specify a voltage value (e.g. 3.3V), the system multiplies the voltage value with the value of the current channel (e.g. I1 = 0.019561A). **Example**: 3.3V x 0.019561A = 0.064553W Alternatively, you can select the corresponding voltage channel (e.g. V1 for P1). In this case, the IProbe or CIProbe automatically uses the voltage value from that voltage channel.

Example:

```
; Configure Analog Probe and IProbe
POD.ADC CIP V0 ON 8/1 ALways
POD.ADC CIP V1 ON 8/1 ALways
POD.ADC CIP V2 ON 8/1 ALways
POD.ADC CIP V3 ON 8/1 ALways
POD.ADC CIP I0 ON 8/1 ALways 1.000
POD.ADC CIP I1 ON 8/1 ALways 1.000
POD.ADC CIP I2 ON 8/1 ALways 1.000
POD.ADC CIP P0 ON 3.300
POD.ADC CIP P1 ON 3.300
POD.ADC CIP P2 ON 3.300

; Initialize the CIProbe.
CIProbe.Init

; Open the POD CIP window. The following screenshot displays the result.
POD.state CIP
```

The **POD.state CIP** window displays the result of the above script:



B::POD.state CIP

		max	res	shunt(Ohms) / voltage(Volts)	compress	sample
voltage						
☑ V0	0.498046	4.999V	0.001220V		8/1	ALways
☑ V1	0.897216	4.999V	0.001220V		8/1	ALways
☑ V2	1.673583	4.999V	0.001220V		8/1	ALways
☑ V3	2.711181	4.999V	0.001220V		8/1	ALways
current				shunt(Ohms)		
☑ I0	0.075988	0.125A	0.000030A	1.000	8/1	ALways
☑ I1	0.105682	0.125A	0.000030A	1.000	8/1	ALways
☑ I2	0.113220	0.125A	0.000030A	1.000	8/1	ALways
power				voltage(Volts)		
☑ P0	0.250762	0.412W	0.000100W	3.300		
☑ P1	0.348751	0.412W	0.000100W	3.300		
☑ P2	0.373626	0.412W	0.000100W	3.300		

See also

- POD
- POD.state

POD.Level

Input state

Format:	**POD.Level** *<group> <level>*	
<group>:	**00-15** | **16-31** | **32-47** | **48-63** | **SOC**	(PowerProbe)
	IP | **A** | **B** | **C** | **D** | **E** | **F**	(PowerIntegrator)
<level>:	**1.0** | **1.4**	(PowerProbe)
	0.0 … **5.0**	(PowerIntegrator)

Defines the variable threshold levels for the PowerProbe and the input probes of the PowerIntegrator.

Default is 1.4 V for all CMOS and TLL targets down to 2.5 V supply voltage.

00-15, …, **SOC**	Input channels of the PowerProbe
IP, **A**, **B**, **C**, **D**, **E**, **F**	**A** to **F**: Input channels of the PowerIntegrator **IP**: Input channel of the IProbe
1.0 and **1.4**	Threshold level settings of the PowerProbe
0.0 to **5.0**	Threshold level range of the PowerIntegrator

See also

- POD
- POD.state

POD.RESet — Input level reset

Format:	**POD.RESet**

All input threshold levels are set to 1.4 V.

All **POD.ADC** settings are reset.

See also

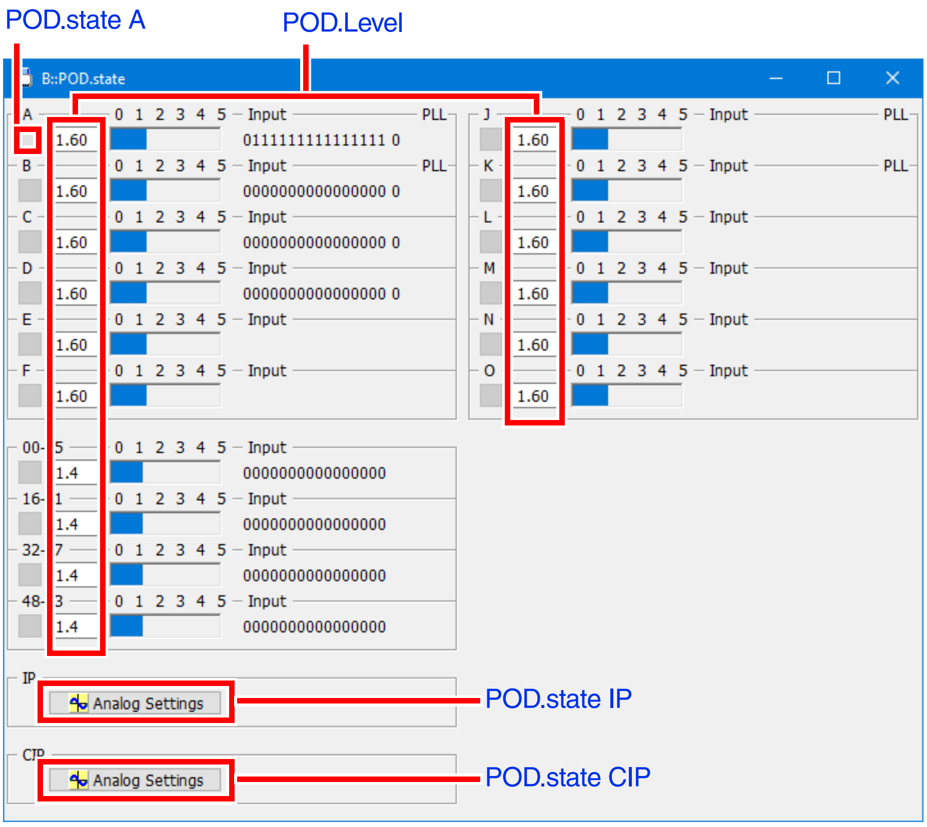
- POD
- POD.state

POD.state — Input state

Format: **POD.state**
POD.state *<probe>*
CIProbe.CONFIG.CHANNEL.state (deprecated)

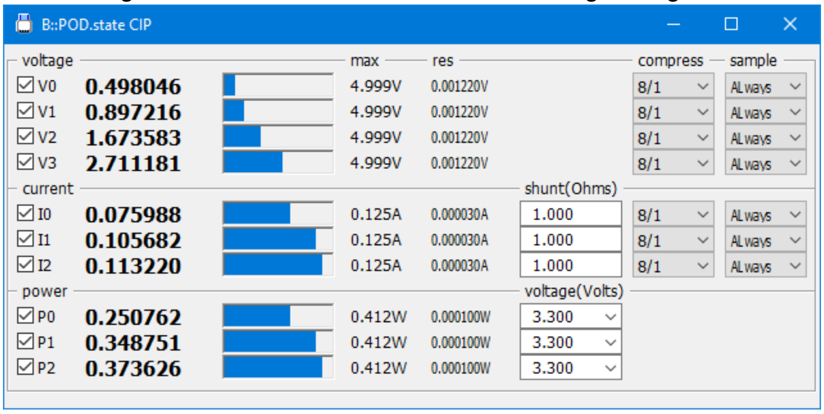
<probe>: **A** | **IP** | **CIP**

Without arguments, shows the digital probe configuration for PowerProbe, PowerIntegrator, PowerIntegrator II, IProbe, and CIProbe. The screenshot below shows the dialog with a PowerProbe, PowerIntegrator, IProbe and CIProbe,



```
B::POD.state
A     0 1 2 3 4 5 - Input             PLL     J     0 1 2 3 4 5 - Input    PLL
  1.60              0111111111111111 0            1.60
B     0 1 2 3 4 5 - Input             PLL     K     0 1 2 3 4 5 - Input    PLL
  1.60              0000000000000000 0            1.60
C     0 1 2 3 4 5 - Input                     L     0 1 2 3 4 5 - Input
  1.60              0000000000000000 0            1.60
D     0 1 2 3 4 5 - Input                     M     0 1 2 3 4 5 - Input
  1.60              0000000000000000 0            1.60
E     0 1 2 3 4 5 - Input                     N     0 1 2 3 4 5 - Input
  1.60                                            1.60
F     0 1 2 3 4 5 - Input                     O     0 1 2 3 4 5 - Input
  1.60                                            1.60

00- 5 0 1 2 3 4 5 - Input
  1.4               0000000000000000
16- 1 0 1 2 3 4 5 - Input
  1.4               0000000000000000
32- 7 0 1 2 3 4 5 - Input
  1.4               0000000000000000
48- 3 0 1 2 3 4 5 - Input
  1.4               0000000000000000

IP
  Analog Settings
CIP
  Analog Settings
```

With an argument, it can be used to show the analog settings of a connected Analog Probe:



```
B::POD.state CIP
voltage                      max      res          compress  sample
☑V0  0.498046                4.999V   0.001220V    8/1       ALways
☑V1  0.897216                4.999V   0.001220V    8/1       ALways
☑V2  1.673583                4.999V   0.001220V    8/1       ALways
☑V3  2.711181                4.999V   0.001220V    8/1       ALways
current                                   shunt(Ohms)
☑I0  0.075988                0.125A   0.000030A   1.000    8/1   ALways
☑I1  0.105682                0.125A   0.000030A   1.000    8/1   ALways
☑I2  0.113220                0.125A   0.000030A   1.000    8/1   ALways
power                                     voltage(Volts)
☑P0  0.250762                0.412W   0.000100W   3.300
☑P1  0.348751                0.412W   0.000100W   3.300
☑P2  0.373626                0.412W   0.000100W   3.300
```

See also

- POD
- POD.ADC
- POD.Level
- POD.RESet
- POD.USB

Format:	**POD.USB USB1 | USB2** **POD.USB ENABLE | DISABLE** *<packet>*
<packet>:	**RESVD | OUT | ACK | DATA0 | PING | SOF | NYET | DATA2 | SPLIT | IN | NAK | DATA1 | ERR | SETUP | STALL | MDATA**

Sets up the hardware of the USB probe.

USB1 | USB2	Selects USB mode.
<packet>	Enables/disables recording of specific USB packets (PID).

See also

- POD
- POD.state

PORT

NOTE:	If not otherwise mentioned, the described commands refer the timing analyzer mode!

PORT.Arm — Arm the trace

See command **<trace>.Arm** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 133).

PORT.AutoArm — Arm automatically

See command **<trace>.AutoArm** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 134).

PORT.BookMark — Set a bookmark in trace listing

See command **<trace>.BookMark** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 140).

PORT.Chart — Display trace contents graphically

See command **<trace>.Chart** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 143).

PORT.DRAW — Plot trace data against time

See command **<trace>.DRAW** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 200).

PORT.FindAll — Find all specified entries in trace

See command **<trace>.FindAll** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 236).

PORT.FindChange — Search for changes in trace flow

See command **<trace>.FindChange** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 237).

PORT.GOTO — Move cursor to specified trace record

See command **<trace>.GOTO** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 243).

PORT.Init — Initialize trace

See command **<trace>.Init** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 245).

PORT.OFF — Switch off

See command **<trace>.OFF** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 277).

PORT.PROfileChart — Profile charts

See command **<trace>.PROfileChart** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 283).

PORT.PROTOcol — Protocol analysis

See command **<trace>.PROTOcol** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 339).

PORT.REF — Set reference point for time measurement

See command **<trace>.REF** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 357).

PORT.RESet — Reset command

See command **<trace>.RESet** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 357).

PORT.SAVE Save trace for postprocessing in TRACE32

See command **<trace>.SAVE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 358).

PORT.SelfArm Automatic restart of trace recording

See command **<trace>.SelfArm** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 362).

PORT.SnapShot Restart trace capturing once

See command **<trace>.SnapShot** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 372).

PORT.STATistic Statistic analysis

See command **<trace>.STATistic** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 377).

PORT.Timing Waveform of trace buffer

See command **<trace>.Timing** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 496).

PORT.TRACK Set tracking record

See command **<trace>.TRACK** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 499).

PORT.ZERO Align timestamps of trace and timing analyzers

See command **<trace>.ZERO** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 502).

Probe

Probe logic analyzer

The trace method **Probe** is available if a PowerProbe module is connected.

For selecting and configuring the trace method Probe, use the TRACE32 command line or a PRACTICE script (*.cmm) or the **Probe.state** window [**A**].

Alternatively, execute the command **Trace.METHOD Probe** in order to select the trace method **Probe** and use the more general command group **Trace**.

Refer for more information to **"PowerProbe User's Guide"** (powerprobe_user.pdf) and **"PowerProbe/Port Analyzer Reference Guide"** (powerprobe_ref.pdf).

See also

■ Trace.METHOD

▲ 'Generic Probe Trace Commands' in 'PowerProbe/Port Analyzer Reference Guide'