



General Commands Reference Guide P

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

TRACE32 Documents	
General Commands and Functions	
General Commands Reference Guide P	1
PCP	6
PCPOnchip	6
PER	7
Function	7
PER.Program	Interactive programming 7
PER.ReProgram	Load default program 8
PER.Set	Modify memory 8
PER.Set.Field	Modify a bit field in memory 9
PER.Set.Index	Modify indirect (indexed) register 11
PER.Set.Out	Write data stream to memory 11
PER.Set.SaveIndex	Modify indirect (indexed) register 12
PER.Set.SHADOW	Modify data based on shadow RAM 13
PER.Set.simple	Modify memory 13
PER.view	Display peripherals 14
Programming Commands	15
PERF	16
Profiling Results	17
PERF.ADDRESS	Restrict evaluation to specified address area 19
PERF.ANYACCESS	Access selectivity 19
PERF.Arm	Activate the Performance Analyzer manually 20
PERF.AutoArm	Couple Performance Analyzer to program execution 21
PERF.DISable	Disable the Performance Analyzer 21
PERF.Display	Select the display format 21
PERF.Entry	Function runtime analysis 22
PERF.EntrySize	Function header size 22
PERF.Filter	Suppress display of items with specified characteristic 23
PERF.Gate	Gate time of the measurements 24
PERF.Init	Reset current measurement 24
PERF.List	Default profiling 25
PERF.ListDistriB	Memory contents profiling 31
PERF.ListFunc	Function profiling 32

PERF.ListFuncMod	Hll function profiling (restricted)	33
PERF.ListLABEL	Label-based profiling	35
PERF.ListLine	Profiling by hll lines	36
PERF.ListModule	Profiling by modules	37
PERF.ListProgram	Profiling based on Performance Analyzer program	37
PERF.ListRange	Profiling by ranges	38
PERF.ListS10	Profiling in n-byte segments	39
PERF.ListTASK	Profiling by tasks/threads	40
PERF.ListTREE	Profiling by module/function tree	42
PERF.ListVarState	Variable state profiling	43
PERF.METHOD	Specify acquisition method	45
The Method StopAndGo		46
The Method Snoop		47
The Method Trace		52
The Method DCC		56
The Emulator Methods Hardware and BusSnoop		57
PERF.MMUSPACES	tdb.	57
PERF.Mode	Specify sampling object	58
PERF.OFF	Stop the Performance Analyzer manually	61
PERF.PreFetch	Prefetch handling	61
PERF.PROfile	Graphic profiling display	62
PERF.Program	Write a Performance Analyzer program	66
PERF.ReProgram	Load an existing Performance Analyzer program	67
PERF.RESet	Reset analyzer	67
PERF.RunTime	Retain time for program run	68
PERF.SCAN	Scanning mode	68
PERF.SnoopAddress	Address for memory sample	69
PERF.SnoopSize	Size for memory sample	69
PERF.Sort	Specify sorting of evaluation results	70
PERF.state	Display state	71
PERF.ToProgram	Automatic generation of Performance Analyzer program	72
PERF.View	Detailed view	74
POD		76
POD.Level	Input state	76
POD.RESet	Input level reset	76
POD.state	Input state	77
Port		78
Port.AutoFocus	Calibrate AutoFocus preprocessor	78
Port.AutoTEST	Continuous measurement	78
Port.BookMark	Set a bookmark in trace listing	78
Port.Chart.Func	Function activity chart	78
Port.Chart.GROUP	Group activity chart	78
Port.Chart.Line	Graphical HLL lines analysis	79

Port.Chart.sYmbol	Symbol analysis	79
Port.Chart.TASK	Task activity display	79
Port.Chart.TASKFunc	Task related function run-time analysis	79
Port.Chart.TASKSRV	Service routine run-time analysis	79
Port.Chart.TASKState	Task state analysis	79
Port.Chart.VarState	Variable activity chart	79
Port.COVerage	Trace based code coverage	80
Port.COVerage.add	Add trace contents to database	80
Port.COVerage.Delete	Coverage modification	80
Port.COVerage.Init	Clear coverage database	80
Port.COVerage.List	Coverage display	80
Port.COVerage.ListFunc	Display coverage for HLL functions	80
Port.COVerage.ListModule	Display coverage for modules	80
Port.COVerage.ListVar	Display coverage for variable	81
Port.COVerage.LOAD	Load coverage database from file	81
Port.COVerage.RESet	Clear coverage database	81
Port.COVerage.SAVE	Save coverage database to file	81
Port.COVerage.Set	Coverage modification	81
Port.DisConfig.view	Trace disassemble setting	81
Port.DRAW	Graphical data display	81
Port.Enable	Operation mode	82
Port.Enable	Operation mode	82
Port.FindAll	Find all specified entries in trace	82
Port.MUX	Select channels	82
Port.PROTOcol.Chart	Graphic display for user defined protocol	82
Port.PROTOcol.Draw	Graphic display for user defined protocol	82
Port.PROTOcol.EXPORT	Export trace buffer for user defined protocol	82
Port.PROTOcol.Find	Find in trace buffer for user defined protocol	83
Port.PROTOcol.List	Display trace buffer for user defined protocol	83
Port.PROTOcol.STATistic	Display statistics for user defined protocol	83
Port.Select	Select trigger/counter line	83
Port.SET	Select line for recording	83
Port.SLAVE	Select slave mode	83
Port.STATistic	Statistic analysis	83
Port.STATistic	Statistic analysis	84
Port.STATistic.BondOut	Bondout mode	84
Port.STATistic.DIStance	Time interval for a single event	84
Port.STATistic.DistriB	Distribution analysis	84
Port.STATistic.DURation	Time between two events	84
Port.STATistic.Func	Function runtime analysis	84
Port.STATistic.Func	Function runtime analysis	84
Port.STATistic.GROUP	Group run-time analysis	85
Port.STATistic.Ignore	Ignore false records in statistic	85

Port.STATistic.Line	HLL-Line analysis	85
Port.STATistic.LINKAge	Linkage analysis	85
Port.STATistic.PreFetch	Prefetch detection	85
Port.STATistic.Sort	Sort statistic results	85
Port.STATistic.sYmbol	Flat run-time analysis	85
Port.STATistic.TASK	Task run-time analysis	86
Port.STATistic.TASKFunc	Task specific function run-time analysis	86
Port.STATistic.TASKFunc	Task specific function run-time analysis	86
Port.STATistic.TASKKernel	Task run-time analysis (KENTRY/KEXIT)	86
Port.STATistic.TASKSRV	Analysis of time in OS service routines	86
Port.STATistic.TASKState	Performance analysis	86
Port.STATistic.TASKTREE	Tree display of task specific functions	86
Port.STATistic.TREE	Tree display of function run-time analysis	87
Port.STATistic.Use	Use records	87
Port.TEST	Init and arm	87
Port.TMode	Select trigger mode	87
Probe		88
Trace Methods		88
Method Probe		89
PULSE		89
Function		90
PULSE.PERiod	Cycle duration	92
PULSE.Pulse	Programming	93
PULSE.RESet	Reset command	94
PULSE.Single	Release single pulse	94
PULSE.state	State display	95
PULSE.Width	Pulse width	96
PULSE2		96
Function		97
PULSE2.Pulse	Programming	97
PULSE2.RESet	Reset command	99
PULSE2.Single	Release single pulse	99
PULSE2.state	Status display	99
PULSE2.Width	Pulse width	100

Usage:

- (B) command only available for ICD
- (E) command only available for ICE
- (F) command only available for FIRE

- 10/28/09 The peripheral file programming commands are moved to "**Peripheral Files Programming Commands**" (per_prog.pdf).
- 07/08/10 **PERF** commands updated to correspond with the new concepts for the Performance Analyzer.

PCP

PCPOnchip

This 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.

Function

The peripherals of integrated microcontrollers can be displayed and manipulated with the command **PER**. The command offers a free configurable window for displaying memory or i/o structures. So it is possible to display the state of peripheral chips or memory based structures very comfortably.

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.

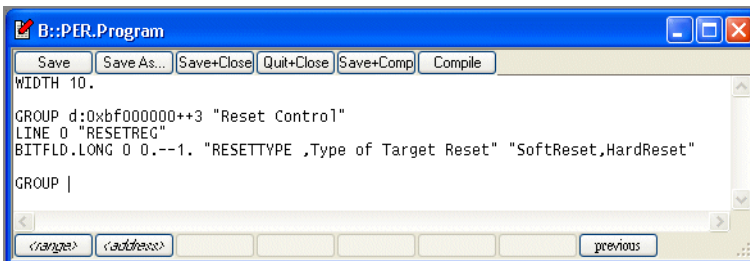
Examples for different microcontrollers reside in the directory `.../demo/per`.

PER.Program

Interactive programming

Format: **PER.Program** [*<filename>*]

This function offers an editor with on-line syntax check to create a definition file. The input is guided by soft keys. The [syntax for the definition file](#) is described below.



See also

- [PER.ReProgram](#)
- [PER.view](#)
- [IOBASE\(\)](#)

'Register and Peripherals' in 'ICE User's Guide'

'Release Information' in 'Release History'

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

Without parameter the default filename in the system directory is used (e.g. per68070.per). With parameter the corresponding file is compiled. The file should not have any errors, when using this command. The peripherals can be displayed with the **PER.view** command without arguments.

See also

■ [PER.Program](#) ■ [PER.view](#) □ [IOBASE\(\)](#)

'Register and Peripherals' in 'ICE User's Guide'

'Release Information' in 'Release History'

PER.Set

Modify memory

Format: **PER.Set** *<address>* %*<format>* *<value>* [*</option>*]

format: **Byte | Word | Long | Quad | TByte | TWord | BE | LE**

options: **Verify | ComPare**

Modifies data memory. Usually appears in the command line after a double click on a register in the peripheral view window. See [Data.Set](#) for details on modify memory.

See also

■ [PER.view](#)

Format:	PER.Set.Field <address> %<format> <mask> [<mult> [<summ>]]<value>
format:	Byte Word Long Quad TByte TWord 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 <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
                                     --         ---  --
```

$$\text{NewData} = (\text{OldData} \& \sim\text{mask}) \mid ((\text{value} \ll \text{offset}(\text{mask})) \& \text{mask})$$

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;
```

Example1: The following profile 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.view](#)

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 | TWord | BE | LE**

Write or modify indirect addressed registers. <idx_addr> specifies the address register and <data_addr> specifies the address if the data register of the indirect access.

PER.Set.Index can be translated into 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 Modify indirect (indexed) register**" (general_ref_p.pdf), to restore the original setting after the access.

See also

■ [PER.view](#)

PER.Set.Out

Write data stream to memory

Format: **PER.Set.Out** <address> %<format> [data]

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

See also

■ [PER.view](#)

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 | TWord | BE | LE**

Write or modify indirect addressed registers. <idx_addr> specifies the address register and <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> can not be read (write only), use **"PER.Set.Index Modify indirect (indexed) register"** (general_ref_p.pdf).

See also

■ [PER.view](#)

Format: **PER.Set.SHADOW** <address1> <address2> %<format> <value>

format: **Byte | Word | Long | Quad | TByte | TWord | BE | LE**

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

See also

- [PER.view](#)

PER.Set.simple

Modify memory

Format: **PER.Set** <range> <address> [/<option>]

options: **Verify | ComPare**

Modifies data memory. Usually appears in the command line after a double click on a register in the peripheral view window. See [Data.Set](#) for details on modify memories. The command is equal to PER.Set.

See also

- [PER.view](#)

'Registers' in 'Training FIRE Basics'

'Registers' in 'Training ICD Basics'

'Registers' in 'Training ICE Basics'

Format: **PER.view** [*<filename>*] [*<tree-search-item>*]

Without parameter the default definition file is used. With parameter a configuration file is compiled and displayed. The optional search parameter can be used to search for a specific tree item containing the string and open it on display.

E68: :w.per															
TIMER															
TSR	80	OV0	Yes	MA1	No	CA1	No	OV1	No	MA2	No	CA2	No	OV2	No
TCR	00	EV1	Input	Inh.		M1	Timer	Inh.		EV2	Input	Inh.		M2	Time
RR	0000														
T0	A1D9														
T1	0000														
T2	0000														
PICR															
CR1	00	PIR		IPL	IPL0					PIR		IPL	IPL0		
CR2	00	PIR		IPL	IPL0					PIR		IPL	IPL0		
DMA1															
CSR	01	COC	No			NDT	No	ERR	No	CA	No				
CER	00								EC	None					
DCR	30	ERM	Bur			DT	A/R			DS	8				
OCR	02	Dir	M>D	OS	Byte										

```
per ; display default peripherals
per dma68430 ; display of data with file dma68430.per
```

See also

- [PER.Program](#)
- [PER.ReProgram](#)
- [PER.Set](#)
- [PER.Set.Field](#)
- [PER.Set.Index](#)
- [PER.Set.Out](#)
- [PER.Set.SaveIndex](#)
- [PER.Set.SHADOW](#)
- [PER.Set.simple](#)

'Register and Peripherals' in 'ICE User's Guide'

'Release Information' in 'Release History'

'Commands' in 'C166 Family Trace'

'Registers' in 'Training FIRE Basics'

'Registers' in 'Training ICD Basics'

'Registers' in 'Training ICE Basics'

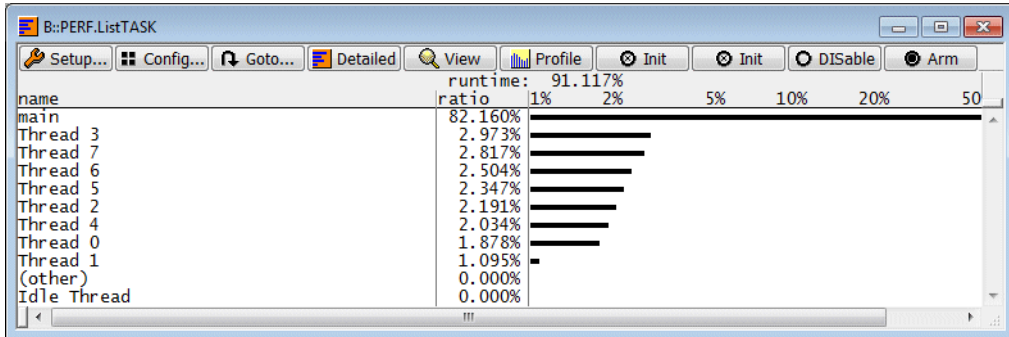
Programming Commands

For a description of the peripheral file [programming commands](#) refer to "[Peripheral Files Programming Commands](#)" (per_prog.pdf).

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.

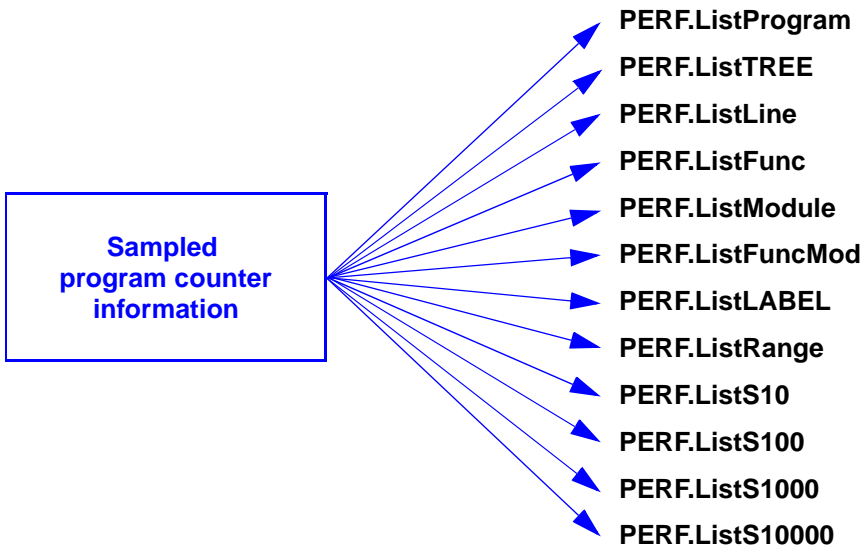


Samples are collected periodically. TRACE32 starts normally with 100 samples/s, but TRACE32's sample acquisition methods are auto-adaptive. They tune the sampling rate to its optimum.

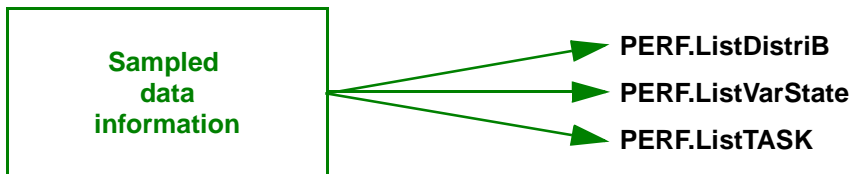
TRACE32 supports several sample acquisition methods. Some have no or nearly no effect on the target's run-time behavior but require special features from the on-chip debug logic (Snoop, Trace, DCC). The acquisition method **StopAndGo** is always supported, but has some impact on the target's run-time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

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

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 memorylocation

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

```
PERF.SnoopAddress V.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
```

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 V.RANGE(sieve)      PERF.ListFunc /Address V.RANGE(sieve)
PERF.ListFunc
```

The following example restricts the sample-based profiling 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 V.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.state](#)

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

The range definitions of the performance analyzer are normally restricted to program fetches. Data operations will not cause the analyzer to account for the data range. This behavior can be changed when **ANYACCESS** is activated. The results are also affected by data operations and will reflect more an access histogram than a performance analysis.

See also

■ [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.state](#)

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

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

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.state](#)

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

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.state](#)

Format:	PERF.Display <i><item></i>
<i><item></i> :	Program TREE LINE Function Module FuncMod LABEL S10 S100 S1000 S10000 DistriBution VarState

tbd.

PERF.Entry

Function runtime analysis

ICE only

Format:	PERF.Entry [OFF ON]
---------	------------------------------

As the analyzer detects accesses to address ranges and the number of passes to that ranges, it is usually not possible to get the average run time of a function. The analyzer will display the mean time spent in a function. When the **Entry** option is switched on, the analyzer tries to calculate the run time of a function with a special method. Each function range is split into two ranges, a short range at the function entry and a long range at the rest of the function. The number of passes in the function header will give the number of function calls and allows to calculate the average run time. To work correctly the header of a function must execute linear for some program cycles, otherwise the number of entries and the average times will be wrong. The size of this header can be adjusted with **PERF.EntrySize**.

See also

- [PERF.state](#)

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

PERF.EntrySize

Function header size

ICE only

Format:	PERF.EntrySize <i><bytes></i>
---------	--

This definition will be used if **PERF.Entry** is activated. It defines the size of the function header. A too small value will cause the performance analyzer to ignore entries, and result in a too small number of entries and a too large average time. This will occur if the time to fetch these bytes is **smaller than 1 µs**. A too large value will also cause errors, when header part is not executed linear, i.e. has jumps or calls inside. This calls

will trigger the passed counter of the header range and cause a too large entry number and a too small average time. The best results will be gained, if the value is chosen as small as possible, but large enough that the fetches take more than 1 μ s (check with the state analyzer and time stamps).

See also

- [PERF.state](#)

PERF.Filter

Suppress display of items with specified characteristic

Format:	PERF.Filter.SET ZEROS PERF.Filter.RESet
---------	--

PERF.Filter.SET ZEROS suppresses items with Ratio 0%.

```
; store the result of function profiling to a file, but
; suppress items with 0% Ratio

PERF.ListFunc                               ; open a window for function
                                           ; profiling

PERF.Filter.SET ZEROS                       ; suppress the display of functions
                                           ; with Ratio 0%

PRinTer.FILE result1                       ; specify a file name

WinPrint.PERF.ListFunc                     ; send the function profiling
                                           ; result to the specified file

TYPE result1.txt                            ; type the file contents
```

PERF.Filter.RESet resets all filters.

Format: **PERF.Gate** *<time>* (deprecated)

This command has no function. It is only available to guarantee the operation of existing PRACTICE scripts.

See also

■ PERF.state

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

Format: **PERF.Init**

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

See also

■ PERF.state

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

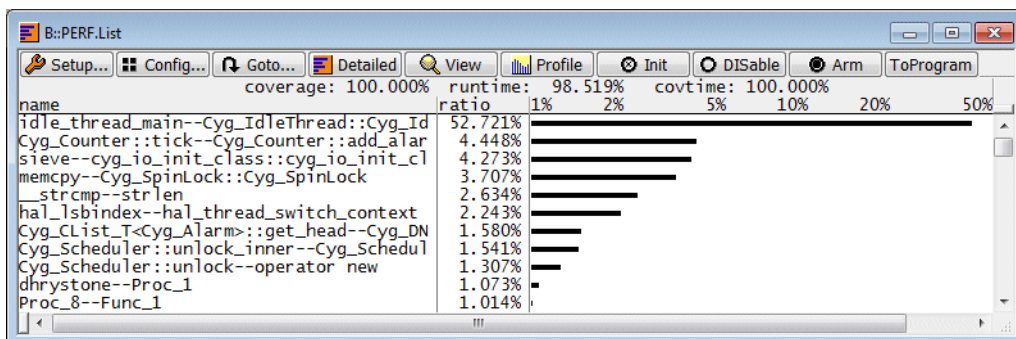
Format: **PERF.List** [*<column>* ...] [*/<option>*]

<column>:
DEfault
DYNamic
ALL
Name
Time
WatchTime
AVeRage (E)
DAVeRage (E)
Ratio
DRatio
BAR [.log | .LIN]
DBAR [.log | .LIN]
Passes (E)
Entrys (E)
Hits
Address
Break (E)

<option>: **Track | Address <range> | <address>**

Default profiling displays:

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



Interpretation of the result:

coverage: 100.000% runtime: 98.519% covtime: 100.000%

coverage	(ICE only) otherwise 100%
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.
covtime	(ICE only) otherwise 100%

columns	
name	Name of the item (here label range)
ratio	Ratio of time spent by the item in percent
bar	Logarithmic bar for the ratio

name	time	watchtime	ratio	dratio	address	hits
idle_thread_main--Cyg_ThreadPool::Cyg_Th	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:00031580--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.
_stricmp--strlen	2.520ms	104.568ms	2.410%	0.000%	P:000414F4--000416EF	135.
hal_lsbindx--hal_thread_switch_context	2.446ms	104.568ms	2.338%	0.000%	P:00041D20--00041D8B	131.
Cyg_List_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:0003BF84--0003BFC7	69.
hal_IRQ_handler--hal_interrupt_mask	1.045ms	104.568ms	0.999%	0.000%	P:000435B4--0004362F	56.
dhystone--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 item in percent
dratio	Ratio of time spent by item in the last second in percent
address	Item's address range or contents of the memory location
hits	Number of samples taken for the item

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, DRation, DBAR.log).
ALL	Display all possible numeric fields in the PERF.List window (columns: Name, Time, WatchTime, Ratio, DRatio, Address, Hits).

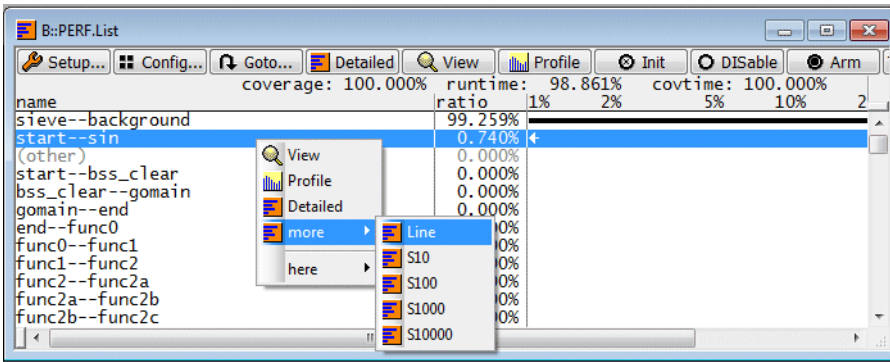
```
PERF.List Hits DEFAult           ; Open a PERF.List window starting with  
                                ; the column Hits followed by the  
                                ; default columns
```

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.TASK : 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.
AVeRage (ICE only)	The average time spent in listed item. This is either the average run time within the function, if the Entries value is not displayed, or the average time executed in the function, if Entries is displayed.
DAVeRage (ICE only)	Similar to above, but only for the last measurement interval (dynamic).
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.
BAR, DBAR	Display the profiling values in a graphical way as horizontal bars. The default display is logarithmic. The keyword .LIN changes to a linear display.
Passes (ICE only)	Number of entries in a range. NOTE: This is not the number of calls of a function. This value is also incremented, when another range is called from this range and the processor returns to that range.
Entrys (ICE only)	Number of entries in a range. This value will be displayed only, if PERF.Entry is switched to ON . You should always observe the entry code of the range to ensure proper operation.
Hits	Number of samples taken for the item.
Address	Item's address range or contents of the memory location.
Break (ICE only)	Display of breakpoints which are in use of the performance analyzer. If there are not all breakpoints in use, it will be possible to use other breakpoints for triggering. The performance analyzer will recognize them as another area for measuring.

Buttons in PERF.List window	
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.

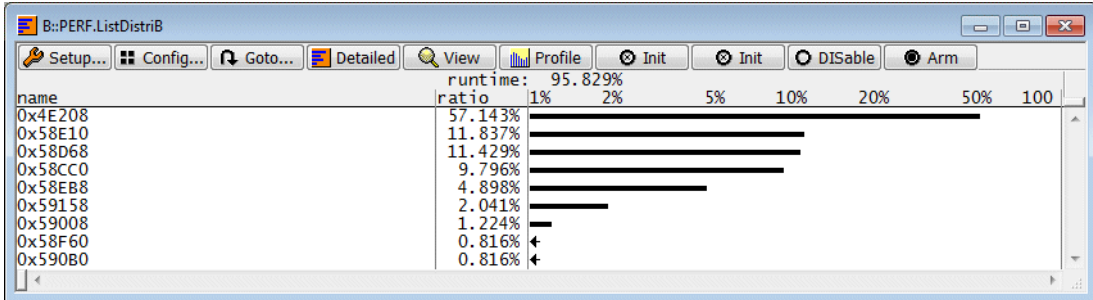
See also

■ [PERF.state](#)

- ‘Emulator Functions’ in ‘FIRE User’s Guide’
- ‘Performance Analysis’ in ‘ICE Performance Analyzer User’s Guide’
- ‘Release Information’ in ‘Release History’

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

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



```

; 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

```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

PERF.ListFunc

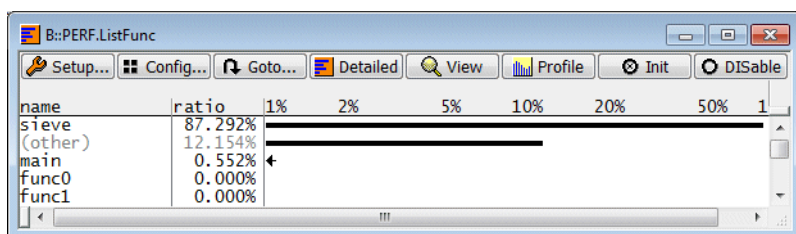
Function profiling

Format: **PERF.ListFunc** [*<column> ...*] [*/!<option>*]
(program counter sampling)

<option> **Track** | **Address** *<range>* | *<address>*

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).



```

; 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

```


A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

See also

■ [PERF.state](#)

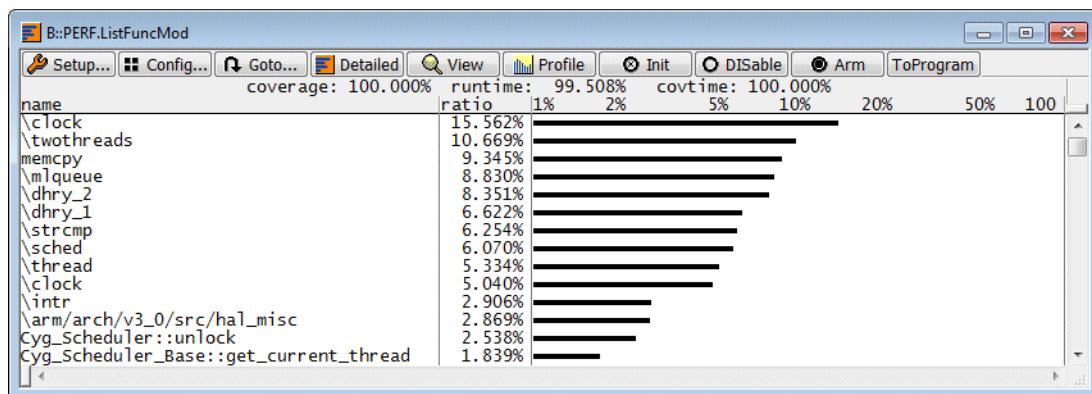
PERF.ListFuncMod

HLL function profiling (restricted)

Format: **PERF.ListFuncMod** [*<column>* ...] [*/<option>*]
(program counter sampling)

<option> **Track | Address** *<range>* | *<address>*

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.



; 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

```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

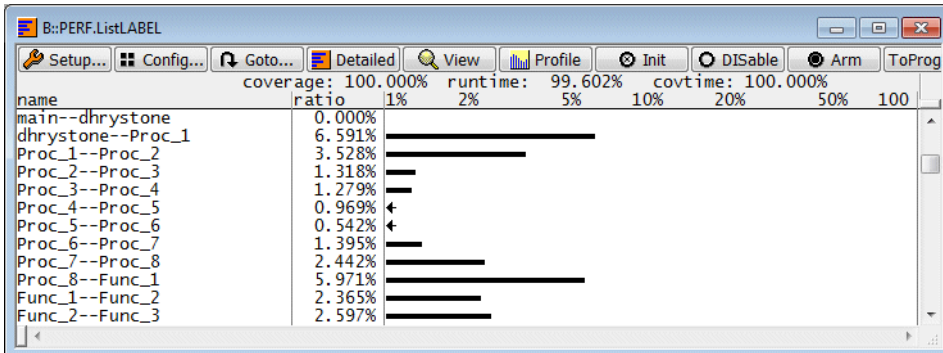
See also

- [PERF.state](#)

Format: **PERF.ListLABEL** [*<column> ...*] [*/<option>*]
(program counter sampling)

<option> **Track | Address** *<range>* | *<address>*

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



```

; 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 data base

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

Go ; start the program execution and
   ; sampling

```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-

down in give in the description of the [PERF.List](#) command.

See also

■ [PERF.state](#)

PERF.ListLine

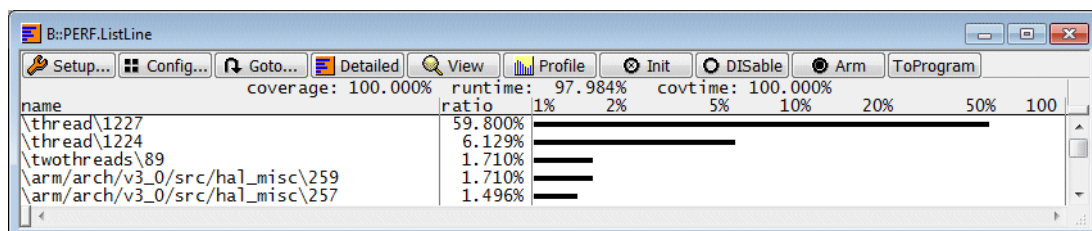
Profiling by hll lines

Format: **PERF.ListLine** [*<column>* ...] [*/<option>*]
(program counter sampling)

<option> **Track | Address** *<range>* | *<address>*

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

If the sampled program counter can not 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.



A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

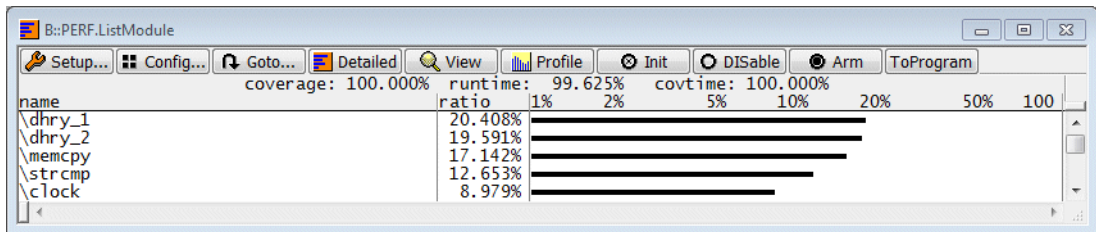
See also

■ [PERF.state](#)

Format: **PERF.ListModule** [*<column> ...*] [*/<option>*]
(program counter sampling)

<option> **Track | Address** *<range>* | *<address>*

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



A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

See also

■ [PERF.state](#)

PERF.ListProgram

Profiling based on Performance Analyzer program

Format: **PERF.ListProgram** [*<column> ...*] [*/<option>*]
(program counter sampling)

<option> **Track | Address** *<range>* | *<address>*

Reports the percentage of run-time spent in the address ranges specified by the Performance Analyzer program. A complete example on how to work with a Performance Analyzer program is give 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 in give in the description of the [PERF.List](#) command.

See also

■ [PERF.state](#)

Format:	PERF.ListRange [<i><column> ...</i>] [<i>!<option></i>] (program counter sampling)
<i><option></i>	Track Address <i><range></i> <i><address></i>

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

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

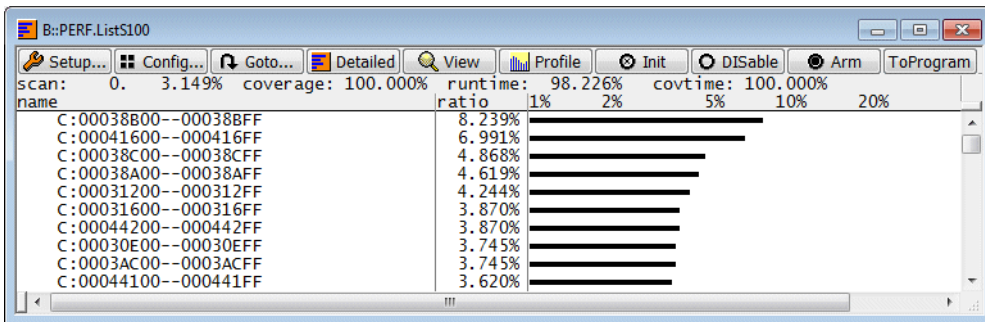
See also

- [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>

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



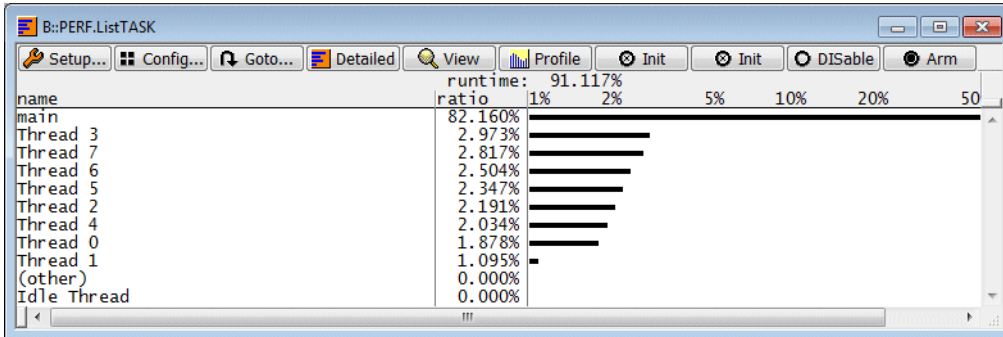
A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

See also

- [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.



```

; 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

```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

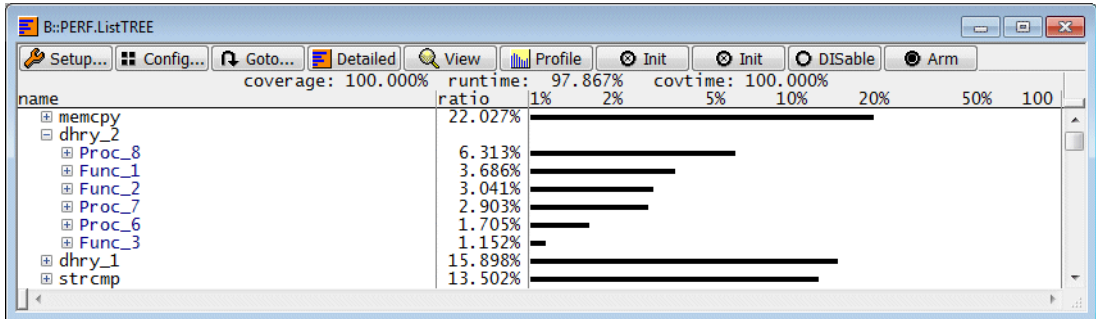
See also

- [PERF.state](#)

Format: **PERF.ListTREE** [*<column> ...*] [*/<option>*]
(program counter sampling)

<option> **Track | Address** *<range>* | *<address>*

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 data base.



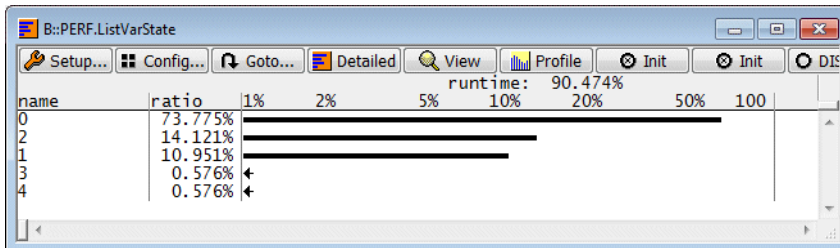
A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

See also

- [PERF.state](#)

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

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



```

; 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 V.RANGE(sched_Lock)    ; specifies the address range
                                           ; of the variable

PERF.SnoopSize Long                       ; specifies the sampling width

PERF.ListVarState                         ; open a window for variable
                                           ; profiling

Go                                         ; start the program execution
                                           ; and sampling

```

A detailed description of all display columns, all options, all window-specific buttons and the context pull-down in give in the description of the [PERF.List](#) command.

See also

- [PERF.state](#)

Format:	PERF.METHOD <mode>
<mode>:	StopAndGo Trace Snoop DCC (only if JTAG interface provides Data Communications Channel) Hardware (E) BusSnoop (E,F)

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

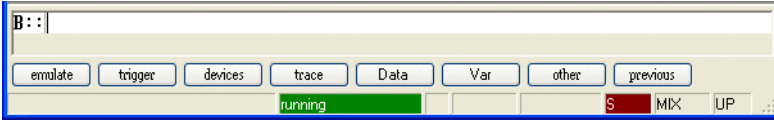
- If the processor allows to read the program counter while the program execution is running and **PERF.Mode PC** is selected.
- If the processor allows to read the contents of a memory locations while the program execution is running and **PERF.Mode MEMory** or **TASK** is selected.

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 46 (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). For details refer to “The Method Snoop” in General Commands Reference Guide P, page 47 (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 shortly to get a big enough section of the most recent trace information (non-intrusive). For details refer to “The Method Trace” in General Commands Reference Guide P, page 52 (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 56 (general_ref_p.pdf).

The method StopAndGo is available for all processors.

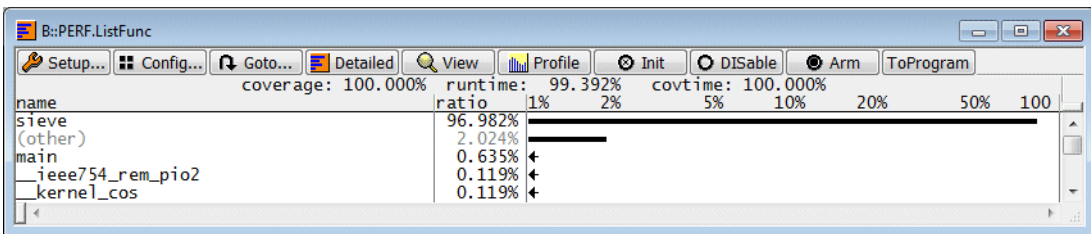
The target processor is stopped periodically in order to get the actual program counter or in order to read the data information of interest. The target processor is restarted afterwards. A stop and restart of the target processor can take more than 1 ms in a worst case scenario.



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.



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.

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

Processor architecture that allow to read the program counter while the program execution is running

TMS320C28xx TMS320C54xx TMS320C55xx TMS320C62xx TMS320C64xx TMS320C67xx	
TriCore	

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	
MPC55xx/56xx	Via NEXUS block
S12X, MCS12, 68HC12	
SH2/SH2A	
TMS320C28xx TMS320C54xx TMS320C55xx TMS320C62xx TMS320C64xx TMS320C67xx	
TriCore	
XC2000/C166S V2	
ZSP500	Debug Emulation Unit only

```

; 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

```

```

; 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

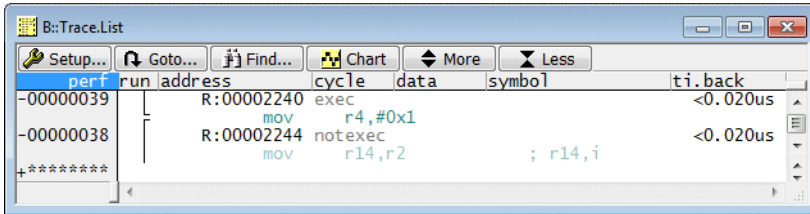
Go ; start the program execution and
; ; the measurement
```

The Method Trace

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.



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 source code to 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  
                                                ; broadcasted  
  
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 V.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
```

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 on how to use the DCC with TRACE32 are given in the example directory on the TRACE32 software DVD

```
~~\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
```


Hardware (ICE only)	A system of 64 (ECC8: 32) hardware counters is used to count the PC fetches in up to 64 (32) different ranges. 6 breakpoint types are needed to divide the 64 different ranges.
BusSnoop (ICE/FIRE only)	The PC fetch of the target CPU is read from the bus while the CPU is running. This fetch address is used to count the corresponding address range counter by software. If the fetch is outside any defined range, the “(other)” counter is incremented

See also

■ [PERF.state](#)

'Emulator Functions' in 'FIRE User's Guide'

PERF.MMUSPACES

tbd.

Format:	PERF.MMUSPACES [ON OFF]
---------	----------------------------------

Not implemented yet.

If a target operating system (e.g. Linux) is used that uses dynamic memory management to handle processes/tasks several processes/tasks can run at the same virtual addresses. In this szenario the virtual address sampled by the Performance Analyzer is not sufficient to assign the address to a function or variable. For a clear assignement the address space identifier is also required.

OFF (default)	The Performance Analyzer does standard sampling.
ON	The Performance Analyzer checks the address space identifier with every sample.

```
Format:          PERF.Mode <mode>

<mode>:         PC
                 TASK
                 MEMory
                 PCTASK
                 PCMEMory

                 LeVel (E,F)
                 FLAGs (E,F)
```

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 independent from each other.

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

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.state](#)

['Emulator Functions'](#) in ['FIRE User's Guide'](#)

['Performance Analysis'](#) in ['ICE Performance Analyzer User's Guide'](#)

['Release Information'](#) in ['Release History'](#)

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.state](#)

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

Because many processors have a prefetch mechanism, they read program areas, but never execute them. This causes the performance analyzer to display times for functions, that were never executed. To prevent this behavior, the ranges programmed have to be a little bit smaller than the defined value. This ensures, that prefetches do not cause the analyzer to count functions that never executed. The disadvantage is, of course, a measurement error caused by the too small range. The **PERF.PreFetch** command allows to select between the two modes. If activated (default) the ranges are shortened by the maximum number of prefetch cycles of the target processor.

See also

■ PERF.state

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

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.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   ; take the samples for the profiling
                          ; from the recorded trace information

PERF.Mode TASK            ; sample the program counter
                          ; information

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

PERF.ListFunc             ; assign the sampled program counter
                          ; information to the hll functions and
                          ; display the profiling

```

An opened window may be zoomed using the soft. Use the vertical auto zooming feature for best getting the best vertical resolution. The auto zoom is switched off by supplying a scale factor, manual zoom or vertical scrolling. The scale factor must be a power of 2.

```

PERF.state                ; Display the performance analyzer
                          ; configuration window

PERF.RESet                ; Reset the performance analyzer
                          ; configuration

PERF.Mode Function        ; Select the operation mode
                          ; function for the analysis

PERF.METHOD Trace       ; Use the trace as acquisition
                          ; method of the performance values

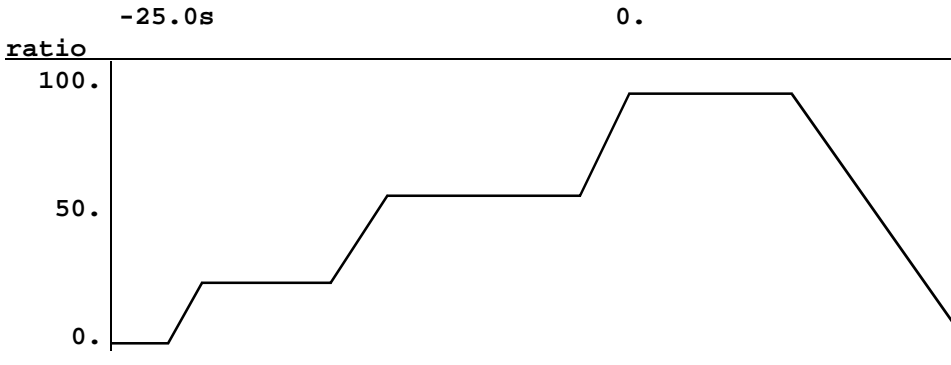
PERF.PROfile func2 func2B ; Display time chart for functions
                          ; func2 and func2B

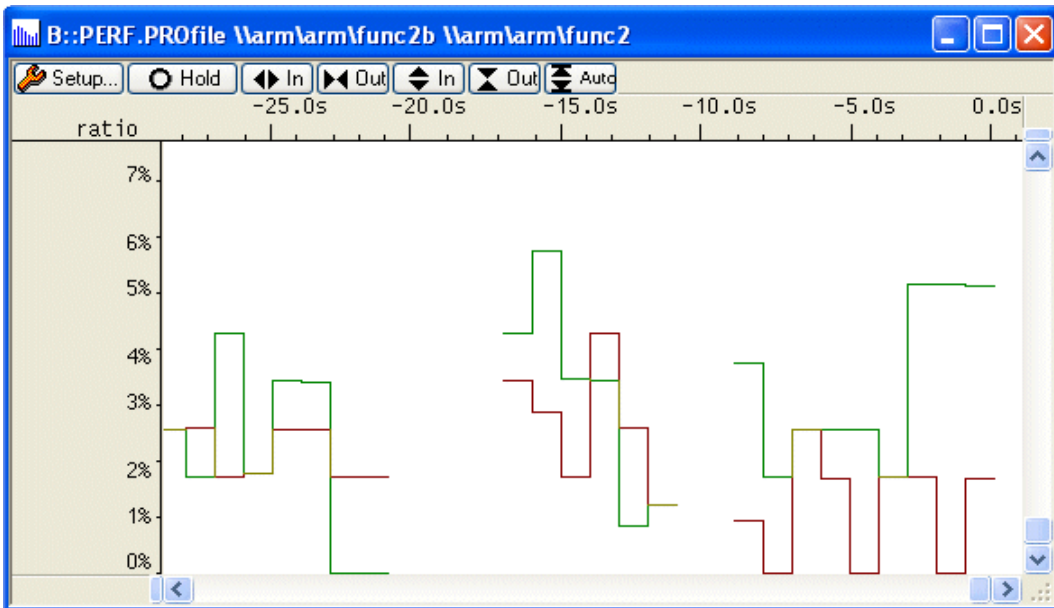
PERF.PROfile funcA funcB funcC ; time chart for funcA, funcB, funcC

PERF.PROfile 0x1A0--0x220 0x420-
0x2310 0x2000-0x3020     ; time chart for given address
                          ; ranges

```

E::PERF.PROfile





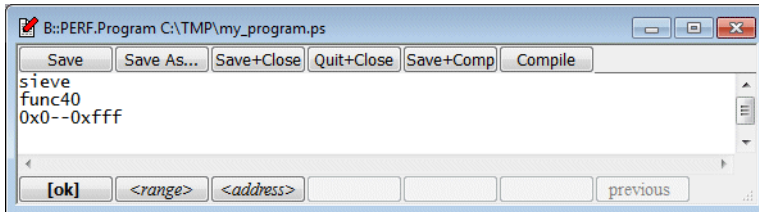
Buttons	
Zoom In T	Zoom in vertical axis by factor 2.
Zoom Out T	Zoom out vertical axis by factor 2.
Zoom In Y	Zoom in horizontal axis by factor 2.
Zoom Out Y	Zoom out horizontal axis by factor 2.
Hold	Stop Updating.
AutoZoom	Auto zooming of the vertical axis.

See also

- [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.



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.

```

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.state](#)

['Emulator Functions' in 'FIRE User's Guide'](#)

['Performance Analysis' in 'ICE Performance Analyzer User's Guide'](#)

['Release Information' in '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.state](#)

['Emulator Functions' in 'FIRE User's Guide'](#)

['Performance Analysis' in 'ICE Performance Analyzer User's Guide'](#)

['Release Information' in '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.state](#)

['Emulator Functions' in 'FIRE User's Guide'](#)

['Performance Analysis' in 'ICE Performance Analyzer User's Guide'](#)

```
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.

```
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.state](#)

PERF.SCAN

Scanning mode

ICE only

```
Format:          PERF.SCAN [OFF | ON]
```

When more ranges than available counters are covered and the **Ratio** sort mode is selected then the performance analyzer enters a scanning mode. In this mode the analyzer searches for the most time consuming areas. When these areas are found, it may be useful to disable the scanning and monitor only these ranges.

See also

■ [PERF.state](#)

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.state](#)

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

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

See also

■ [PERF.state](#)

Format:	PERF.Sort <i><mode></i>
<i><mode></i> :	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.state](#)

'Emulator Functions' in 'FIRE User's Guide'

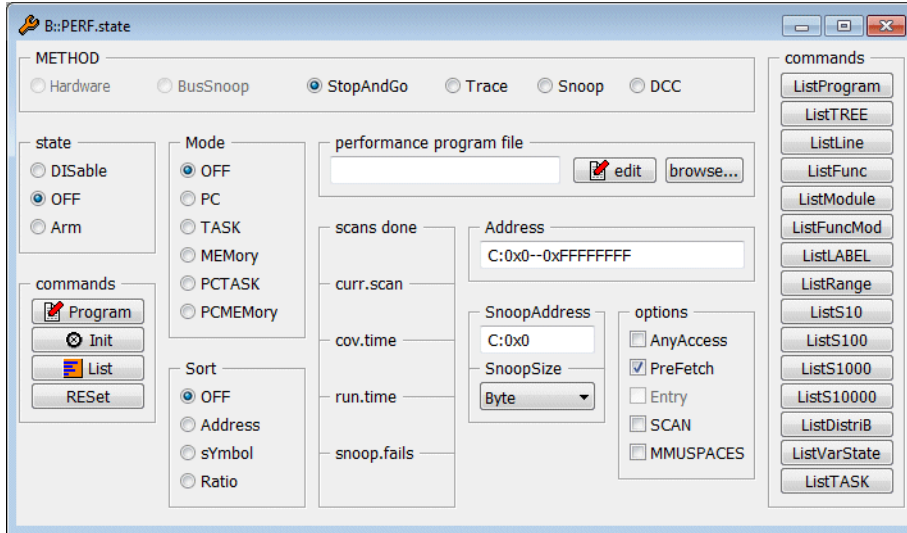
'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

Format: **PERF.state**

Displays control window for Performance Analyzer.



The field 'scans 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. The 'current scan' field displays the ratio of the scanned ranges to total the number of ranges. The 'covered time' field gives the time covered by the current set of ranges.

See also

- PERF.ADDRESS
- PERF.DISable
- PERF.Init
- PERF.ListFuncMod
- PERF.ListProgram
- PERF.ListTREE
- PERF.OFF
- PERF.ReProgram
- PERF.SnoopAddress
- PERF.View
- PERF.ANYACCESS
- PERF.Entry
- PERF.List
- PERF.ListLABEL
- PERF.ListRange
- PERF.ListVarState
- PERF.PreFetch
- PERF.RESet
- PERF.SnoopSize
- PERF.WATCHTIME()
- PERF.Arm
- PERF.EntrySize
- PERF.ListDistriB
- PERF.ListLine
- PERF.ListS10
- PERF.ListS100
- PERF.METHOD
- PERF.PROfile
- PERF.RunTime
- PERF.Sort
- PERF.AutoArm
- PERF.Gate
- PERF.ListFunc
- PERF.ListModule
- PERF.ListTASK
- PERF.Mode
- PERF.Program
- PERF.SCAN
- PERF.ToProgram

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

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

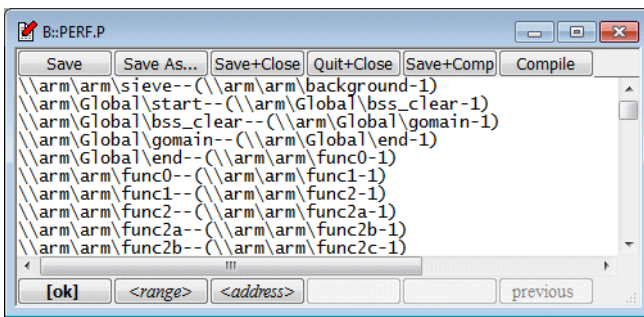
PERF.ListLABEL            ; open a window for label-based
                          ; profiling

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

```

See also

- [PERF.state](#)

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

Displays all numerical results of a symbol or an area.

```

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 V.RANGE(flags[3]) ; specify the memory address

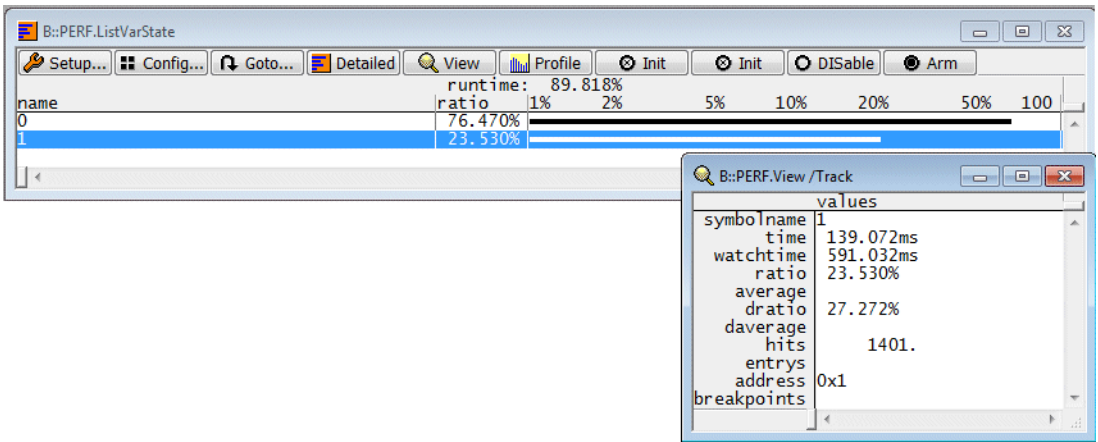
PERF.SnoopSize Byte ; specify the sampling width

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>

```



See also

■ PERF.state

'Emulator Functions' in 'FIRE User's Guide'

'Performance Analysis' in 'ICE Performance Analyzer User's Guide'

Format: **POD.Level** *<group>* *<level>*

<group>:

<level>: **1.0**
 1.4
 <variable value>

The PowerIntegrator has input probes with variable threshold level. Default is 1.4 V for all CMOS and TLL targets down to 2.5 V supply voltage.

See also

- [POD.state](#)

Format: **POD.RESet**

All input threshold levels are set to 1.4 V.

See also

- [POD.state](#)

Format: **POD.state**

PP : :POD			
0-15	0-1-2-3-4-5	Input	
1.0	—		0000000000000000
16-31	0-1-2-3-4-5	Input	
1.0	—		0000000000000000
32-47	0-1-2-3-4-5	Input	
1.4	—		0000000000000000
48-63	0-1-2-3-4-5	Input	
1.4	—		0000000000000000

See also

■ [POD.Level](#)

■ [POD.RESet](#)

NOTE:

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

Port.AutoFocus

Calibrate AutoFocus preprocessor

see command [<trace>.AutoFocus](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 51)

Port.AutoTEST

Continuous measurement

see command [<trace>.AutoTEST](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 56)

Port.BookMark

Set a bookmark in trace listing

see command [<trace>.BookMark](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 58)

Port.Chart.Func

Function activity chart

see command [<trace>.Chart.Func](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 73)

Port.Chart.GROUP

Group activity chart

see command [<trace>.Chart.GROUP](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 74)

see command [<trace>.Chart.Line](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 75)

see command [<trace>.Chart.sYmbol](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 78)

see command [<trace>.Chart.TASK](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 79)

see command [<trace>.Chart.TASKFunc](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 80)

see command [<trace>.Chart.TASKSRV](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 80)

see command [<trace>.Chart.TASKState](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 81)

see command [<trace>.Chart.VarState](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 82)

see command [<trace>.COVERage](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 87)

Port.COVERage.addAdd trace contents to database

see command [<trace>.COVERage.add](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 88)

Port.COVERage.DeleteCoverage modification

see command [<trace>.COVERage.Delete](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 89)

Port.COVERage.InitClear coverage database

see command [<trace>.COVERage.Init](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 89)

Port.COVERage.ListCoverage display

see command [<trace>.COVERage.List](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 90)

Port.COVERage.ListFuncDisplay coverage for HLL functions

see command [<trace>.COVERage.ListFunc](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 91)

Port.COVERage.ListModuleDisplay coverage for modules

see command [<trace>.COVERage.ListModule](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 92)

see command [<trace>.COVERage.ListVar](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 93)

see command [<trace>.COVERage.LOAD](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 94)

see command [<trace>.COVERage.RESet](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 94)

see command [<trace>.COVERage.SAVE](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 94)

see command [<trace>.COVERage.Set](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 95)

see command [<trace>.DisConfig.view](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 98)

see command [<trace>.DRAW](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 99)

see command [<trace>.Enable](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 99)

see command [<trace>.Enable](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 99)

see command [<trace>.FindAll](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 107)

see command [<trace>.MUX](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 137)

see command [<trace>.PROTOcol.Chart](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 141)

see command [<trace>.PROTOcol.Draw](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 143)

see command [<trace>.PROTOcol.EXPORT](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 144)

see command [<trace>.PROTOcol.Find](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 145)

see command [<trace>.PROTOcol.List](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 146)

see command [<trace>.PROTOcol.STATistic](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 149)

see command [<trace>.Select](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 161)

see command [<trace>.SET](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 163)

see command [<trace>.SLAVE](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 169)

see command [<trace>.STATistic](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 172)

see command [<trace>.STATistic](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 172)

Port.STATistic.BondOut

Bondout mode

see command [<trace>.STATistic.BondOut](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 176)

Port.STATistic.DIStance

Time interval for a single event

see command [<trace>.STATistic.DIStance](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 181)

Port.STATistic.DistriB

Distribution analysis

see command [<trace>.STATistic.DistriB](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 185)

Port.STATistic.DURation

Time between two events

see command [<trace>.STATistic.DURation](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 188)

Port.STATistic.Func

Function runtime analysis

see command [<trace>.STATistic.Func](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 193)

Port.STATistic.Func

Function runtime analysis

see command [<trace>.STATistic.Func](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 193)

see command [<trace>.STATistic.GROUP](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 218)

Port.STATistic.IgnoreIgnore false records in statistic

see command [<trace>.STATistic.Ignore](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 219)

Port.STATistic.LineHLL-Line analysis

see command [<trace>.STATistic.Line](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 223)

Port.STATistic.LINKageLinkage analysis

see command [<trace>.STATistic.LINKage](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 226)

Port.STATistic.PreFetchPrefetch detection

see command [<trace>.STATistic.PreFetch](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 230)

Port.STATistic.SortSort statistic results

see command [<trace>.STATistic.Sort](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 231)

Port.STATistic.sYmbolFlat run-time analysis

see command [<trace>.STATistic.sYmbol](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 232)

see command [<trace>.STATistic.TASK](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 236)

Port.STATistic.TASKFuncTask specific function run-time analysis

see command [<trace>.STATistic.TASKFunc](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241)

Port.STATistic.TASKFuncTask specific function run-time analysis

see command [<trace>.STATistic.TASKFunc](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241)

Port.STATistic.TASKKernelTask run-time analysis (KENTRY/KEXIT)

see command [<trace>.STATistic.TASKKernel](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 249)

Port.STATistic.TASKSRVAnalysis of time in OS service routines

see command [<trace>.STATistic.TASKSRV](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 252)

Port.STATistic.TASKStatePerformance analysis

see command [<trace>.STATistic.TASKState](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 252)

Port.STATistic.TASKTREETree display of task specific functions

see command [<trace>.STATistic.TASKTREE](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 256)

see command [<trace>.STATistic.TREE](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 258)

Port.STATistic.Use

Use records

see command [<trace>.STATistic.Use](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 259)

Port.TEST

Init and arm

see command [<trace>.TEST](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 262)

Port.TMode

Select trigger mode

see command [<trace>.TMode](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 270)

Trace Methods

The command **Trace** is a general command for trace display and configuration. It is available for all kind of trace methods provided by TRACE32.

The following trace methods are available:

- Analyzer
- ART
- FDX
- Integrator
- LogicAnalyzer
- LOGGER
- On-chip
- Probe
- PORT
- SNOOPer

The currently used trace method is displayed in the **METHOD** field of the [Trace.state](#) window.

The screenshot shows the **B::Trace.state** window. At the top, the **METHOD** field is set to **ART**. Below this, there are several configuration panels:

- state**: Includes **DISable** (checked), **OFF**, **Arm**, **trigger**, and **break**.
- used**: Shows the value **65535.**
- SIZE**: Shows the value **65535.**
- TDelay**: Shows **0.** and **0%**.
- ACCESS**: An empty field.
- Mode**: Includes **Fifo** (checked), **Stack**, **BusTrace**, **ClockTrace**, **FlowTrace** (checked), **Poststore**, **PostTrace**, and **SLAVE** (checked).
- commands**: Includes **RESet**, **Init**, **TEST**, **List**, **AutoArm** (checked), **AutoInit**, and **AutoTEST**.

The trace method **Probe** is mainly used for TRACE32-ICD without a trace extension.

Problem description: A TRACE32-ICD debugger is used to test and integrate an application program on the target. Now a problem occurs, that could easily be solved if more information about the program history would be available.

Usually a TRACE32-ICD trace extension can be used to get more information about the program history. But not all targets allow the operation of such a trace. For these targets TRACE32 is offering a software trace. The software trace however needs RAM from the target and is influencing the real-time behavior.

To operate a software trace, TRACE32 provides:

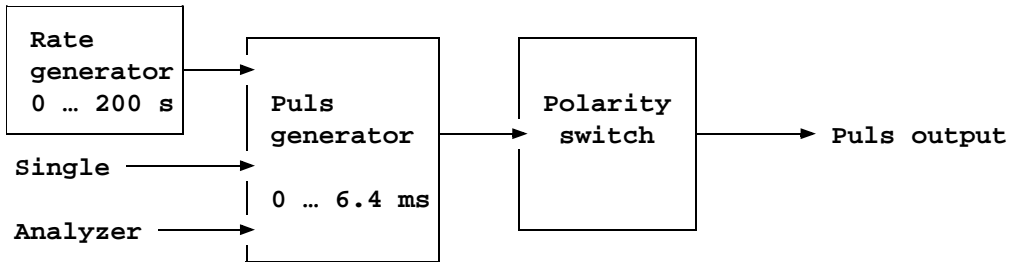
- a general trace format for a software trace located in the target RAM.
- configuration and display commands for the software trace in the TRACE32 software (command: `Probe`).
- predefined algorithms to operate the software trace from the target program.

To use the software trace basic knowledge of the target hardware and the processor architecture is required.

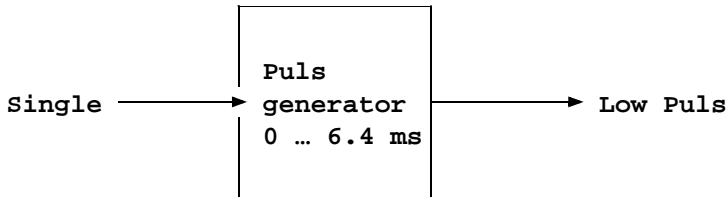
Implementation of the trace memory	The user reserves a part of the target RAM, that can be used for the trace information.
Max. trace size	Any desired.
Sampling	The trace memory is filled either by an algorithm predefined by LAUTERBACH or by a user defined algorithm. The algorithm can either be called by an interrupt or the code has to be instrumented.
Influence on the real-time behavior	Yes, how much depends on the implementation of the sampling algorithm.
Selective tracing	Possible by the sampling algorithm.
Fastest sampling rate	Depends on the sampling algorithm.

Function

The pulse generator is an independent system for generating short pulses or static signals, like used for stimulation in the target system or to reset the target hardware. The output pin of the generator is placed on the output probe of the ECU module. The triggering may occur periodically, manually by the keyboard, or by the trigger unit of the analyzer. If no pulse generation is needed, the output line will be set to high or low by selecting the polarity of the pulse.

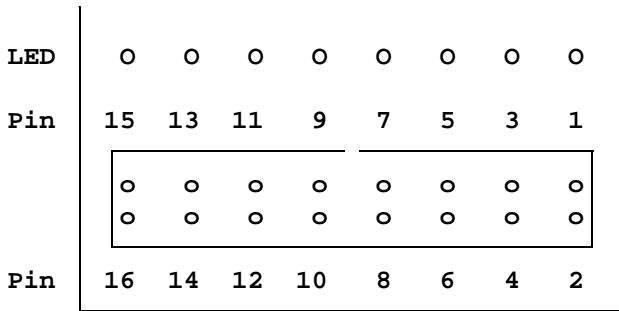


Puls Generator on ECU32



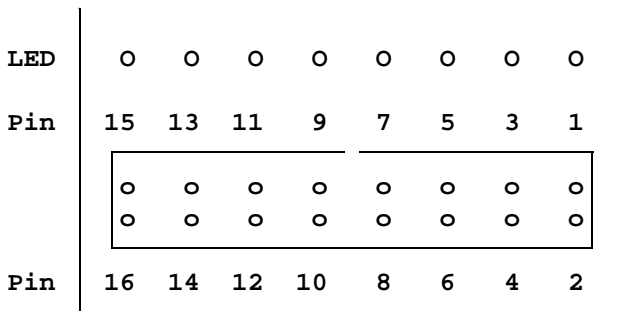
Puls Generator on ECC8

Pin assignment of the STROBE probe (ECU32)



Pin 1	Line 0	EVENT
Pin 3	Line 1	TriggerAddress
Pin 5	Line 2	RUN-(Foreground)
Pin 7	Line 3	TRIGGER
Pin 9	Line 4	SIGNAL
Pin 11	Line 5	RUNCYCLE-
Pin 13	Line 6	PULSE2
Pin 15	Line 7	PULSE
Pin 2,4,6,8,10,12,14,16	Ground	

Pin assignment of the STROBE probe (ECC8)



Pin 1	Line 0	OUT.C
Pin 3	Line 1	OUT.D
Pin 5	Line 2	RUN-(Foreground)
Pin 7	Line 3	TRIGGER
Pin 9	Line 4	CharlyBreak
Pin 11	Line 5	RUNCYCLE-
Pin 13	Line 6	PULSE2
Pin 15	Line 7	PULSE
Pin 2,4,6,8,10,12,14,16	Ground	

Format: **PULSe.PERiod** *<width>* | ON | OFF

<width>: **0.4us ... 200.s**

On ECC8 the period is limited to 6.5 ms. The pulse width is automatically set to half of the period time or 100ns on ECC8.

```
pulse.per off           ; set pulse generator to single pulse mode
pulse.per on            ; set pulse generator to periodic pulse
                        ; mode
pulse.per 1.ms          ; activate periodic mode, cycle duration
                        ; is 1 ms (1 kHz)
```

See also

■ [PULSE.Pulse](#)

■ [PULSE.state](#)

■ [PULSE.Width](#)

'Emulator Functions' in 'FIRE User's Guide'

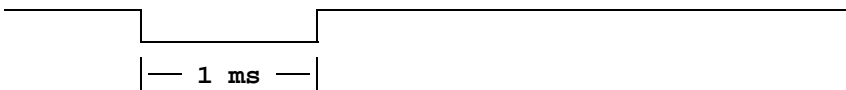
'Pulse Generator' in 'ICE User's Guide'

Format:	PULSE.Pulse [<i><width></i>] [<i><period></i>] [<i><polarity></i>]
<i><width></i> :	0.1us ... 6.4ms
<i><period></i> :	0.4us ... 200.s ON OFF
<i><polarity></i> :	+ -

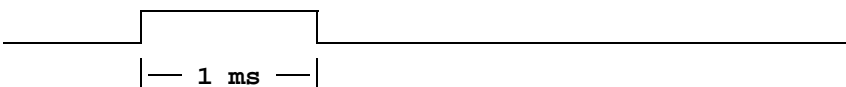
On ECC8 the period is limited to 6.5 ms. The pulse width is automatically set to half of the period time or 100ns on ECC8.

```
pulse.pulse 100.us 1.ms -      ; Pulse active low, 100 µs, 1 kHz
pulse.pulse 100.us +          ; Single pulse 100 µs, active high
pulse -                        ; active low pulse
pulse off                      ; switch off
pulse -                        ; set output to high level
...
pulse +                        ; set output to low level
```

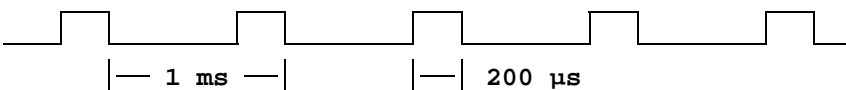
Puls 1.ms -



Puls 1.ms +



Puls 200.us 1.ms +



See also

[PULSE.PERiod](#)

 [PULSE.Single](#)

 [PULSE.state](#)

 [PULSE.Width](#)

'Emulator Functions' in 'FIRE User's Guide'
 'Pulse Generator' in 'ICE User's Guide'

Format: **PULSE.RESet**

See also

- [PULSE.state](#)

'Emulator Functions' in 'FIRE User's Guide'

'Pulse Generator' in 'ICE User's Guide'

PULSE.Single

Release single pulse

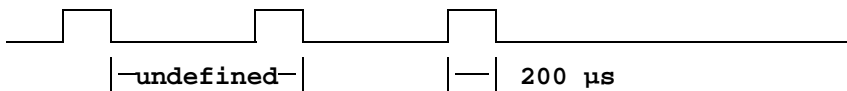
Format: **PULSE.Single** [*<count>*]

<count>: 1 ...

Releasing more than one single pulse occurs under software control, i.e. the time between two pulses is not constant.

```
pulse.s ; Release single pulse
pulse.s 3. ; Release threefold pulse
```

```
Puls.Width 200.us
Puls.Single 3.
```



See also

- [PULSE.Pulse](#)

- [PULSE.state](#)

'Emulator Functions' in 'FIRE User's Guide'

'Pulse Generator' in 'ICE User's Guide'

Format: **PULSE.state**

Display state of the pulse generator.

<u>E68::w.pulse</u>	
Pulse	
Single	
	+
√	-
Width	
1.000 μ s	
Period	
100.000 μ s	

Button for single pulse
Polarity

Pulse width

Cycle duration for periodic release

```
E68::
width: 1.000  $\mu$ s period:100.000  $\mu$ s pol: -
```

See also

- [PULSE.PERiod](#)
- [PULSE.Pulse](#)
- [PULSE.RESet](#)
- [PULSE.Single](#)
- [PULSE.Width](#)

'Emulator Functions' in 'FIRE User's Guide'

Format: **PULSE.Width** *<width>*

<width>: **0.4us ... 25.0ms**

The pulse width is limited to 6.5 ms on the ECC8. Periodical pulses can only be 100 ns or 50% ratio on the ECC8.

```
pulse.width 20.u ; Set pulse width to 20 μs
```

```
pulse.w 5.ms ; Set pulse width to 5 ms
```

See also

■ [PULSE.PERiod](#) ■ [PULSE.Pulse](#) ■ [PULSE.state](#)

'Emulator Functions' in 'FIRE User's Guide'

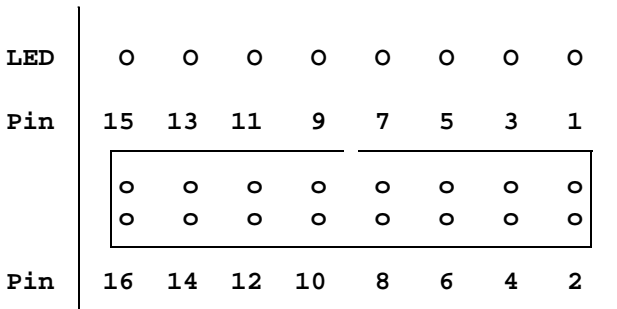
'Pulse Generator' in 'ICE User's Guide'

'Pulse Generator' in 'ICE User's Guide'

Function

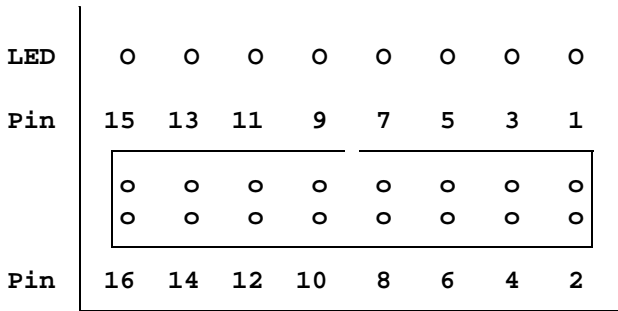
The pulse generator 2 is an independent system for generating short pulses. The output pin of the generator is placed on the output probe of the ECU module. This pulse generator is software controlled, the pulse periods may not match exactly. Mainly this output may be used as an reset signal for the target system. If no pulse is needed, but a signal which may be programmed to fixed levels, this may be done by setting the polarity (+ = LOW, - = HIGH).

Pin assignment of the STROBE probe (ECU32)



Pin 1	Line 0	OUT.C
Pin 3	Line 1	OUT.D
Pin 5	Line 2	RUN-(Foreground)
Pin 7	Line 3	TRIGGER
Pin 9	Line 4	CharlyBreak
Pin 11	Line 5	RUNCYCLE-
Pin 13	Line 6	PULSe2
Pin 15	Line 7	PULSe
Pin 2,4,6,8,10,12,14,16	Ground	

Pin assignment of the STROBE probe (ECC8)



Pin 1	Line 0	OUT.C
Pin 3	Line 1	OUT.C
Pin 5	Line 2	RUN-(Foreground)
Pin 7	Line 3	TRIGGER
Pin 9	Line 4	SIGnal
Pin 11	Line 5	RUNCYCLE-
Pin 13	Line 6	PULSe2
Pin 15	Line 7	PULSe
Pin 2,4,6,8,10,12,14,16	Ground	

PULSE2.Pulse

Programming

ICE only

Format: **PULSE2.Pulse** [*<width>*] [*<polarity>*]

<width>: **10.0us ... 25.0ms**

<polarity>: **+ | -**

```
pulse2.pulse 100.us - ; Pulse active low, 100 µs
```

```
pulse2 - ; active low pulse
```

See also

■ [PULSE2.state](#)

Format:	PULSE2.RESet
---------	---------------------

See also

- [PULSE2.state](#)

PULSE2.Single

Release single pulse

ICE only

Format:	PULSE.Single2 [<i><count></i>]
---------	---

<i><count></i> :	1 ...
------------------------	--------------

The releasing of more than one single pulse occurs under software control, therefore the time between two pulses is not constant.

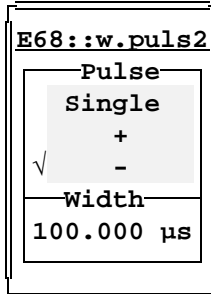
```
pulse2.s ; Release single pulse
pulse2.s 3. ; Release threefold pulse
```

See also

- [PULSE2.state](#)

Format: **PULSE2.state**

Display state of the second pulse generator.



Button for single pulse
Polarity

Pulse width

```
E68::
width:100.000 μs pol: -
```

See also

■ [PULSE2.Pulse](#)

■ [PULSE2.RESet](#)

■ [PULSE2.Single](#)

■ [PULSE2.Width](#)

PULSE2.Width

Pulse width

Format: **PULSE2.Width** *<width>*

<width>: **10.0us ... 25.0ms**

```
pulse2.width 20.us ; Set pulse width to 20 μs
```

```
pulse2.w 10.ms ; Set pulse width to 10 ms
```

See also

■ [PULSE2.state](#)