LAUTERBACH
DEVELOPMENT TOOLS

General Commands Reference Guide C

Release 09.2023

General Commands Reference Guide C

General Commands Reference Guide C

Version 09-Oct-2023

History

31-Jul-23	TriCore DAP streaming via AUTO26 V3 debug cable has been added as a configuration for the CAnalyzer command group.
18-Apr-23	Updated description of **COVerage.TreeWalkSETUP** and subcommands.
20-Mar-23	Added μTrace (MicroTrace) with MIPI34 whisker to the list of setups that support advanced AutoFocus features to match software since build 156270, DVD 09/2023.
24-Jan-22	Marked the command COVerage.StaticInfo as deprecated.
13-Jul-21	New command: CORE.SINGLE.
02-Mar-21	The object code coverage tagging details and the read/write coverage tagging have been moved to **"Application Note for Trace-Based Code Coverage"** (app_code_coverage.pdf).

CACHE — View and modify CPU cache contents

Using the **CACHE** command group, you can view and modify the CPU cache contents. Note that some targets support only a subset of the **CACHE.*** commands.

When you are trying to execute a command that is not supported for your target, TRACE32 displays the error message “unknown command”.

For targets without accessible CPU cache, the entire **CACHE** command group is locked.

See also

■ CACHE.CLEAN	■ CACHE.ComPare	■ CACHE.DUMP	■ CACHE.FLUSH
■ CACHE.GET	■ CACHE.INFO	■ CACHE.INVALIDATE	■ CACHE.List
■ CACHE.ListFunc	■ CACHE.ListLine	■ CACHE.ListModule	■ CACHE.ListVar
■ CACHE.LOAD	■ CACHE.RELOAD	■ CACHE.SAVE	■ CACHE.SNAPSHOT
■ CACHE.UNLOAD	■ CACHE.view		

▲ ’CACHE Functions’ in ’General Function Reference’

CACHE.CLEAN — Clean CACHE

Format:	**CACHE.CLEAN** *<cache>*
<cache>:	**IC** | **DC** | **L2**

Writes back modified (dirty) lines to the next cache level or memory. Only the specified cache is affected.

In case the operation is not supported by the CPU, the result will be a “function not implemented” error message.

See also

■ CACHE	■ CACHE.FLUSH	■ CACHE.INVALIDATE	■ CACHE.view

Format:	**CACHE.ComPare** *<cache>*
<cache>:	**IC** | **DC** | **L2**

Compares CACHE contents with memory contents.

Example:

```
CACHE.ComPare DC        ; compare contents of the data CACHE with the
                        ; memory
```

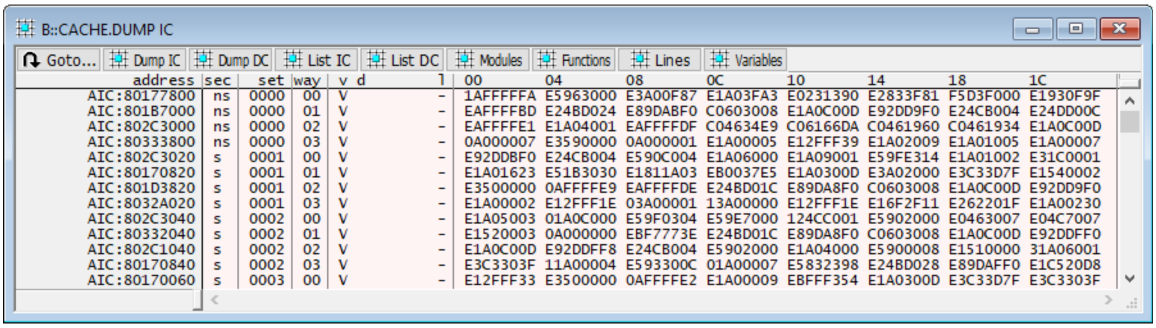
See also

■ CACHE ■ CACHE.view

▲ 'Release Information' in 'Legacy Release History'

Format:	**CACHE.DUMP** *<cache>* [**/***<options>*]
<cache>:	**IC** | **DC** | **L2**
<options>:	**ALL** | **RAW** | **ValidOnly**

Displays a hex dump of the CACHE contents. This command extracts useful information from the raw data read from the target and present them in a table in sequential order of the sets and ways. By default, only valid cache lines are presented.



```
          address sec   set way  v d            l  00       04       08       0C       10       14       18       1C
    AIC:80177800  ns  0000  00  V              -  1AFFFFFA E5963000 E3A00F87 E1A03FA3 E0231390 E2833F81 F5D3F000 E1930F9F
    AIC:801B7000  ns  0000  01  V              -  EAFFFFBD E24BD024 E89DABF0 C0603008 E1A0C00D E92DD9F0 E24CB004 E24DD00C
    AIC:802C3000  ns  0000  02  V              -  EAFFFFE1 E1A04001 EAFFFFDF C04634E9 C06166DA C0461960 C0461934 E1A0C00D
    AIC:80333800  ns  0000  03  V              -  0A000007 E3590000 0A000001 E1A00005 E12FFF39 E1A02009 E1A01005 E1A00007
    AIC:802C3020  s   0001  00  V              -  E92DDBF0 E24CB004 E590C004 E1A06000 E1A09001 E59FE314 E1A01002 E31C0001
    AIC:80170820  s   0001  01  V              -  E1A01623 E51B3030 E1811A03 EB0037E5 E1A0300D E3A02000 E3C33D7F E1540002
    AIC:801D3820  s   0001  02  V              -  E3500000 0AFFFFE9 EAFFFFDE E24BD01C E89DA8F0 C0603008 E1A0C00D E92DD9F0
    AIC:8032A020  s   0001  03  V              -  E1A00002 E12FFF1E 03A00001 13A00000 E12FFF1E E16F2F11 E262201F E1A00230
    AIC:802C3040  s   0002  00  V              -  E1A05003 01A0C000 E59F0304 E59E7000 124CC001 E5902000 E0463007 E04C7007
    AIC:80332040  s   0002  01  V              -  E1520003 0A000000 EBF7773E E24BD01C E89DA8F0 C0603008 E1A0C00D E92DDFF0
    AIC:802C1040  s   0002  02  V              -  E1A0C00D E92DDFF8 E24CB004 E5902000 E1A04000 E5900008 E1510000 31A06001
    AIC:80170840  s   0002  03  V              -  E3C3303F 11A00004 E593300C 01A00007 E5832398 E24BD028 E89DAFF0 E1C520D8
    AIC:80170060  s   0003  00  V              -  E12FFF33 E3500000 0AFFFFE2 E1A00009 EBFFF354 E1A0300D E3C33D7F E3C3303F
```

RAW	**Dump also the raw data.** If the option **RAW** is used, all cache lines, no matter valid or not, will be displayed.

The **CACHE.DUMP** window typically involves multiple columns, some of which are used to present architecture-specific attributes of the cache lines. In the following table, we describe some commonly presented attributes. Please refer to the design manual of the respective architecture to understand the detailed meaning of these attributes.

Attribute	**Description**
Valid	• Column Name: "v". • Value "V" : valid. • Value "-" : invalid.
Dirty	• Column Name: "d". • Value "D" : dirty. • Value "-" : not dirty.
Secure	• Column Name: "sec". • Value "s" : secure. • Value "ns" : non-secure.

Attribute	Description
Shared	• Column Name: “s”. • Value “S”: shared. • value “-”: non-shared.
Coherence	• Column Name: “c” • The possible values of this column depend on the cache coherence protocol used by the architecture. E.g, for the MOESI protocol: - Value “M” : modified. - Value “O” : owned. - Value “E” : exclusive. - Value “S” : shared. - Value “I” : invalid.

See also

■ CACHE ■ CACHE.view

▲ ’Release Information’ in ’Legacy Release History’

CACHE.FLUSH Clean and invalidate CACHE

Format:	**CACHE.FLUSH** *<cache>*
<cache>:	**IC** | **DC** | **L2**

Writes back modified (dirty) lines to the next cache level or memory and invalidate the entire cache. Only the specified cache is affected.

In case the operation is not supported by the CPU, the result will be a “function not implemented” error message.

See also

■ CACHE ■ CACHE.CLEAN ■ CACHE.INVALIDATE ■ CACHE.view

CACHE.GET

Format:	**CACHE.GET**

Synchronizes the TRACE32 software with the target on the entire cache. TRACE32 loads all cache lines for which it does not have up-to-date data. For diagnostic purposes only.

Previously loaded data are not explicitly reloaded, unless they are marked for reload by the **CACHE.RELOAD** command executed before **CACHE.GET**.

See also

- CACHE
- CACHE.RELOAD
- CACHE.view

CACHE.INFO

View all information related to an address

Format:	**CACHE.INFO.***<sub_cmd> <address>*
<sub_cmd>:	**create** | **scanSTART** | **scanRESUME** | **scanSTOP** **TaskPageTable** *<address> <task>*

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

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

create	Views all translation information related to an address.
scanSTART	Starts a scan in all MMU page tables for entries that contain translation rules for the physical address.
scanRESUME	Resumes the scan stopped with **scanSTOP**.
scanSTOP	Stops the scan.
TaskPageTable	Displays all translation information related to a give address and task page table. Refer to **MMU.INFO.TaskPageTable** for more information.

See also

- CACHE
- CACHE.view

CACHE.INVALIDATE Invalidate CACHE

Format:	**CACHE.INVALIDATE** *<cache>*
<cache>:	**IC** | **DC** | **L2**

Invalidates the entire cache. Only the specified cache is affected. In case the operation is not supported by the CPU, the result will be a “function not implemented” error message.

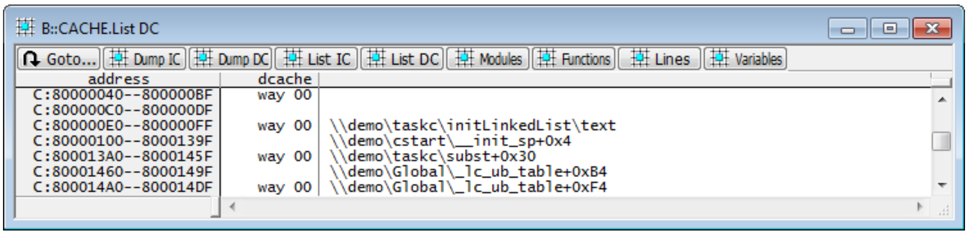
See also

- CACHE
- CACHE.CLEAN
- CACHE.FLUSH
- CACHE.view

CACHE.List List CACHE contents

Format:	**CACHE.List** *<cache>*
<cache>:	**IC** | **DC** | **L2**

Displays a list of the CACHE contents.



```
        address               dcache
  C:80000040--800000BF        way 00
  C:800000C0--800000DF
  C:800000E0--800000FF        way 00   \\demo\taskc\initLinkedList\text
  C:80000100--8000139F                 \\demo\cstart\__init_sp+0x4
  C:800013A0--8000145F        way 00   \\demo\taskc\subst+0x30
  C:80001460--8000149F                 \\demo\Global\_lc_ub_table+0xB4
  C:800014A0--800014DF        way 00   \\demo\Global\_lc_ub_table+0xF4
```

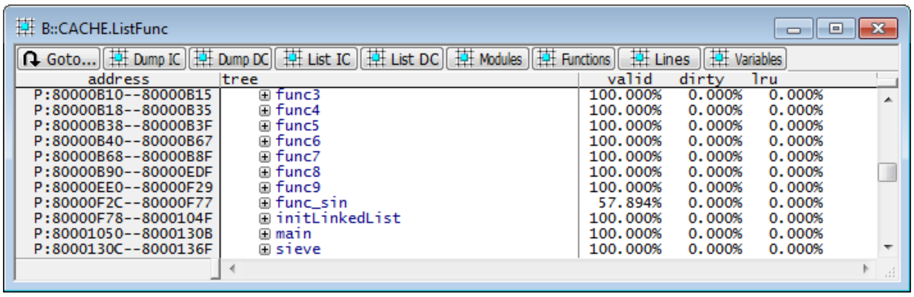
See also

- CACHE
- CACHE.view

CACHE.ListFunc

Format:	**CACHE.ListFunc** *<cache>*
<cache>:	**IC** | **DC** | **L2**

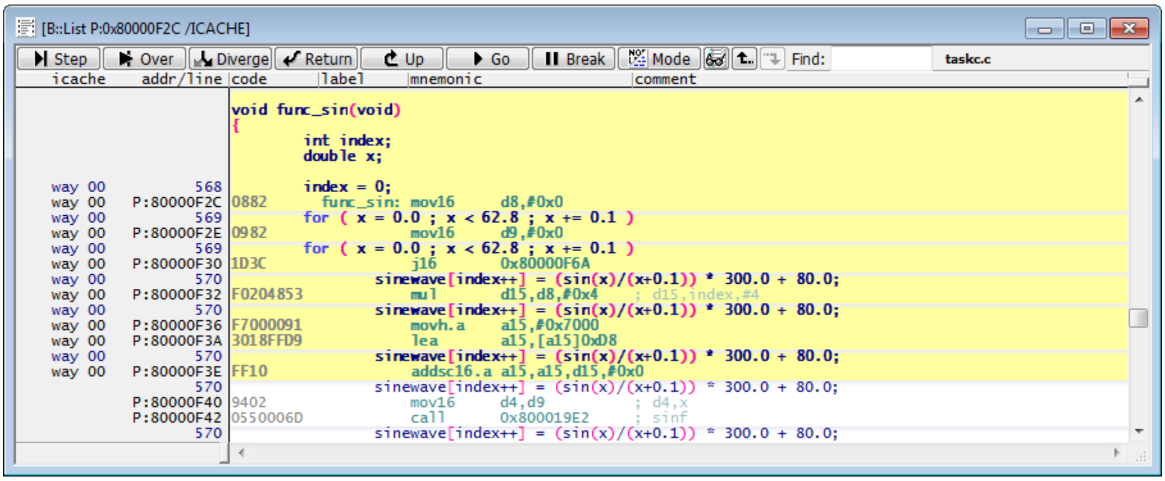
Displays how much of each function is cached.



B::CACHE.ListFunc

```
      address              tree                          valid     dirty     lru
P:80000B10--80000B15       ⊞ func3                       100.000%   0.000%   0.000%
P:80000B18--80000B35       ⊞ func4                       100.000%   0.000%   0.000%
P:80000B38--80000B3F       ⊞ func5                       100.000%   0.000%   0.000%
P:80000B40--80000B67       ⊞ func6                       100.000%   0.000%   0.000%
P:80000B68--80000B8F       ⊞ func7                       100.000%   0.000%   0.000%
P:80000B90--80000EDF       ⊞ func8                       100.000%   0.000%   0.000%
P:80000EE0--80000F29       ⊞ func9                       100.000%   0.000%   0.000%
P:80000F2C--80000F77       ⊞ func_sin                     57.894%   0.000%   0.000%
P:80000F78--8000104F       ⊞ initLinkedList              100.000%   0.000%   0.000%
P:80001050--8000130B       ⊞ main                        100.000%   0.000%   0.000%
P:8000130C--8000136F       ⊞ sieve                       100.000%   0.000%   0.000%
```

Detailed information about a function is displayed by double-clicking the function.



[B::List P:0x80000F2C /ICACHE]

```
  icache     addr/line  code       label      mnemonic                       comment
                        void func_sin(void)
                        {
                                int index;
                                double x;

  way 00           568          index = 0;
  way 00    P:80000F2C  0882       func_sin: mov16      d8,#0x0
  way 00           569          for ( x = 0.0 ; x < 62.8 ; x += 0.1 )
  way 00    P:80000F2E  0982                 mov16      d9,#0x0
  way 00           569          for ( x = 0.0 ; x < 62.8 ; x += 0.1 )
  way 00    P:80000F30  1D3C                 j16        0x80000F6A
  way 00           570                  sinewave[index++] = (sin(x)/(x+0.1)) * 300.0 + 80.0;
  way 00    P:80000F32  F0204853             mul        d15,d8,#0x4     ; d15,index,#4
  way 00           570                  sinewave[index++] = (sin(x)/(x+0.1)) * 300.0 + 80.0;
  way 00    P:80000F36  F7000091             movh.a     a15,#0x7000
  way 00    P:80000F3A  3018FFD9             lea        a15,[a15]0xD8
  way 00           570                  sinewave[index++] = (sin(x)/(x+0.1)) * 300.0 + 80.0;
  way 00    P:80000F3E  FF10                 addsc16.a  a15,a15,d15,#0x0
                   570                  sinewave[index++] = (sin(x)/(x+0.1)) * 300.0 + 80.0;
            P:80000F40  9402                 mov16      d4,d9           ; d4,x
            P:80000F42  0550006D             call       0x800019E2      ; sinf
                   570                  sinewave[index++] = (sin(x)/(x+0.1)) * 300.0 + 80.0;
```

See also

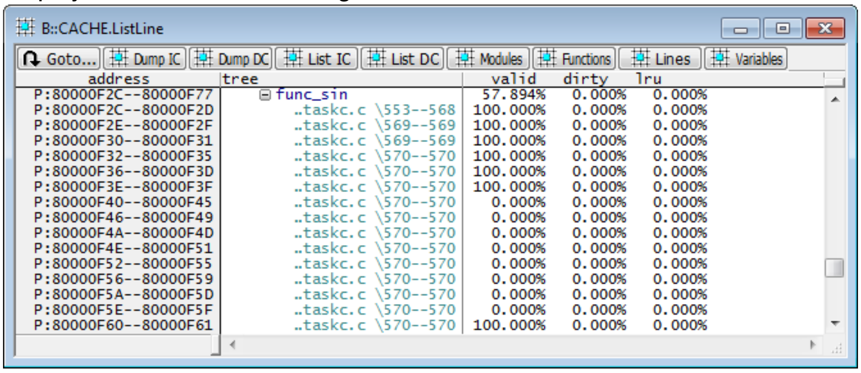
- ■ CACHE
- ■ CACHE.view

CACHE.ListLine

List cached source code lines

Format:	**CACHE.ListLine** *<cache>*
<cache>:	**IC** | **DC** | **L2**

Displays how much of each high-level source code line is cached.



```
       address       tree                           valid    dirty    lru
 P:80000F2C--80000F77      ⊟ func_sin               57.894%   0.000%   0.000%
 P:80000F2C--80000F2D         ..taskc.c \553--568  100.000%   0.000%   0.000%
 P:80000F2E--80000F2F         ..taskc.c \569--569  100.000%   0.000%   0.000%
 P:80000F30--80000F31         ..taskc.c \569--569  100.000%   0.000%   0.000%
 P:80000F32--80000F35         ..taskc.c \570--570  100.000%   0.000%   0.000%
 P:80000F36--80000F3D         ..taskc.c \570--570  100.000%   0.000%   0.000%
 P:80000F3E--80000F3F         ..taskc.c \570--570  100.000%   0.000%   0.000%
 P:80000F40--80000F45         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F46--80000F49         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F4A--80000F4D         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F4E--80000F51         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F52--80000F55         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F56--80000F59         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F5A--80000F5D         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F5E--80000F5F         ..taskc.c \570--570    0.000%   0.000%   0.000%
 P:80000F60--80000F61         ..taskc.c \570--570  100.000%   0.000%   0.000%
```

Detailed information about a line is displayed by double-clicking the line.

See also

- CACHE
- CACHE.view

CACHE.ListModule

List cached modules

Format:	**CACHE.ListModule** *<cache>*
<cache>:	**IC** | **DC** | **L2**

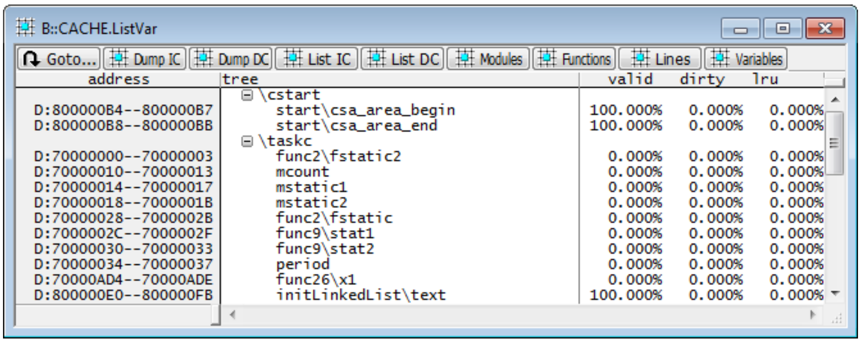
Displays how much of each module is cached.

See also

- CACHE
- CACHE.view

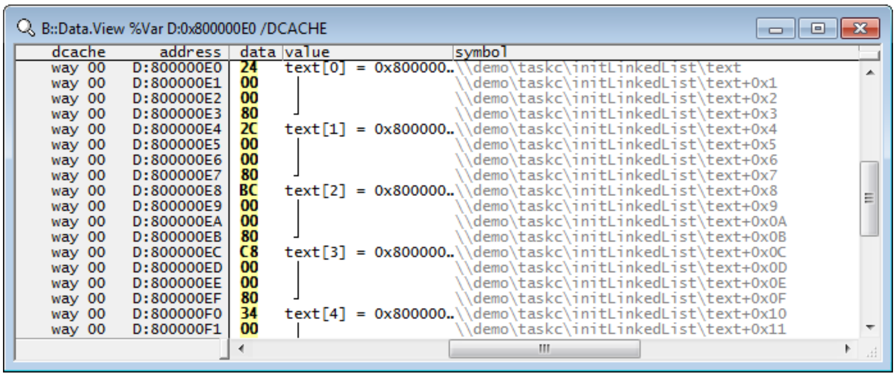
Format:	**CACHE.ListVar** *<cache>* [*<range>* | *<address>*]
<cache>:	**IC** | **DC** | **L2**

Displays all cached variables.



```
address                  tree                               valid    dirty   lru
                         ⊟ \cstart
D:800000B4--800000B7        start\csa_area_begin           100.000%  0.000%  0.000%
D:800000B8--800000BB        start\csa_area_end             100.000%  0.000%  0.000%
                         ⊟ \taskc
D:70000000--70000003        func2\fstatic2                   0.000%  0.000%  0.000%
D:70000010--70000013        mcount                           0.000%  0.000%  0.000%
D:70000014--70000017        mstatic1                         0.000%  0.000%  0.000%
D:70000018--7000001B        mstatic2                         0.000%  0.000%  0.000%
D:70000028--7000002B        func2\fstatic                    0.000%  0.000%  0.000%
D:7000002C--7000002F        func9\stat1                      0.000%  0.000%  0.000%
D:70000030--70000033        func9\stat2                      0.000%  0.000%  0.000%
D:70000034--70000037        period                           0.000%  0.000%  0.000%
D:70000AD4--70000ADE        func26\x1                        0.000%  0.000%  0.000%
D:800000E0--800000FB        initLinkedList\text            100.000%  0.000%  0.000%
```

Detailed information about a variable is displayed by double-clicking the variable.



```
dcache     address    data value                   symbol
way 00   D:800000E0   24   text[0] = 0x800000..\\demo\taskc\initLinkedList\text
way 00   D:800000E1   00                        \\demo\taskc\initLinkedList\text+0x1
way 00   D:800000E2   00                        \\demo\taskc\initLinkedList\text+0x2
way 00   D:800000E3   80                        \\demo\taskc\initLinkedList\text+0x3
way 00   D:800000E4   2C   text[1] = 0x800000..\\demo\taskc\initLinkedList\text+0x4
way 00   D:800000E5   00                        \\demo\taskc\initLinkedList\text+0x5
way 00   D:800000E6   00                        \\demo\taskc\initLinkedList\text+0x6
way 00   D:800000E7   80                        \\demo\taskc\initLinkedList\text+0x7
way 00   D:800000E8   BC   text[2] = 0x800000..\\demo\taskc\initLinkedList\text+0x8
way 00   D:800000E9   00                        \\demo\taskc\initLinkedList\text+0x9
way 00   D:800000EA   00                        \\demo\taskc\initLinkedList\text+0x0A
way 00   D:800000EB   80                        \\demo\taskc\initLinkedList\text+0x0B
way 00   D:800000EC   C8   text[3] = 0x800000..\\demo\taskc\initLinkedList\text+0x0C
way 00   D:800000ED   00                        \\demo\taskc\initLinkedList\text+0x0D
way 00   D:800000EE   00                        \\demo\taskc\initLinkedList\text+0x0E
way 00   D:800000EF   80                        \\demo\taskc\initLinkedList\text+0x0F
way 00   D:800000F0   34   text[4] = 0x800000..\\demo\taskc\initLinkedList\text+0x10
way 00   D:800000F1   00                        \\demo\taskc\initLinkedList\text+0x11
```

See also

■ CACHE ■ CACHE.view

CACHE.LOAD — Load previously stored cache contents

Format: **CACHE.LOAD** [**IC** | **DC** | **L2**] *<file.cd>*

Loads the cache contents previously stored with **CACHE.SAVE**.

This command is not supported for all target processor architectures.

See also

■ CACHE ■ CACHE.view

CACHE.RELOAD — Reload previously loaded cache contents

Format: **CACHE.RELOAD**

Deletes all cache data that TRACE32 already loaded. Cache data that is needed afterwards will be reloaded from the target. For diagnostic purpose only.

This command can be useful when the cache data are loaded during a subsequent operation that needs them, such as when executing **CACHE.List** or **CACHE.GET** command. It means that **Cache.RELOAD** does not trigger any immediate cache read operation but simply marks the data for reloading.

See also

■ CACHE ■ CACHE.GET ■ CACHE.view

CACHE.SAVE — Save cache contents for postprocessing

Format: **CACHE.SAVE** [**IC** | **DC** | **L2**] *<file.cd>*

The cache contents are stored to a selected file. The file can be loaded for post processing with the command **CACHE.LOAD**.

See also

■ CACHE ■ CACHE.view

Format:	**CACHE.SNAPSHOT** *<cache>* [**/***ComPare* [**/***<cmp_opt>*]]
<cache>:	**IC** | **DC** | **L2**
<cmp_opt>:	**AREA <area>** | **VERBOSE** | **RAW**

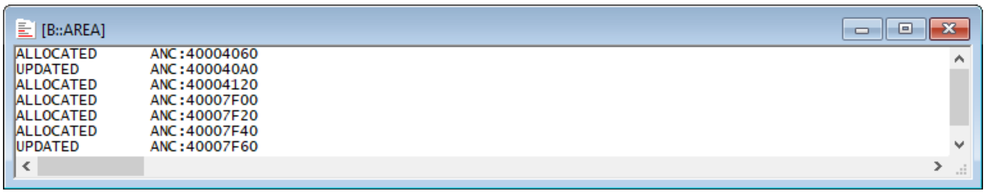
This command helps to investigate how the cache changes, e.g. before and after a function call. If the command is executed without option, it takes a snapshot of the specified cache.

If the command is executed with option /ComPare, it compares the previously taken snapshot to the current cache and prints the differences into the message **AREA**. Destination area and level of detail can be configured using the options outlined below. Without detail option, the output contains event and affected address.

AREA <area>	The message AREA with name <area> will receive the comparison result.
VERBOSE	Additionally print cache set and cache way of the affected cache line.
RAW	Additionally print cache set and way, all status flags, and old and new data stored in the affected cache line.

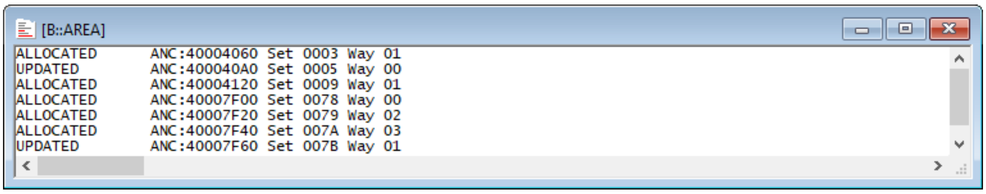
Examples:

```
CACHE.SNAPSHOT /ComPare
```



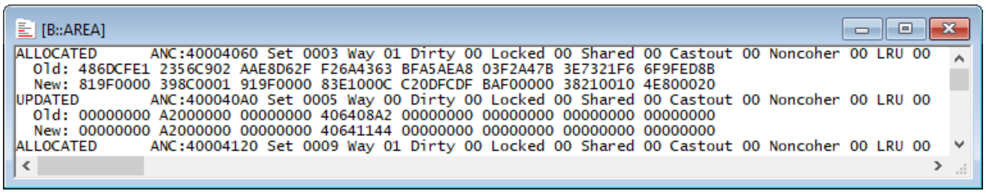
```
ALLOCATED     ANC:40004060
UPDATED       ANC:400040A0
ALLOCATED     ANC:40004120
ALLOCATED     ANC:40007F00
ALLOCATED     ANC:40007F20
ALLOCATED     ANC:40007F40
UPDATED       ANC:40007F60
```

```
CACHE.SNAPSHOT /ComPare /VERBOSE
```



```
ALLOCATED     ANC:40004060 Set 0003 Way 01
UPDATED       ANC:400040A0 Set 0005 Way 00
ALLOCATED     ANC:40004120 Set 0009 Way 01
ALLOCATED     ANC:40007F00 Set 0078 Way 00
ALLOCATED     ANC:40007F20 Set 0079 Way 02
ALLOCATED     ANC:40007F40 Set 007A Way 03
UPDATED       ANC:40007F60 Set 007B Way 01
```

```
CACHE.SNAPSHOT /RAW
```



```
ALLOCATED      ANC:40004060 Set 0003 Way 01 Dirty 00 Locked 00 Shared 00 Castout 00 Noncoher 00 LRU 00
  Old: 486DCFE1 2356C902 AAE8D62F F26A4363 BFA5AEA8 03F2A47B 3E7321F6 6F9FED8B
  New: 819F0000 398C0001 919F0000 83E1000C C20DFCDF BAF00000 38210010 4E800020
UPDATED        ANC:400040A0 Set 0005 Way 00 Dirty 00 Locked 00 Shared 00 Castout 00 Noncoher 00 LRU 00
  Old: 00000000 A2000000 00000000 406408A2 00000000 00000000 00000000 00000000
  New: 00000000 A2000000 00000000 40641144 00000000 00000000 00000000 00000000
ALLOCATED      ANC:40004120 Set 0009 Way 01 Dirty 00 Locked 00 Shared 00 Castout 00 Noncoher 00 LRU 00
```

See also

- CACHE
- CACHE.view

CACHE.UNLOAD — Unload previously loaded cache contents

Format: **CACHE.UNLOAD** [**IC** | **DC** | **L2**]

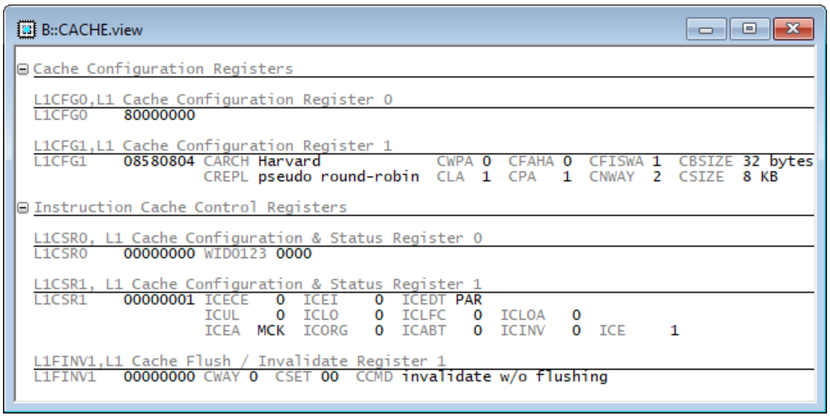
Unloads cache contents previously loaded with the command **CACHE.LOAD**.

See also

- CACHE
- CACHE.view

Format: **CACHE.view**

Displays all cache registers (not available for all processor architectures).


B::CACHE.view

```
⊟ Cache Configuration Registers

  L1CFG0,L1 Cache Configuration Register 0
  L1CFG0     80000000

  L1CFG1,L1 Cache Configuration Register 1
  L1CFG1     08580804 CARCH Harvard              CWPA 0   CFAHA 0   CFISWA 1   CBSIZE 32 bytes
                      CREPL pseudo round-robin   CLA  1   CPA   1   CNWAY  2   CSIZE  8 KB
⊟ Instruction Cache Control Registers

  L1CSR0, L1 Cache Configuration & Status Register 0
  L1CSR0     00000000 WID0123 0000

  L1CSR1, L1 Cache Configuration & Status Register 1
  L1CSR1     00000001 ICECE    0  ICEI     0  ICEDT PAR
                      ICUL     0  ICLO     0  ICLFC    0  ICLOA    0
                      ICEA   MCK  ICORG    0  ICABT    0  ICINV    0  ICE      1

  L1FINV1,L1 Cache Flush / Invalidate Register 1
  L1FINV1    00000000 CWAY 0  CSET 00  CCMD invalidate w/o flushing
```

See also

■ CACHE	■ CACHE.CLEAN	■ CACHE.ComPare	■ CACHE.DUMP
■ CACHE.FLUSH	■ CACHE.GET	■ CACHE.INFO	■ CACHE.INVALIDATE
■ CACHE.List	■ CACHE.ListFunc	■ CACHE.ListLine	■ CACHE.ListModule
■ CACHE.ListVar	■ CACHE.LOAD	■ CACHE.RELOAD	■ CACHE.SAVE
■ CACHE.SNAPSHOT	■ CACHE.UNLOAD		

▲ 'Release Information' in 'Legacy Release History'

CAnalyzer — Trace features of Compact Analyzer

CAnalyzer (Compact Analyzer) is the command group that controls the trace of the following TRACE32 tools:

- **TRACE32 CombiProbe**

  The TRACE32 CombiProbe can be used for the following type of trace information:

  - Any type of trace information generated by a STM or a comparable trace generation unit.
  - All types of trace information generated by the Cortex-M trace infrastructure.
  - MCDS data exported from a TriCore chip via DAP streaming.

  Further information is provided by **“CombiProbe for Cortex-M User’s Guide”** (combiprobe_cortexm.pdf), by **“Intel® x86/x64 Debugger”** (debugger_x86.pdf) or by **“MCDS User’s Guide”** (mcds_user.pdf).

- **µTrace (MicroTrace)**

  The µTrace (MicroTrace) can record all types of trace information generated by the Cortex-M trace infrastructure.

  Further information is provided by **“MicroTrace for Cortex-M User’s Guide”** (microtrace_cortexm.pdf).

- **Serial Wire Viewer (SWV) trace via Debug Cable**

  With some hardware combinations, TRACE32 can record ITM generated trace information exported over the SWO (Serial Wire Output) pin of the debug connector. Trace data is stored in the PowerDebug module without requiring specialized hardware like a CombiProbe or PowerTrace. The PowerDebug Module provides 64 MiB of trace memory.
  To use this feature, one of the following Debug Cables are required:

  - IDC20A DebugCable V5b (formerly ARM DebugCable V5b)
  - AUTO26 Debug Cable V2 (formerly Automotive-Pro Debug Cable)
  - AUTO26 Debug Cable V3

  In addition, the feature is only available with the PowerDebug PRO/E40/X50 modules.

- **TriCore DAP streaming via Debug Cable**

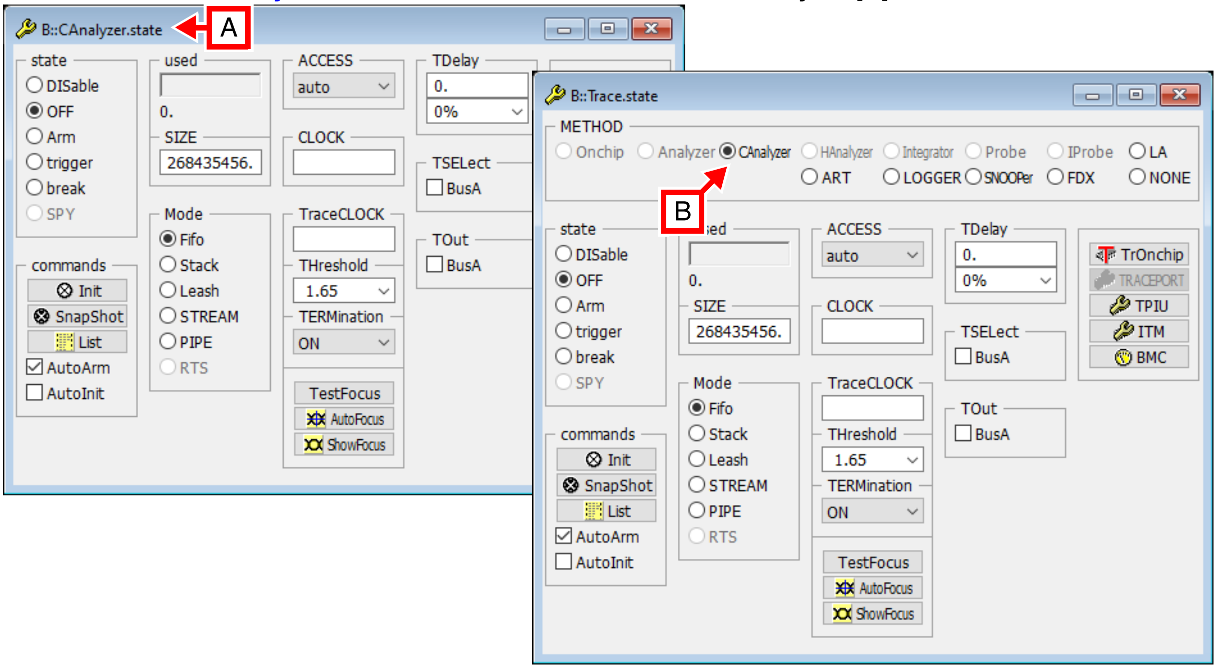
  With the combination of PowerDebug PRO/E40/X50 and the AUTO26 Debug Cable V3, it is possible to stream MCDS data from the target via the normal debug interface in real time. Trace data is stored in the PowerDebug module without requiring specialized hardware like a CombiProbe or PowerTrace. The PowerDebug Module provides 64 MiB of trace memory.
  Further information is provided by **“MCDS User’s Guide”** (mcds_user.pdf).

Trace generation is usually done in real-time. An exception is any trace information generated by code instrumentation.

The amount of trace memory can be extended by using host memory (CAnalyzer STREAM mode, see **CAnalyzer.Mode STREAM**). In many cases, this allows a virtually unlimited amount of trace memory because the average data rate from the TRACE32 hardware to the host exceeds the average trace data rate from the target.

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

Alternatively, use the **Trace.state** window: click the option **CAnalyzer** or execute the command **Trace.METHOD CAnalyzer** in order to select the trace method **CAnalyzer** [**B**].



The chapter **"CAnalyzer - Compact Analyzer specific Trace Commands"**, page 28 describes the CAnalyzer-specific configuration commands. While the chapter **"Generic CAnalyzer Trace Commands"**, page 43 lists the CAnalyzer trace analysis and display commands, which are shared with other TRACE32 trace methods.

See also

■ Trace.METHOD

▲ 'CAnalyzer - Compact Analyzer specific Trace Commands' in 'General Commands Reference Guide C'
▲ 'Generic CAnalyzer Trace Commands' in 'General Commands Reference Guide C'
▲ 'Release Information' in 'Legacy Release History'

CAnalyzer - Compact Analyzer specific Trace Commands

CAnalyzer.<specific_cmds> Overview of CAnalyzer-specific commands

See also

- CAnalyzer.SAMPLE
- CAnalyzer.ShowFocusClockEye
- <trace>.DRAW
- CAnalyzer.PipeWRITE
- CAnalyzer.TOut
- CAnalyzer.WRITE
- CAnalyzer.ShowFocus
- CAnalyzer.ShowFocusEye
- CAnalyzer.DecodeMode
- CAnalyzer.TERMination
- CAnalyzer.TraceCLOCK

▲ 'CAnalyzer' in 'General Commands Reference Guide C'

CAnalyzer.CLOCKDelay Set clock delay

Format:	**CAnalyzer.CLOCKDelay** *<delay>*
<delay>:	**Auto** | **None** | **Small** | **MEDium** | **Large** | **MAXimum**

Default: Auto. Sets the clock delay.

This command exists for setups with the CombiProbe and a whisker other than the MIPI20T-HS whisker. In this case, the command sets the configurable delay between the TRACECLK signal and the registers that sample the trace data, while the data delays cannot be configured.

If available, use **CAnalyzer.SAMPLE** for more precise control of the individual sample points.

CAnalyzer.CLOSE Close named pipes

Format:	**CAnalyzer.CLOSE**

Closes all named pipes defined with **CAnalyzer.PipeWRITE**.

Format:	**CAnalyzer.DecodeMode** *<format>*
<format>:	**AUTO** **SDTI** **STP** **STP64** **STPV2** **STPV2LE** **SWV** **ITM** (deprecated) use SWV instead. **CSITM** **CSETM** **CSSTM**

Default: AUTO.

This command can be used to explicitly define how the recorded trace data should be decoded. In general, the CombiProbe will try to use the correct setting automatically, dependent on the CPU selection and enabled debug features (like ITM for example). Nevertheless, it is possible that you explicitly need to specify the trace decoding in cases where the debugger chooses the wrong defaults; for example if you are debugging an ARM core, which implements an ITM and at the same time an STP module and you now need to specify which of the two outputs you are actually recording.

AUTO	Automatically derive settings. The chosen mode depends on **SYStem.CPU**, the **SYStem.CONFIG** settings and **CAnalyzer.TraceCONNECT**.
SDTI	System Debug Trace Interface (SDTI) by Texas Instruments.
STP	STP protocol (MIPI STPv1, D32 packets).
STP64	STP64 protocol (MIPI STPv1, D64 packets).
STPV2	STPv2 protocol (MIPI STPv2, big endian mode).
STPV2LE	STPv2 protocol (MIPI STPv2, little endian mode).
SWV	ITM data transferred via Serial Wire Output.
CSITM	ITM data transferred via a TPIU continuous mode. The trace ID is taken from the ITM component configuration.

CSETM	ETM + optionally ITM data transferred via TPIU continuous mode. The trace IDs are taken from the ETM and ITM component configuration.
CSSTM	STM data transferred via TPIU continuous mode. The trace ID is taken from the STM component configuration.

See also

- CAnalyzer.<specific_cmds>

CAnalyzer.I2C — I2C control

Format:	**CAnalyzer.I2C.***<sub_cmd>*

Synonym for the **I2C** command group. Only makes sense if your debug hardware supports accessing an I2C bus on your target (e.g. CombiProbe with MIPI60-Cv2).

CAnalyzer.PipeLOAD — Load a previously saved file

Format:	**CAnalyzer.PipeLOAD** *<file>*

Loads a file previously saved with **CAnalyzer.PipeSAVE**. Please note that the decoding will only work if your trace setup matches the setup you used when you did save the data via **CAnalyzer.PipeSAVE** (selected CPU, trace component setup,...).

This command is used in conjunction with **CAnalyzer.Mode PIPE**.

CAnalyzer.PipeRePlay — Replay a previously recorded stream

Format:	**CAnalyzer.PipeRePlay** *<file>*

Replays a previously recorded stream of data, which was stored via **CAnalyzer.PipeSAVE**.

This command is useful if you want to develop a PIPE mode processing DLL.Additionally you might also “replay” artificially produced mock-up data to test your DLL.

This command is used in conjunction with **CAnalyzer.Mode PIPE**.

CAnalyzer.PipeSAVE — Define a file that stores received data

Format:	**CAnalyzer.PipeSAVE** *<file>*

Defines a file into which all received data is stored in an **unprocessed** manner.

This command is used in conjunction with **CAnalyzer.Mode PIPE**. It might be used for developing PIPE mode processing DLLs (see **CAnalyzer.PipeRePlay**).

CAnalyzer.Mode STREAM offers a similar functionality.

CAnalyzer.PipeWRITE — Define a named pipe as trace sink

Format:	**CAnalyzer.PipeWRITE** *<pipe_name>* [**/***<options>*]
<options>:	**ChannelID** *<channel_id>* **MasterID** *<master_id>* **XtiMaster DSP** | **CPU** | **MCU** (XTIv2) **XtiMaster DSP** | **CPU1** | **CPU2** (SDTI) **Payload**

This command is used to define a Windows or Unix named pipe as trace sink. Up to 8 named pipes can be defined as trace sinks simultaneously.

The named pipe has to be created by the receiving application, before you can connect to the named pipe. If the pipe is not already connected to a receiving application, the debugger software will report an error.

If you use this command without specifying a pipe name, all open pipes currently used as trace sinks are closed.

The options are the same as for the **CAnalyzer.WRITE** command.

See also

■ CAnalyzer.<specific_cmds>

CAnalyzer.SAMPLE

Set sample time offset

Format:	**CAnalyzer.SAMPLE** [*<channel>*] *<time>*
<channel>: (parallel)	**D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7**
<channel>: (SWV)	**SWO0** | **SWO1** | **SWO2** | **SWO3** | **SWO4** | **SWO5** | **SWO6** | **SWO7**

Use this command to manually configure the sample times of the trace channels. It is typically used to restore values previously stored using the **Store...** button of the **CAnalyzer.ShowFocus** window or with the **STOre CAnalyzerFocus** command.

The availability of this command depends on the plugged hardware. It is only available in the following scenarios:

- CombiProbe with MIPI20T-HS whisker
- CombiProbe 2 or µTrace (MicroTrace) with MIPI20T-HS or MIPI34 whisker
- CombiProbe 2 with MIPI60 whisker (parallel only)
- PowerDebug PRO/E50/X50 with ARM Debug Cable v5 (SWV only)

<channel>	Trace signal to be configured If the parameter is omitted, all signals are configured with the *<time>* setting.
<time> *(parallel)*	**Parameter Type**: Float. The value is interpreted as time in nanoseconds. Sample time offset to trace clock: • Positive value: Data is sampled after the clock edge. • Negative value: Data is sampled before the clock edge.
<time> *(SWV)*	**Parameter Type**: Float. The value is interpreted as time in nanoseconds. Sample time offset to nominal sample point derived from **CAnalyzer.TraceCLOCK** setting: • Positive value: Data is sampled after nominal sample point. • Negative value: Data is sampled before nominal sample point.

Examples:

```
; Set the delay for all channels to 0
CAnalyzer.SAMPLE , 0.0

; Set the delay for the D0 line to 0.4 ns
CAnalyzer.SAMPLE D0 0.4
```

See also

■ CAnalyzer.<specific_cmds>

▲ 'Release Information' in 'Legacy Release History'

CAnalyzer.ShowFocus — Display data eye

Format:	**CAnalyzer.ShowFocus** [*<channels>* …]
<channels>: (parallel)	**D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** **CLK**
<channels>: (SWV)	**SWO0** | **SWO1** | **SWO2** | **SWO3** | **SWO4** | **SWO5** | **SWO6** | **SWO7** **SWOSTOP**

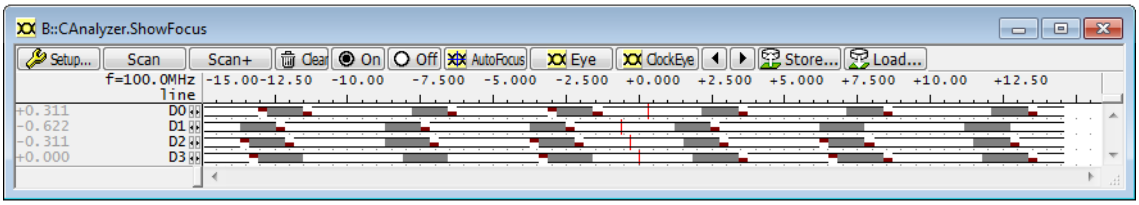
Use this command to get a quick overview of the data eyes for all signals of your trace port.

The availability of this command depends on the plugged hardware. It is only available in the following scenarios:

- CombiProbe with MIPI20T-HS whisker
- CombiProbe 2 or µTrace (MicroTrace) with MIPI20T-HS or MIPI34 whisker
- CombiProbe 2 with MIPI60 whisker (parallel only)
- PowerDebug PRO/E50/X50 with ARM Debug Cable v5 (SWV only)

If used without any arguments, the channels are chosen automatically based on the current **TPIU** settings.

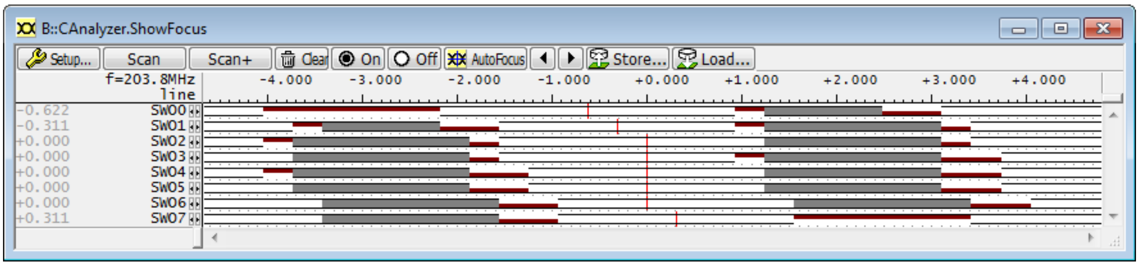
Result for parallel trace:



```
          f=100.0MHz |-15.00-12.50  -10.00   -7.500  -5.000  -2.500  +0.000  +2.500  +5.000  +7.500  +10.00   +12.50
                line
+0.311            D0
-0.622            D1
-0.311            D2
+0.000            D3
```

The horizontal axis is the time difference from the edge of the TRACECLK signal. Each row corresponds to one data channel **D0**, **D1**, etc. The sample point is also displayed numerically at the left of the window (in nanoseconds). Positive values mean that the data line is sampled after the rising clock edge.

Result for SWV trace:



```
          f=203.8MHz      -4.000     -3.000     -2.000     -1.000     +0.000     +1.000     +2.000     +3.000     +4.000
                line
-0.622          SWO0
-0.311          SWO1
+0.000          SWO2
+0.000          SWO3
+0.000          SWO4
+0.000          SWO5
+0.000          SWO6
+0.311          SWO7
```

With SWV trace, there is only a single data line. This line is separated into eight virtual channels, one for each bit of a transmitted byte. For each channel, the delay 0 refers to the “ideal” sample point that is derived from the **CAnalyzer.TraceCLOCK** setting.

Color Legend

- **White** areas represent periods where the corresponding data line was stable.
- **Gray** areas indicate that changes of the data line were detected for both rising and falling clock edges.
- Parallel trace: **Red** areas show that the data line changed only on rising or falling clock edges, not both.
- SWV and parallel trace: **Red** lines indicate the sample points for each data line.

Description of Buttons in the CAnalyzer.ShowFocus Window

The local buttons of the **CAnalyzer.ShowFocus** window have the following functions:

Setup…	Open **CAnalyzer.state** window to configure the trace.
Scan	Perform a **CAnalyzer.TestFocus** scan. This replaces the currently displayed data with a new scan of a test pattern.
Scan+	Perform a **CAnalyzer.TestFocus /Accumulate** scan. This works like **Scan**, but adds to the existing data.
Clear	Clear the currently displayed data.
On	Enable continuous capture. No specific test pattern is generated, but the capture can run in parallel to the recording of normal trace data. The **CAnalyzer.ShowFocus** window updates continuously.
Off	Disable continuous capture.
AutoFocus	Perform a **CAnalyzer.AutoFocus** scan.
Eye	Open a **CAnalyzer.ShowFocusEye** window.
ClockEye	Open a **CAnalyzer.ShowFocusClockEye** window.
Store…	Save the current configuration to a file (**STOre** *<file>* C**AnalyzerFocus**).
Load…	Load a configuration from a file (**DO** *<file>*).
◀	Move all sampling points one step to the left.
▶	Move all sampling points one step to the right.

See also

■ CAnalyzer.<specific_cmds>

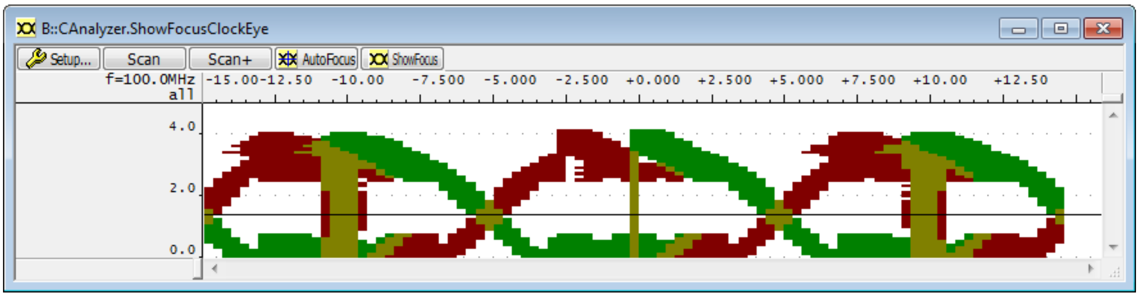
▲ 'Release Information' in 'Legacy Release History'

Format: **CAnalyzer.ShowFocusClockEye**

CAnalyzer.ShowFocusClockEye shows the clock eye. The data is captured by the **CAnalyzer.AutoFocus**, **CAnalyzer.TestFocusClockEye** and **CAnalyzer.TestFocusEye** commands.

The availability of this command depends on the plugged hardware. It is only available in the following scenarios:

- CombiProbe with MIPI20T-HS whisker
- CombiProbe 2 or µTrace (MicroTrace) with MIPI20T-HS, MIPI34 or MIPI60 whisker



The horizontal axis represents time, measured in nanoseconds. The vertical axis represents the voltage. The visible voltage range depends on the hardware capabilities of the whisker.

To generate this view, the clock signal is sampled using the clock signal itself as the trigger. For example, a white area around the coordinate (2.0 V, 7.5 ns) means that there were no recorded clock crossings exactly 7.5 ns apart when using a 2.0 V threshold.

Color Legend

- **White** areas indicate that there were no pairs of clock crossings.
- **Green** indicates that the reference clock crossing at t = 0 was rising.
- **Red** indicates that the reference clock crossing at t = 0 was falling.
- **Olive green** areas indicate that both occurred.

Description of Buttons in the CAnalyzer.ShowFocusClockEye Window

Please see **CAnalyzer.ShowFocusEye**.

See also

■ CAnalyzer.<specific_cmds>

▲ 'Release Information' in 'Legacy Release History'

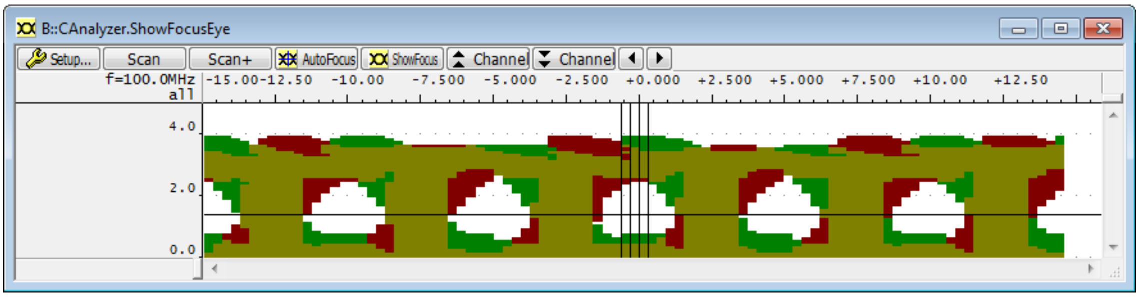
Format:	**CAnalyzer.ShowFocusEye** [*<channels>* …]
<channels>:	**D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7**

CAnalyzer.ShowFocusEye shows the data eyes. The data is captured by the **CAnalyzer.AutoFocus**, **CAnalyzer.TestFocusClockEye** and **CAnalyzer.TestFocusEye** commands.

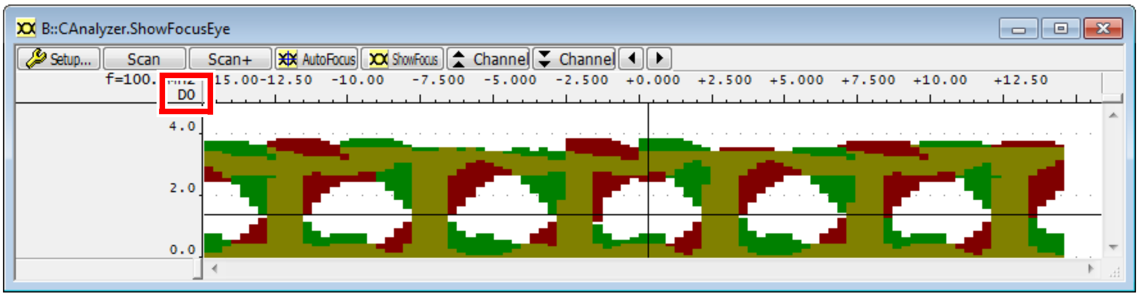
The availability of this command depends on the plugged hardware. It is only available in the following scenarios:

- CombiProbe with MIPI20T-HS whisker
- CombiProbe 2 or µTrace (MicroTrace) with MIPI20T-HS, MIPI34or MIPI60 whisker

This screenshot shows multiple eyes overlaid on each other.



This screenshot shows a single data eye.



Color Legend

- **White** areas indicate that the data was stable (no changes were observed).
- **Green** indicates that the data changed in response to a rising clock edge at t = 0.
- **Red** indicates that the data changed in response to a falling clock edge at t = 0.
- **Olive green** areas indicate that both occurred.

Description of Buttons in the CAnalyzer.ShowFocusEye Window

The toolbar buttons of the **CAnalyzer.ShowFocusEye** window have the following functions:

Setup…	Open **CAnalyzer.state** window to configure the trace.
Scan	Perform a **CAnalyzer.TestFocusEye** scan. This replaces the currently displayed data with a new scan of a test pattern.
Scan+	Perform a **CAnalyzer.TestFocusEye /Accumulate** scan. This works like **Scan**, but adds to the existing data.
AutoFocus	Perform a **CAnalyzer.AutoFocus** scan.
ShowFocus	Open a **CAnalyzer.ShowFocus** window.
Channel up/down	Switch between displayed channels. The default view shows all selected channels overlaid onto each other.
◀	Move the sampling points of all visible channels one step to the left.
▶	Move the sampling points of all visible channels one step to the right.

See also

■ CAnalyzer.<specific_cmds>

CAnalyzer.TERMination — Configure parallel trace termination

Format: **CAnalyzer.TERMination** [**ON** | **OFF** | **ALways**]

Configures the termination of the trace data and clock signals (TRACED0 to TRACED3 and TRACECLK) on the MIPI20T-HS whisker.

This command is only available if a MIPI20T-HS whisker is plugged. This whisker has a switchable 100 Ohm parallel termination to GND. It has no effect in Serial Wire Viewer (SWV) mode.

ON	Termination is enabled while the trace is armed. This is the default and recommended setting. Parallel termination reduces overshoots of the electrical signals.
OFF	Termination is disabled completely. Use this if your target's drivers are too weak to drive against the termination.
ALways	Termination is always enabled.

See also

■ CAnalyzer.<specific_cmds>

▲ 'Release Information' in 'Legacy Release History'

CAnalyzer.TOut — Route trigger to PODBUS (CombiProbe/µTrace)

Format: **CAnalyzer.TOut BusA ON** | **OFF**

When the **BusA** check box is enabled, the CombiProbe/µTrace (MicroTrace) will send out a trigger on the PODBUS, as soon as a trigger event is detected in the trace data.

```
Trace.METHOD.CAnalyzer   ; select the trace method Compact Analyzer
Trace.state              ; open the Trace.state window
Trace.TOut BusA ON       ; enable the BusA check box
```

For information about PODBUS devices, see "**Interaction between independent PODBUS devices**".

See also

■ CAnalyzer.<specific_cmds>

Format:	**CAnalyzer.TraceCLOCK** *<frequency>* **CAnalyzer.ExportClock** *<frequency>* (deprecated)

This command is used to *manually* configure the frequency of the trace port.

The interpretation of this value is different depending on whether a parallel or a SWV trace port is used.

Interpretation when parallel trace is used

With parallel trace, this setting is optional and does not affect the capture of data. However, it is used to interpolate the timestamps in the recorded trace data where multiple logical records share a physical timestamp. Set the value to zero (0.0) to disable timestamp interpolation.

The given frequency must be the bit rate of the trace port. Since all parallel trace ports supported by the CAnalyzer operate in double data rate (DDR) mode, this is twice the frequency of the trace clock pin.

The command **CAnalyzer.AutoFocus** automatically sets this setting.

Interpretation when SWV trace is used

The bit rate of the Serial Wire Output (SWO) signal is used as frequency.

<frequency> (MIPI34 whisker and ARM Debug Cable v5)	Frequency range: • Minimum: 60 kHz • Maximum: 100 MHz
<frequency> (MIPI20T-HS whisker)	Frequency range: • Minimum: 60 kHz • Maximum: 200 MHz

You might need to select an appropriate SWO clock divider to remain in the allowed range. For an example, see **TPIU.SWVPrescaler**.

Examples:

```
CAnalyzer.TraceCLOCK 32MHz
```

To *auto-detect* the bit rate, click the **AutoFocus** button in the **CAnalyzer** window or type at the command line:

```
CAnalyzer.AutoFocus
```

See also

■ CAnalyzer.<specific_cmds>

CAnalyzer.TracePORT   Select which trace port is used

Format:	**CAnalyzer.TracePORT DEFault | TracePortA | TracePortB**

Selects which trace port is used for recording trace data. This command only makes sense if you have **two** whiskers connected to a **CombiProbe**.

DEFault	Use same whisker for tracing as is used for debugging. The debug port can be selected with the command **SYStem.CONFIG DEBUGPORT**. **TracePortA** is selected per default if only one debug port is available.
TracePortA	Select whisker A as trace port.
TracePortB	Select whisker B as trace port.

Format:	**CAnalyzer.WRITE** *<file>* [**/***<options>*]
<options>:	**ChannelID** *<channel_id>* **MasterID** *<master_id>* **XtiMaster DSP** | **CPU** | **MCU** (XTIv2) **XtiMaster DSP** | **CPU1** | **CPU2** (SDTI) **Payload**

This command is used to define a file as trace sink. Up to 8 files can be specified as trace sinks simultaneously.

<file>	If you use this command without specifying a *<file>* name, all open files currently used as trace sinks are closed.
ChannelID **MasterID**	If you record MIPIs STP trace (System Trace Protocol), then the options **/ChannelID** and **/MasterID** are available. You can use this options to only store messages into the file, which match the given ChannelID or MasterID. You can specify a single value, a range of values or a bitmask for the **ChannelID** and **MasterID**. If you record ARMs ITM trace, the **MasterID** option is not available, because ITM does not use master IDs.
Payload	The **/Payload** option specifies, that only the payload of the ITM or STP messages is stored into the file.

See also

- CAnalyzer.<specific_cmds>

Generic CAnalyzer Trace Commands

CAnalyzer.ACCESS — Define access path to program code for trace decoding

See command **<trace>.ACCESS** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 131).

CAnalyzer.Arm — Arm the trace

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

CAnalyzer.AutoArm — Arm automatically

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

CAnalyzer.AutoFocus — Calibrate AUTOFOCUS preprocessor

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

CAnalyzer.AutoInit — Automatic initialization

See command **<trace>.AutoInit** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 139).

CAnalyzer.BookMark — Set a bookmark in trace listing

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

CAnalyzer.BookMarkToggle Toggles a single trace bookmark

See command **<trace>.BookMarkToggle** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 142).

CAnalyzer.Chart Display trace contents graphically

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

CAnalyzer.CLOCK Clock to calculate time out of cycle count information

See command **<trace>.CLOCK** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 190).

CAnalyzer.ComPare Compare trace contents

See command **<trace>.ComPare** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 191).

CAnalyzer.ComPareCODE Compare trace with memory

See command **<trace>.ComPareCODE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 193).

CAnalyzer.CustomTrace Custom trace

See command **<trace>.CustomTrace** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 194).

CAnalyzer.CustomTraceLoad Load a DLL for trace analysis/Unload all DLLs

See command **<trace>.CustomTraceLoad** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 195).

CAnalyzer.DISable — Disable the trace

See command **<trace>.DISable** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 196).

CAnalyzer.DRAW — Plot trace data against time

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

CAnalyzer.EXPORT — Export trace data for processing in other applications

See command **<trace>.EXPORT** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 211).

CAnalyzer.ExtractCODE — Extract code from trace

See command **<trace>.ExtractCODE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 231).

CAnalyzer.FILE — Load a file into the file trace buffer

See command **<trace>.FILE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 232).

CAnalyzer.Find — Find specified entry in trace

See command **<trace>.Find** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 234).

CAnalyzer.FindAll — Find all specified entries in trace

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

CAnalyzer.FindChange — Search for changes in trace flow

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

CAnalyzer.FindProgram — Advanced trace search

See command **<trace>.FindProgram** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 238).

CAnalyzer.FindReProgram — Activate advanced existing trace search program

See command **<trace>.FindReProgram** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 239).

CAnalyzer.FindViewProgram — State of advanced trace search programming

See command **<trace>.FindViewProgram** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 239).

CAnalyzer.FLOWPROCESS — Process flowtrace

See command **<trace>.FLOWPROCESS** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 240).

CAnalyzer.FLOWSTART — Restart flowtrace processing

See command **<trace>.FLOWSTART** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 240).

CAnalyzer.Get — Display input level

See command **<trace>.Get** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241).

CAnalyzer.GOTO — Move cursor to specified trace record

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

CAnalyzer.Init — Initialize trace

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

CAnalyzer.JOINFILE — Concatenate several trace recordings

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

CAnalyzer.List — List trace contents

See command **<trace>.List** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 247).

CAnalyzer.ListNesting — Analyze function nesting

See command **<trace>.ListNesting** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 262).

CAnalyzer.ListVar — List variable recorded to trace

See command **<trace>.ListVar** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 265).

CAnalyzer.LOAD — Load trace file for offline processing

See command **<trace>.LOAD** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 269).

CAnalyzer.MERGEFILE — Combine two trace files into one

See command **<trace>.MERGEFILE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 271).

CAnalyzer.Mode — Set the trace operation mode

See command **<trace>.Mode** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 275).

CAnalyzer.OFF Switch off

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

CAnalyzer.PortFilter Specify utilization of trace memory

See command **<trace>.PortFilter** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 279).

CAnalyzer.PortType Specify trace interface

See command **<trace>.PortType** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 281).

CAnalyzer.PROfileChart Profile charts

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

CAnalyzer.PROfileSTATistic Statistical analysis in a table versus time

See command **<trace>.PROfileSTATistic** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 322).

CAnalyzer.PROTOcol Protocol analysis

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

CAnalyzer.PROTOcol.Chart Graphic display for user-defined protocol

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

CAnalyzer.PROTOcol.Draw Graphic display for user-defined protocol

See command **<trace>.PROTOcol.Draw** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 341).

CAnalyzer.PROTOcol.EXPORT Export trace buffer for user-defined protocol

See command **<trace>.PROTOcol.EXPORT** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 342).

CAnalyzer.PROTOcol.Find Find in trace buffer for user-defined protocol

See command **<trace>.PROTOcol.Find** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 343).

CAnalyzer.PROTOcol.List Display trace buffer for user-defined protocol

See command **<trace>.PROTOcol.List** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 344).

CAnalyzer.PROTOcol.PROfileChart Profile chart for user-defined protocol

See command **<trace>.PROTOcol.PROfileChart** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 347).

CAnalyzer.PROTOcol.PROfileSTATistic Profile chart for user-defined protocol

See command **<trace>.PROTOcol.PROfileSTATistic** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 348).

CAnalyzer.PROTOcol.STATistic Display statistics for user-defined protocol

See command **<trace>.PROTOcol.STATistic** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 350).

CAnalyzer.REF — Set reference point for time measurement

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

CAnalyzer.RESet — Reset command

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

CAnalyzer.SAVE — Save trace for postprocessing in TRACE32

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

CAnalyzer.SelfArm — Automatic restart of trace recording

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

CAnalyzer.SIZE — Define buffer size

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

CAnalyzer.SnapShot — Restart trace capturing once

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

CAnalyzer.SPY — Adaptive stream and analysis

See command **<trace>.SPY** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 373).

CAnalyzer.state — Display trace configuration window

See command **<trace>.state** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 375).

CAnalyzer.STATistic Statistic analysis

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

CAnalyzer.STREAMCompression Select compression mode for streaming

See command **<trace>.STREAMCompression** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 482).

CAnalyzer.STREAMFILE Specify temporary streaming file path

See command **<trace>.STREAMFILE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 483).

CAnalyzer.STREAMFileLimit Set size limit for streaming file

See command **<trace>.STREAMFileLimit** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 484).

CAnalyzer.STREAMLOAD Load streaming file from disk

See command **<trace>.STREAMLOAD** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 485).

CAnalyzer.STREAMSAVE Save streaming file to disk

See command **<trace>.STREAMSAVE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 487).

CAnalyzer.TDelay Trigger delay

See command **<trace>.TDelay** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 488).

CAnalyzer.TestFocus Test trace port recording

See command **<trace>.TestFocus** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 491).

CAnalyzer.TestFocusClockEye Scan clock eye

See command **<trace>.TestFocusClockEye** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 493).

CAnalyzer.TestFocusEye Check signal integrity

See command **<trace>.TestFocusEye** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 494).

CAnalyzer.TestUtilization Tests trace port utilization

See command **<trace>.TestUtilization** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 494).

CAnalyzer.THreshold Optimize threshold for trace lines

See command **<trace>.THreshold** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 495).

CAnalyzer.Timing Waveform of trace buffer

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

CAnalyzer.TraceCONNECT Select on-chip peripheral sink

See command **<trace>.TraceCONNECT** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 498).

CAnalyzer.TRACK Set tracking record

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

CAnalyzer.TSELect Select trigger source

See command **<trace>.TSELect** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 500).

CAnalyzer.View Display single record

See command **<trace>.View** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 501).

CAnalyzer.ZERO Align timestamps of trace and timing analyzers

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

CIProbe Trace with Analog Probe and CombiProbe/µTrace (MicroTrace)

CIProbe is the command group that is used to configure, display, and evaluate signal trace information recorded with one of the following setups:

- **Analog Probe connected to port B of a CombiProbe or µTrace (MicroTrace)**

  Using the converter LA-4508, a PowerIntegrator Analog probe can be used to capture analog trace data, which can be correlated with flow trace, e. g. for Energy Trace Analysis (ETA).

- **Mixed-Signal Probe connected to port B of a CombiProbe 2 or µTrace (MicroTrace)**

  A Mixed-Signal probe can be connected directly to port B of a CombiProbe 2 or µTrace (MicroTrace). Like the analog probe, it can be used for ETA, but it is also possible to capture digital signals for protocol analysis or for measuring interrupt latency for external events.

- **Built-in logic analyzer for debug signals**

  With a PowerDebug PRO/E40/X50 and a regular Debug Cable, it is possible to trace the signals that form the debug port (e. g. JTAG). This can be useful when analyzing problems with the debug connection.
  For the IDC20A and AUTO26 Debug Cables, a script to set up signal names and JTAG protocol analysis can be found at **~~/demo/etc/diagnosis/debug_cable_probe_setup.cmm**.

⚠	The Analog Probe must be connected to the port B of the CombiProbe/µTrace (MicroTrace) using a special adapter (LA-4508). Please do not connect the Analog Probe directly to the CombiProbe/µTrace (MicroTrace).

The **CIProbe** feature set and usage is very similar to the **IProbe**, which refers to the analog or logic analyzer port of a PowerTrace module. Notable differences include:

- The **CIProbe** only supports the TRACE32 Analog Probe and Mixed-Signal Probe.
- The **CIProbe** uses the main trace memory of the CombiProbe/µTrace (MicroTrace). The maximum depth is 16/32/64 million records when used with a CombiProbe/µTrace (MicroTrace)/CombiProbe 2, respectively. The built-in logic analyzer for debug signals can store 16 million records.
- The **CIProbe** supports TRACE32 streaming to the host to provide virtually unlimited recording time, limited only by hard drive or SSD capacity of the host PC. Simultaneous **CIProbe** and **CAnalyzer** streaming is also supported.

Due to the similarities, there is no dedicated **CIProbe** user's guide. For general instructions on how to use the **CIProbe** or to learn about its analog capabilities, please refer to **"IProbe User's Guide"** (iprobe_user.pdf). When commands starting with **IProbe** are mentioned, remember to use their **CIProbe** equivalents instead.

The chapter **"CIProbe-specific Trace Commands"**, page 56 describes the CIProbe-specific configuration commands. While the chapter **"Generic CIProbe Trace Commands"**, page 64 lists the CIProbe trace analysis and display commands, which are shared with other TRACE32 trace methods.

CIProbe-specific Trace Commands

CIProbe.<specific_cmds>   Overview of CIProbe-specific commands

See also

■ CIProbe.ALOWerLIMit	■ CIProbe.ATrigEN	■ CIProbe.ATrigMODE	■ CIProbe.AUPPerLIMit
■ CIProbe.Mode	■ CIProbe.state	■ CIProbe.TDelay	■ CIProbe.TOut
■ CIProbe.TSELect			

CIProbe.ALOWerLIMit   Set lower trigger/filter comparator value

Format:	**CIProbe.ALOWerLIMit** *<channel>* *<value>*
<channel>:	**V0** | **V1** | **V2** | **V3** | **I0** | **I1** | **I2**

Sets the lower limit for the trigger and filter logic of a physical ADC channel. The *<value>* must be given in Volts for voltage channels or Amperes for current channels.

The actual comparison performed depends on the **CIProbe.ATrigMODE** setting.

See also

■ CIProbe.<specific_cmds>

CIProbe.ATrigEN   Enable/disable trigger contribution of a channel

Format:	**CIProbe.ATrigEN** *<channel>* [**ON** | **OFF**]
<channel>:	**V0** | **V1** | **V2** | **V3** | **I0** | **I1** | **I2**

Enables or disables the contribution of a physical channel's comparator logic to the CIProbe trigger. If this setting is enabled for multiple channels, a trigger condition is generated when the trigger condition of any channels is satisfied.

If no [**ON** | **OFF**] argument is given, the current state of the setting is toggled.

NOTE:	Even if this setting is **OFF** for a given channel, the comparator may still be used for filtering. Refer to the **POD.ADC** command for details.

See also

- CIProbe.<specific_cmds>

Format:	**CIProbe.ATrigMODE** *<channel> <mode>*
<channel>:	**V0** | **V1** | **V2** | **V3** | **I0** | **I1** | **I2**
<mode>:	**DISabled** | **GreaterUPPer** | **SmallerUPPer** | **GreaterLOWer** | **SmallerLOWer** | **INBound** | **BEYONDbound**

Sets the condition for a physical channel's comparator logic.

DISabled	No value matches.
GreaterUPPer	Value must be greater than upper limit. See **CIProbe.AUPPerLIMit**.
SmallerUPPer	Value must be less than upper limit. See **CIProbe.AUPPerLIMit**.
GreaterLOWer	Value must be greater than lower limit. See **CIProbe.ALOWerLIMit**.
SmallerLOWer	Value must be less than lower limit. See **CIProbe.ALOWerLIMit**.
INBound	Value must be greater than lower limit and less than upper limit. See **CIProbe.ALOWerLIMit** and **CIProbe.AUPPerLIMit**.
BEYONDBound	Value must be less than lower limit or greater than upper limit. See **CIProbe.ALOWerLIMit** and **CIProbe.AUPPerLIMit**.

See also

- CIProbe.<specific_cmds>

CIProbe.AUPPerLIMit — Set upper trigger/filter comparator value

Format:	**CIProbe.AUPPerLIMit** *<channel> <value>*
<channel>:	**V0** | **V1** | **V2** | **V3** | **I0** | **I1** | **I2**

Sets the upper limit for the trigger and filter logic of a physical ADC channel. The *<value>* is in Volts for voltage channels and Amperes for current channels.

The actual comparison performed depends on the **CIProbe.ATrigMODE** setting.

See also

■ CIProbe.<specific_cmds>

CIProbe.Mode — Set trace operation mode

Format:	**CIProbe.Mode Fifo | Stack**

Fifo	If the trace is full, new records will overwrite older records. The trace records always the last cycles before the break.
Stack	If the trace is full recording will be stopped. The trace always records the first cycles after starting the trace.

See also

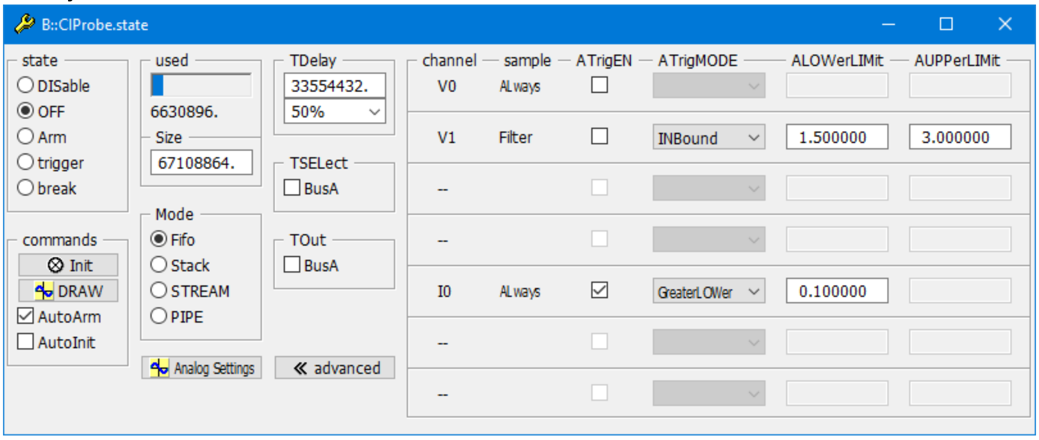
■ CIProbe.<specific_cmds> ■ <trace>.Mode

CIProbe.state

Format: **CIProbe.state**

Displays the main CIProbe configuration window. Use the **advanced** button to get access to analog trigger settings.

Use the **Analog Settings** button or the command **POD.state CIP** to enable and configure channels. Note that by default, all channels are disabled, so no data will be recorded.



B::CIProbe.state

state: ( ) DISable [● OFF] ( ) Arm ( ) trigger ( ) break

used: 6630896.

Size: 67108864.

TDelay: 33554432. / 50%

TSELect: ☐ BusA

TOut: ☐ BusA

commands: Init, DRAW, ☑ AutoArm, ☐ AutoInit

Mode: [● Fifo] ( ) Stack ( ) STREAM ( ) PIPE

Analog Settings | « advanced

channel	sample	ATrigEN	ATrigMODE	ALOWerLIMit	AUPPerLIMit
V0	ALways	☐			
V1	Filter	☐	INBound	1.500000	3.000000
--					
--					
I0	ALways	☑	GreaterLOWer	0.100000	
--					
--					

See also

■ CIProbe.<specific_cmds>

CIProbe.TDelay

Format: **CIProbe.TDelay** *<records>* **|** *<percent>***%**

Selects the delay between the trigger point and the time where the trace stops recording. This delay is always defined as a number of records. For convenience, you can also specify a the delay as a percentage of the current **CIProbe.SIZE** setting.

When the trigger point occurs (either from the trigger comparator or from the **BusA** source), the CIProbe will enter the **trigger** state and keep recording. After the number of records specified in this setting, the CIProbe will then enter the **break** state and no longer record new samples.

Examples:

```
; Stop immediately after the trigger condition. All recorded samples
; will have been sampled before or at the trigger point.
CIProbe.TDelay 0.
```

```
; After the trigger condition occurs, fill the entire trace buffer with
; new samples. All recorded samples will have been sampled after the
; trigger point.
CIProbe.TDelay 100%
```

```
; Stop such that the sample point is exactly in the middle of the
; recorded data.
CIProbe.TDelay 50%
```

See also

■ CIProbe.<specific_cmds>

CIProbe.TOut — Route CIProbe trigger to PODBUS

Format:	**CIProbe.TOut BusA** [**ON** | **OFF**]

When this setting is enabled, the CIProbe will send out a trigger on the PODBUS as soon as a trigger event is detected in the trace data.

Regardless of this setting, a trigger condition will cause the CIProbe to enter the **trigger** and eventually the **break** state.

If no [**ON** | **OFF**] argument is given, the current state of the setting is toggled.

For information about PODBUS devices, see "**Interaction between independent PODBUS devices**".

See also

■ CIProbe.<specific_cmds>

CIProbe.TSELect — Route PODBUS trigger to CIProbe

Format:	**CIProbe.TSELect BusA** [**ON** | **OFF**]

When this setting is enabled, a trigger condition on the PODBUS will trigger the CIProbe, This will cause the CIProbe to enter the **trigger** and eventually the **break** state.

If other trigger conditions are configured with **CIProbe.ATrigEN**, these conditions can independently trigger the CIProbe.

If no [**ON** | **OFF**] argument is given, the current state of the setting is toggled.

For information about PODBUS devices, see "**Interaction between independent PODBUS devices**".

See also

- CIProbe.<specific_cmds>

CIProbe.TSYNC.SELect — Select trigger input pin and edge or state

Format:	**CIProbe.TSYNC.SELect** [*<channel>* *<mode>*]
<mode>:	**Low** | **High** | **Falling** | **Rising** *<value>* *<mask>*

Set the trigger condition for digital trace. Only available for the Mixed-Signal Probe.

The *<channel>* can be one of the digital CIProbe channels (e. g. CIProbe.00 or an alias set with the command **NAME.Set**) or a Word or Group channel (created with **NAME.Word** or **NAME.Group**). The *<mode>* can be either a level (**Low**, **High**), an edge (**Falling**, **Rising**) or a numeric value or mask, which will assign Low, High or don't care to each of the bits in *<channel>*.

It is possible to specify multiple pairs of **[***<channel>* *<mode>***]** with this command. The trigger condition will be the logical AND of the given conditions. If no condition is given at all, no digital trigger will be generated.

While it is possible to specify edge triggers for multiple channels, these edges would have to occur within the same digital sample period. Since the CIProbe's sample clock is asynchronous to the target, this makes it impossible to guarantee that two edges will be sampled at the same time. Therefore, a sensible trigger condition will have at most one edge channel in addition to any number of level channels.

Example:

```
; set up a named word and two named signals
NAME.Word data CIProbe.00 CIProbe.01 CIProbe.02 CIProbe.03
NAME.Set CIProbe.04 clk
NAME.Set CIProbe.05 valid

; set up a trigger condition:
;  - CLK (channel 04) must have a rising edge
;  - VALID (channel 05) must be a logic 1
;  - DATA (channels 00 to 03) must have the value 0x8 or 0xA
CIProbe.TSYNC.SELect CIProbe.clk Rising \
                     CIProbe.valid High \
                     CIProbe.data 0b10x0
```

Generic CIProbe Trace Commands

CIProbe.Arm
Arm the trace

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

CIProbe.AutoArm
Arm automatically

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

CIProbe.AutoInit
Automatic initialization

See command **<trace>.AutoInit** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 139).

CIProbe.BookMark
Set a bookmark in trace listing

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

CIProbe.BookMarkToggle
Toggles a single trace bookmark

See command **<trace>.BookMarkToggle** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 142).

CIProbe.Chart
Display trace contents graphically

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

CIProbe.ComPare Compare trace contents

See command **<trace>.ComPare** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 191).

CIProbe.DISable Disable the trace

See command **<trace>.DISable** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 196).

CIProbe.DisConfig Trace disassembler configuration

See command **<trace>.DisConfig** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 197).

CIProbe.DRAW Plot trace data against time

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

CIProbe.EXPORT Export trace data for processing in other applications

See command **<trace>.EXPORT** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 211).

CIProbe.FILE Load a file into the file trace buffer

See command **<trace>.FILE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 232).

CIProbe.Find Find specified entry in trace

See command **<trace>.Find** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 234).

CIProbe.FindAll Find all specified entries in trace

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

CIProbe.FindChange Search for changes in trace flow

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

CIProbe.Get Display input level

See command **<trace>.Get** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241).

CIProbe.GOTO Move cursor to specified trace record

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

CIProbe.Init Initialize trace

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

CIProbe.List List trace contents

See command **<trace>.List** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 247).

CIProbe.ListNesting Analyze function nesting

See command **<trace>.ListNesting** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 262).

CIProbe.ListVar List variable recorded to trace

See command **<trace>.ListVar** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 265).

CIProbe.LOAD Load trace file for offline processing

See command **<trace>.LOAD** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 269).

CIProbe.OFF Switch off

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

CIProbe.PROfileChart Profile charts

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

CIProbe.REF Set reference point for time measurement

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

CIProbe.RESet Reset command

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

CIProbe.SAVE Save trace for postprocessing in TRACE32

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

CIProbe.SIZE Define buffer size

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

CIProbe.STREAMCompression Select compression mode for streaming

See command **<trace>.STREAMCompression** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 482).

CIProbe.STREAMFILE Specify temporary streaming file path

See command **<trace>.STREAMFILE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 483).

CIProbe.STREAMFileLimit — Set size limit for streaming file

See command **<trace>.STREAMFileLimit** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 484).

CIProbe.TRACK — Set tracking record

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

CIProbe.View — Display single record

See command **<trace>.View** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 501).

CIProbe.ZERO — Align timestamps of trace and timing analyzers

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

ClipSTOre — Store settings to clipboard

Format:	**ClipSTOre** [**%**<*format*>] [<*item*> …]
<*format*>:	**sYmbol** | **NosYmbol**
<*item*>:	**default** | **ALL** | **Win** | **WinPAGE** | **Symbolic** | **HEX** | **SYStem** …

Stores settings in the format of PRACTICE commands to the clipboard.

<*item*>, <*format*>	For a detailed descriptions, refer to the **STOre** command.

Example:

```
ClipSTOre SYStem        ; store the settings of the SYStem.state window
                        ; to the clipboard
```

Result (example):

```
B::

SYStem.RESet
SYStem.CPU CortexA9
SYStem.CONFIG CORE 1.
SYStem.MemAccess Enable
SYStem.CpuBreak Enable
SYStem.CpuSpot Enable
SYStem.Option.IMASKASM  ON
SYStem.Mode Up

ENDDO
```

See also

■ AutoSTOre ■ STOre ■ SETUP.STOre

CLOCK Display date and time

The command group **CLOCK** is used to display and calculate the system clock configuration. The results are also used to decode the on-chip trace timestamp information in complex scenarios.

Currently this feature is only implemented for TriCore, PCP, and GTM.

For architectures that do not have the **CLOCK** command group, **CLOCK** is an alias for **DATE**.

See also

■ CLOCK.BACKUP	■ CLOCK.DATE	■ CLOCK.OFF	■ CLOCK.ON
■ CLOCK.OSCillator	■ CLOCK.Register	■ CLOCK.RESet	■ CLOCK.state
■ CLOCK.SYSCLocK	■ CLOCK.VCOBase	■ CLOCK.VCOBaseERAY	■ DATE

▲ 'Release Information' in 'Legacy Release History'

CLOCK.BACKUP Set backup clock frequency

TriCore only, device dependent

Format:	**CLOCK.BACKUP** *<frequency>*

Default: 100.0MHz (TriCore, device dependent)

Configure the backup clock frequency. Required to compute the clock frequencies when TriCore switches to the backup clock. Check CPU data sheet for details.

See also

■ CLOCK	■ CLOCK.state

CLOCK.DATE — Alias for DATE command

Format:	**CLOCK.DATE**

Alias for the **DATE** command.

See also

■ CLOCK ■ CLOCK.state

CLOCK.OFF — Disable clock frequency computation

Format:	**CLOCK.OFF**

Default: OFF

Disables the computation of clock frequencies.

See also

■ CLOCK ■ CLOCK.state

CLOCK.ON — Enable clock frequency computation

Format:	**CLOCK.ON**

Enables the computation of clock frequencies.

Prior to enabling the computation of clock frequencies, it is recommended to configure the clock sources (oscillator, backup, VCOBase). The resulting clock frequencies are also used for decoding on-chip trace timestamps, if supported by device and TRACE32.

See also

■ CLOCK ■ CLOCK.state

CLOCK.OSCillator — Set board oscillator frequency

Format:	**CLOCK.OSCillator** *<frequency>*

Default: 20.0MHz (TriCore)

Configures the board oscillator clock frequency. Check board oscillator and/or schematics.

See also

■ CLOCK ■ CLOCK.state

CLOCK.Register — Display PLL related registers

Format:	**CLOCK.Register**

Opens the PLL or system clock register section within the device's peripheral file.

See also

■ CLOCK ■ CLOCK.state

CLOCK.RESet — Reset CLOCK command group settings

Format:	**CLOCK.RESet**

Resets all CLOCK command group related settings to defaults.

See also

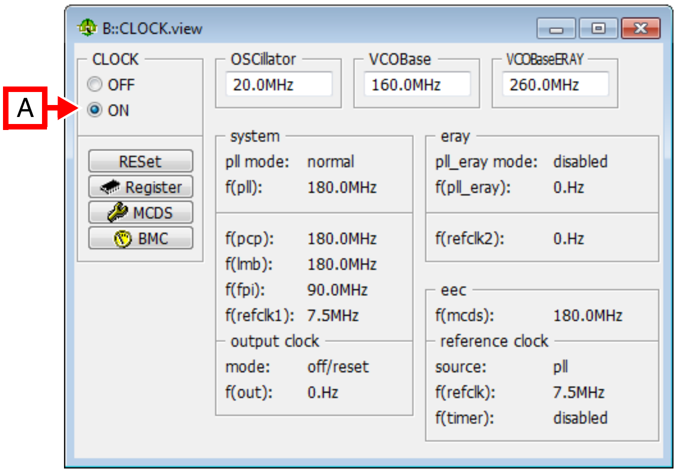
■ CLOCK ■ CLOCK.state

CLOCK.state

Display clock frequencies

Format:	**CLOCK.state**

Opens a dialog with all computed clock frequencies and related settings.



```
CLOCK          OSCillator   VCOBase    VCOBaseERAY
○ OFF          20.0MHz      160.0MHz   260.0MHz
● ON

system                      eray
pll mode:     normal        pll_eray mode:  disabled
f(pll):       180.0MHz      f(pll_eray):    0.Hz

f(pcp):       180.0MHz      f(refclk2):     0.Hz
f(lmb):       180.0MHz
f(fpi):       90.0MHz       eec
f(refclk1):   7.5MHz        f(mcds):        180.0MHz
output clock                reference clock
mode:         off/reset     source:         pll
f(out):       0.Hz          f(refclk):      7.5MHz
                            f(timer):       disabled
```

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

See also

- CLOCK
- CLOCK.BACKUP
- CLOCK.DATE
- CLOCK.OFF
- CLOCK.ON
- CLOCK.OSCillator
- CLOCK.Register
- CLOCK.RESet
- CLOCK.SYSCLocK
- CLOCK.VCOBase
- CLOCK.VCOBaseERAY

CLOCK.SYSCLocK

Set external clock frequency

TriCore only, device dependent

Format:	**CLOCK.SYSCLocK** *<frequency>*

Configure the external clock frequency when the SYSCLOCK pin is used as clock source. Check CPU data sheet for details.

See also

- CLOCK
- CLOCK.state

CLOCK.VCOBase Set "VCOBase" clock frequency

TriCore only, device dependent

Format:	**CLOCK.VCOBase** *<frequency>*

Default: device dependent

Configures the VCO base clock frequency. Required when TriCore PLL operates in free-running mode. Check CPU data sheet for details.

See also

■ CLOCK ■ CLOCK.state

CLOCK.VCOBaseERAY Set "FlexRay VCOBase" clock frequency

TriCore only, device dependent

Format:	**CLOCK.VCOBaseERAY** *<frequency>*

Default: device dependent

Configures the FlexRay VCO base clock frequency. Required when TriCore FlexRay PLL operates in free-running mode. Check CPU data sheet for details.

See also

■ CLOCK ■ CLOCK.state

CMI Clock management interface

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

CMN — Coherent mesh network

The Coherent Mesh Network (CMN) is a scalable and configurable coherent interconnect which enables the developer to output the messages of the coherence protocol without affecting the run-time behavior of the system.

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

CMN<trace> - Trace Data Analysis

CMN<trace>

Command groups for CMN<trace>

Overview CMN<trace>

Using the **CMN<trace>** command group, you can configure the trace recording as well as analyze and display the recorded CMN trace data. The command groups consist of the name of the trace source, here **CMN**, plus the TRACE32 trace method you have chosen for recording the CMN trace data.

For more information about the TRACE32 convention of combining *<trace_source>* and *<trace_method>* to a *<trace>* command group that is aimed at a specific trace source, see **"Replacing <trace> with Trace Source and Trace Method - Examples"** (general_ref_t.pdf).

Not any arbitrary combination of *<trace_source>* and *<trace_method>* is possible. For an overview of the available command groups **"Related Trace Command Groups"** (general_ref_t.pdf).

Example:

```
CMNTrace.state              ;optional step: open the window in which the
                            ;trace recording is configured.
CMNTrace.METHOD Analyzer    ;select the trace method Analyzer for
;<configuration>            ;recording trace data.

CMN.state                   ;optional step: open the window in which
                            ;the trace source CMN is configured.
CMN.ON                      ;switch the trace source CMN on.
;<configuration>

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

CMNAnalyzer.List            ;display the CMN trace data recorded with the
                            ;trace method Analyzer as a trace listing.

CMNTrace.List               ;this is the generic replacement for the above
                            ;CMNAnalyzer.List command.
```

CMNAnalyzer Analyze CMN information recorded by TRACE32 PowerTrace

[Example]

Format:	**CMNAnalyzer.***<sub_cmd>*

The **CMNAnalyzer** command group allows to display and analyze the information emitted by the Coherent Mesh Network (CMN) component.

The CMN information emitted off-chip via the Trace Port Interface Unit (**TPIU**) is recorded by the TRACE32 PowerTrace.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **CMNAnalyzer.List** refer to **<trace>.List**

CMNCAnalyzer Analyze CMN information recorded by CombiProbe

Format:	**CMNCAnalyzer.***<sub_cmd>*

The **CMNCAnalyzer** command group allows to display and analyze the information emitted by the Coherent Mesh Network (CMN) component.

The CMN information emitted off-chip via the Trace Port Interface Unit (**TPIU**) is recorded by the TRACE32 CombiProbe.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **CMNCAnalyzer.List** refer to **<trace>.List**

CMNHAnalyzer — Analyze CMN information captured by the host analyzer

Format:	**CMNHAnalyzer.***<sub_cmd>*

The **CMNHAnalyzer** command group allows to display and analyze the information emitted by the Coherent Mesh Network (CMN) component. Trace data is transferred off-chip via the USB port and is recorded in the trace memory of the TRACE32 host analyzer.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **CMNHAnalyzer.List** refer to **<trace>.List**

CMNLA — Analyze CMN information from binary source

Format:	**CMNLA.***<sub_cmd>*

The **CMNLAnalyzer** command group allows to display and analyze the information emitted by the Coherent Mesh Network (CMN) component. Trace data is collected form Lauterbach's Logic Analyzer or from a binary file.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **CMNLAnalyzer.List** refer to **<trace>.List**

CMNOnchip — Analyze CMN information captured in target onchip memory

Format:	**CMNOnchip.***<sub_cmd>*

The **CMNOnchip** command group allows to display and analyze the information emitted by the Coherent Mesh Network (CMN) component.

The CMN trace is sent to the device-specific onchip trace memory and is read by TRACE32 via debug cable (JTAG).

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **CMNOnchip.List** refer to **<trace>.List**

CORE

Cores in an SMP system

See also

■ CORE.ADD	■ CORE.ASSIGN	■ CORE.List	■ CORE.NUMber
■ CORE.ReMove	■ CORE.select	■ CORE.SHOWACTIVE	■ CORE.SINGLE

▲ ’CORE Functions’ in ’General Function Reference’

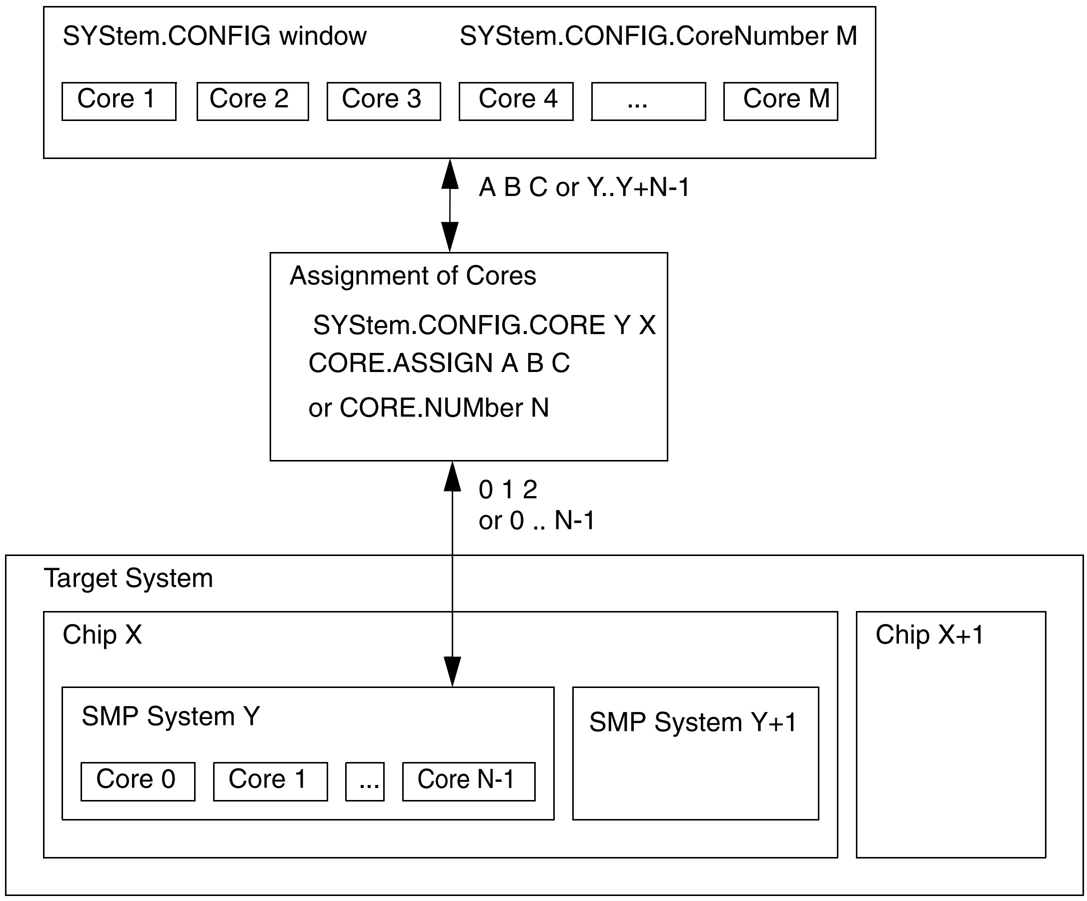
Overview CORE

With the **CORE** command group, TRACE32 supports debugging of SMP systems (symmetric multiprocessing).

For various architectures like ARM, MIPS, PowerPC, and SH4 there are chips containing two or more identical cores.

When debugging SMP systems with TRACE32, the context (**Register** window, **Lis**t window, etc.) of a **single core** is displayed at a time, but it is possible to switch to another core within the same TRACE32 instance. In contrast to this, all debug actions as **Go** or **Break** are effected on **all cores** to maintain synchronicity between the cores.

To set up an SMP System the commands **SYStem.CONFIG.CoreNumber** and **CORE.ASSIGN** or **CORE.NUMber** are necessary. The **SYStem.CONFIG** window and commands define how the access to a certain hardware thread can be achieved and how many hardware threads are available. The **CORE** commands assign the hardware threads to the SMP system that is handled by this TRACE32 instance. In case there are multiple SMP systems configured on the chip, the command **SYStem.CONFIG.CORE** is necessary to define different SMP System indices (Y) that are used as start value for the command **CORE.NUMber** and the information whether the SMP System is located at a different or the same chip by the chip index (X).



Setup of SMP Systems

CORE.ADD   Add core/thread to the SMP system

Format: **CORE.ADD** *<core>* | *<thread>*
THREAD.ADD (deprecated)

Adds a physical core/thread to the SMP System. This synchronizes it with other cores/threads when debug features are applied to the SMP System.

See also

- CORE
- CORE.select

CORE.ASSIGN Assign a set of physical cores/threads to the SMP system

[Examples]

Format 1:	**CORE.ASSIGN** *<core1>* [*<core2>* …]
Format 2:	**CORE.ASSIGN** *<thread1>* [*<thread2>* …] MIPS64, XLR, XLS, XLP, QorIQ64 only

The command configures an instance of the TRACE32 PowerView GUI so that this particular instance knows for which physical cores or physical threads of the target system it is “responsible”. Typically this configuration is required in multicore systems:

- In AMP (asynchronous multiprocessing) systems, each TRACE32 PowerView instance is responsible for a single physical core/thread.
- In SMP (symmetric multiprocessing) systems, an instance of TRACE32 PowerView may be responsible for multiple physical cores/threads.
- Mixed AMP SMP systems may have several TRACE32 PowerView instances, where one or more TRACE32 PowerView instances are responsible for more than one physical core/thread.

<core>	The physical *<core>* number refers to the respective physical core in the chip. This applies to CPUs that have only physical cores (i.e. no physical threads at all, or just one thread).
<thread>	The physical *<thread>* number refers to the respective physical thread in the chip. This applies to CPUs with physical cores that have more than one thread per core. The physical threads are numbered sequentially throughout all cores. Thus, the cores themselves can be ignored in the multicore setup of TRACE32.

Each core/thread assignment is also referred to as TRACE32 configuration. A TRACE32 configuration contains information about how to access a specific *physical core/thread* in a multicore chip, e.g.:

- TAP coordinates (IRPRE, IRPOST, DRPRE, DRPOST)
- CoreSight addresses for ARM chips
- Other physical access parameters for the core/thread

The setup of the individual cores/threads is done in the **SYStem.CONFIG** window.

NOTE:	For each assigned physical core/thread, TRACE32 uses a logical core number, which serves as an alias for the physical core/thread.

Examples

To illustrate the **CORE.ASSIGN** command, the following examples are provided:

- Example 1 - Assignment of Physical Cores
- Example 2 - Assignment of Physical Threads (MIPS specific)
- Example 3 - Core Assignment for an SMP-4 / AMP-3 Setup (MIPS specific)
- Example 4 - Core Assignment for an AMP-2 Setup (MIPS specific)

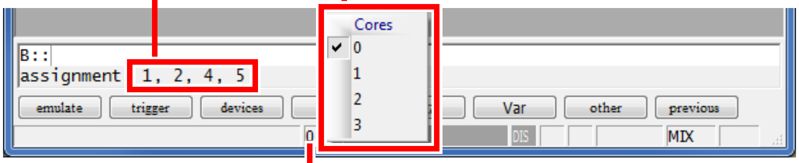
Example 1 - Assignment of Physical Cores

In this example, the *physical* cores 1, 2, 4, and 5 of a multicore chip are assigned to TRACE32; core 3 is not used in this example setup. The resulting *logical* cores can be seen from the **Cores** pull-down list in TRACE32.

```
CORE.ASSIGN 1. 2. 4. 5. ;assign the physical cores 1, 2, 4, and 5
```

Assigned *physical* cores

Resulting *logical* cores

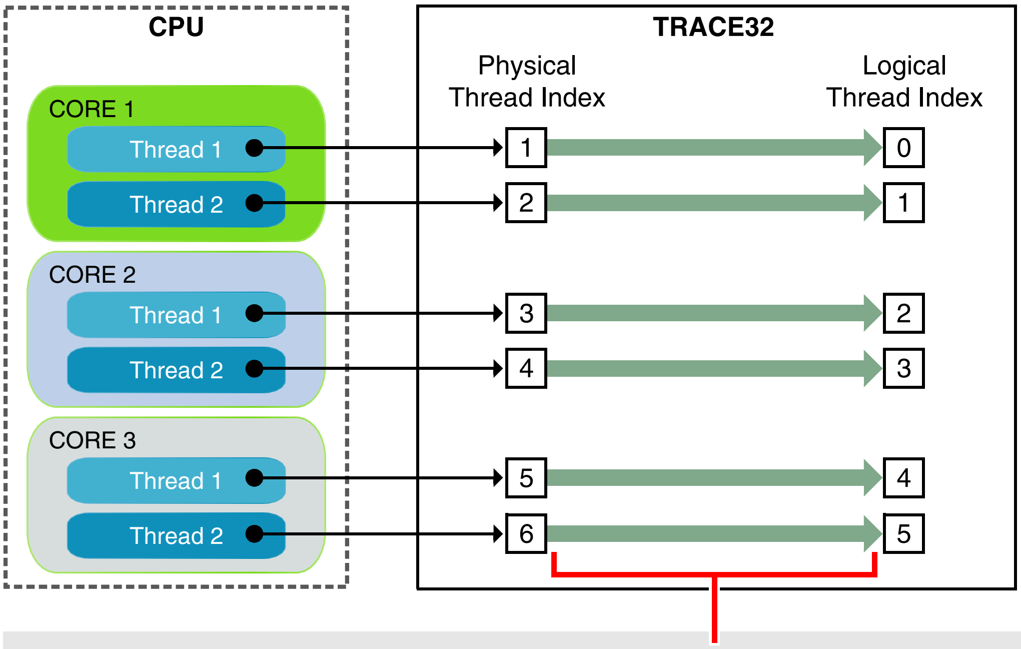


```
B::
assignment  1, 2, 4, 5
```

Right-click to open the **Cores** pull-down list.
In the status line, this box shows the currently selected core, here core 0.

Example 2 - Assignment of Physical Threads

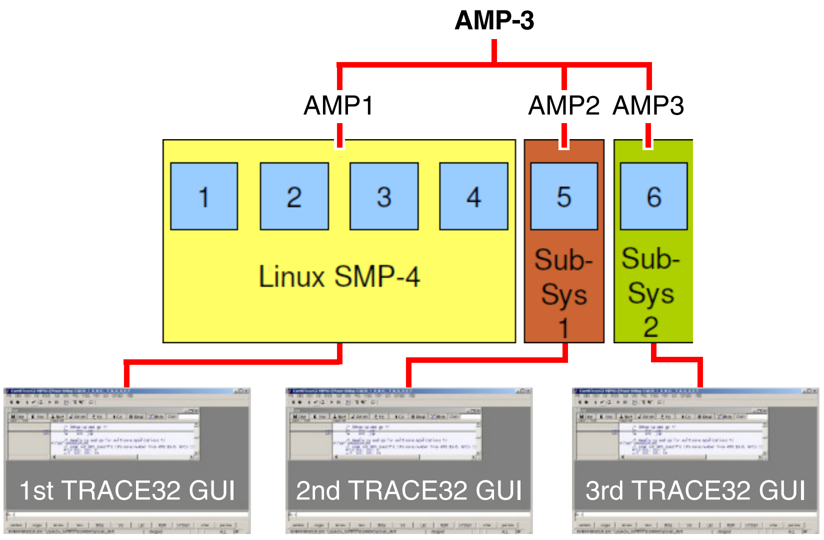
In this example, a CPU has 3 physical cores, each core has 2 physical threads. That means for TRACE32, this CPU has 6 physical threads in total. Use **CORE.ASSIGN** as shown below to assign the 6 *physical* threads. The resulting *logical* threads can be seen from the **Cores** pull-down list in TRACE32.



```
CORE.ASSIGN 1. 2. 3. 4. 5. 6.   ;assign the physical threads 1 to 6
```

Example 3: Core Assignment for an SMP-4 / AMP-3 Setup (MIPS specific)

The figure shows an SMP-4 / AMP-3 setup. For this kind of setup, the six cores need to be assigned to three TRACE32 PowerView GUIs. The target is a MIPS64 with six cores (CPU CN6335).



Code required for assigning the cores 1 to 4 to the first TRACE32 PowerView GUI:

```
SYStem.CPU CN63XX ; Select the target CPU (MIPS CN6335).

; Inform TRACE32 about the total number of cores of this multicore chip.
SYStem.CONFIG.CoreNumber 6.

; Start core assignment at this <core> of this <chip>.
SYStem.CONFIG.CORE                1.             1.

; Assign the cores 1 to 4 to the first TRACE32 PowerView GUI.
CORE.ASSIGN 1. 2. 3. 4.
```

Code required for assigning core 5 to the second TRACE32 PowerView GUI:

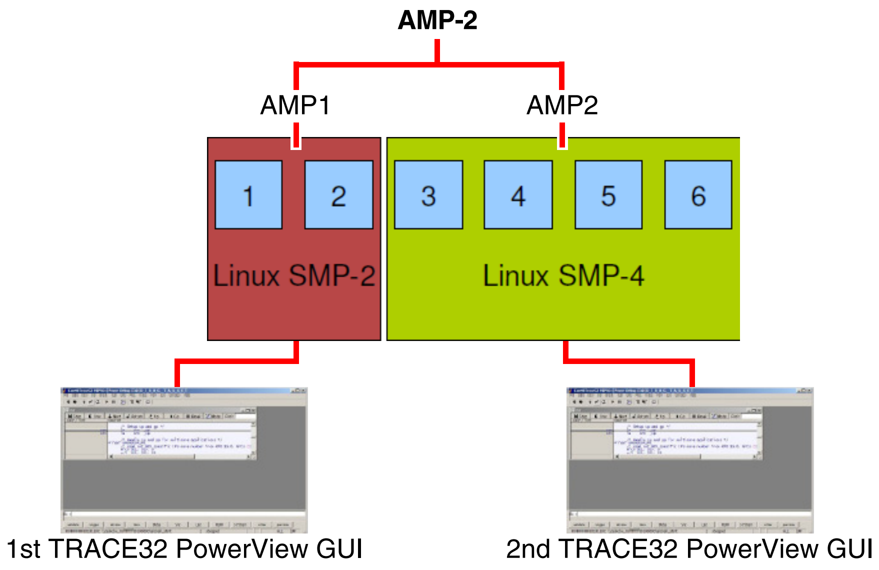
```
; This step needs to be repeated for the second TRACE32 PowerView GUI:
SYStem.CPU CN63XX ;Select the target CPU (MIPS CN6335).

; This step needs to be repeated for the second TRACE32 PowerView GUI:
; Inform TRACE32 about the total number of cores of this multicore chip.
SYStem.CONFIG.CoreNumber 6.

; Start core assignment at this <core> of this <chip>.
SYStem.CONFIG.CORE                5.             1.

; Assign the core 5 to the second TRACE32 PowerView GUI.
CORE.ASSIGN 5.
```

Code required for assigning core 6 to the third TRACE32 PowerView GUI:

```
; This step needs to be repeated for the third TRACE32 PowerView GUI:
SYStem.CPU CN63XX ;Select the target CPU (MIPS CN6335).

; This step needs to be repeated for the third TRACE32 PowerView GUI:
; Inform TRACE32 about the total number of cores of this multicore chip.
SYStem.CONFIG.CoreNumber 6.

; Start core assignment at this <core> of this <chip>.
SYStem.CONFIG.CORE                6.              1.

; Assign the core 6 to the third TRACE32 PowerView GUI.
CORE.ASSIGN 6.
```

Example 4: AMP-2 Setup (MIPS specific)

The figure shows an AMP-2 setup, which in turn consists of an SMP-2 and SMP-4 setup. For this kind of setup, the six cores need to be assigned to two TRACE32 PowerView GUIs. The target is a MIPS64 with six cores (CPU CN6335).



Code required for assigning the cores 1 and 2 to the first TRACE32 PowerView GUI:

```
SYStem.CPU CN63XX ; Select the target CPU (MIPS CN6335).

; Inform TRACE32 about the total number of cores of this multicore chip.
SYStem.CONFIG.CoreNumber 6.

; Start core assignment at this <core> of this <chip>.
SYStem.CONFIG.CORE                1.              1.

; Assign the cores 1 and 2 to the first TRACE32 PowerView GUI.
CORE.ASSIGN 1. 2.
```

Code required for assigning the cores 3 to 6 to the second TRACE32 PowerView GUI:

```
; This step needs to be repeated for the second TRACE32 PowerView GUI:
SYStem.CPU CN63XX ; Select the target CPU (MIPS CN6335).

; This step needs to be repeated for the second TRACE32 PowerView GUI:
; Inform TRACE32 about the total number of cores of this multicore chip.
SYStem.CONFIG.CoreNumber 6.

; Start core assignment at this <core> of this <chip>.
SYStem.CONFIG.CORE                3.              1.
; Assign the cores 3 to 6 to the first TRACE32 PowerView GUI.
CORE.ASSIGN 3. 4. 5. 6.
```

NOTE: The numbering of physical and logical cores is as follows:

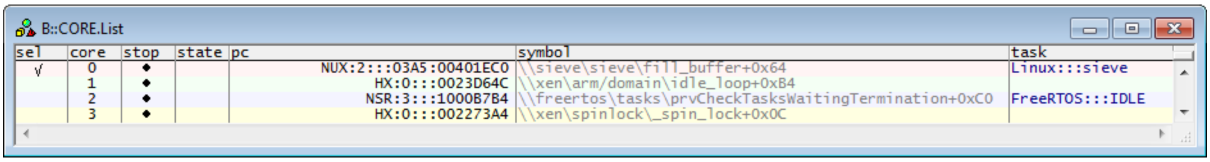
- "Physical cores" may have numbers starting with 1.
- "Logical cores" have numbers starting with 0.

See also

■ CORE ■ CORE.select ■ SYStem.CONFIG.CORE ❏ CORE.ISASSIGNED()

Format:	**CORE.List**

Lists for each core the location of the PC (program counter) and the current task. The list is empty while the cores are running and updated as soon as the program execution is stopped.


B::CORE.List

```
sel   core  stop  state pc                          symbol                                               task
 √     0     ◆          NUX:2:::03A5:00401EC0 \\sieve\sieve\fill_buffer+0x64                          Linux:::sieve
       1     ◆                HX:0:::0023D64C \\xen\arm/domain\idle_loop+0xB4
       2     ◆              NSR:3:::1000B7B4 \\freertos\tasks\prvCheckTasksWaitingTermination+0xC0   FreeRTOS:::IDLE
       3     ◆                HX:0:::002273A4 \\xen\spinlock\_spin_lock+0x0C
```

Description of Columns in the CORE.List Window

sel	Currently selected core.
core	Logical core number.
stop	Stopped cores.
state	Architecture-specific states, e.g. power down.
pc	Location of the PC.
symbol	Symbol information about the PC
task	Active task on core.

See also

■ CORE ■ CORE.select ■ TASK.List.tasks ❏ CORE()

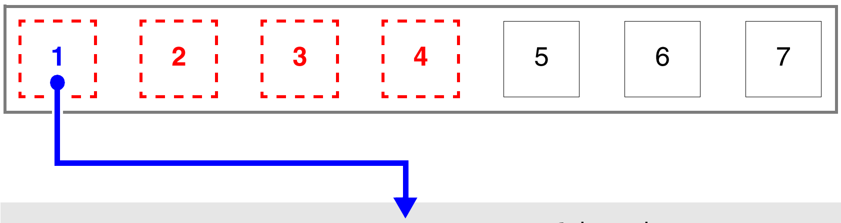
▲ 'PowerView - Screen Display' in 'PowerView User's Guide'

Format: **CORE.NUMber** *<number_of_cores>* | *<number_of_threads>*

Assigns multiple physical cores/threads to the SMP system. The cores/threads are assigned in a linear sequence and without gaps.

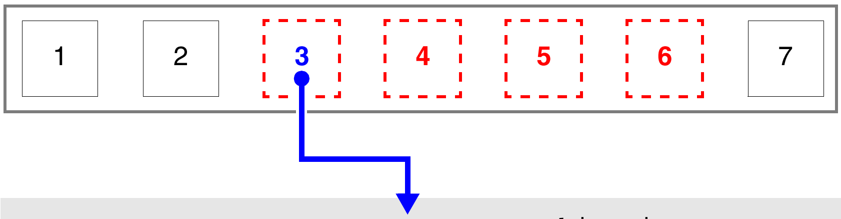
The setup of the cores/threads is done in the **SYStem.CONFIG** window. The assignment starts with the *<core>* parameter of the **SYStem.CONFIG.CORE** command and iterates through the number of cores/threads passed to the **CORE.NUMber** command.

Example 1 shows how to assign the first 4 cores of a chip. In our example, chip 1 has 7 cores.



```
;                      <core>    <chip> i.e. start at core 1 of chip 1
SYStem.CONFIG.CORE      1.         1.

CORE.NUMber 4.    ;assign the first 4 cores.
                  ;this assignment corresponds to: CORE.ASSIGN 1. 2. 3. 4.
```

Example 2 shows how to assign the cores 3 to 6 of a chip. In our example, chip 1 has 7 cores.



```
;                      <core>    <chip> i.e. start at core 3 of chip 1
SYStem.CONFIG.CORE      3.         1.

CORE.NUMber 4.    ;assign cores 3 to 6, i.e. 4 cores.
                  ;this assignment corresponds to: CORE.ASSIGN 3. 4. 5. 6.
```

See also

- CORE
- CORE.select

CORE.ReMove — Remove core from the SMP system

Format:	**CORE.ReMove** *<core>*
	THREAD.ReMove (deprecated)

Removes a physical core from the SMP system.

See also

■ CORE	■ CORE.select

CORE.select — Change currently selected core

Format:	**CORE.select** *<logical_core>*
	THREAD.select (deprecated)

Changes the currently selected core to the specified *<logical_core>.* As a result the debugger view is changed to *<logical_core>* and all commands without **/CORE** *<number>* option apply to *<logical_core>.*

The number of the selected core is displayed in the state line at the bottom of the TRACE32 main window.

NOTE:	**CORE.List** shows the states of all cores and allows to switch between cores with a simple mouse-click.

See also

■ CORE	■ CORE.ADD	■ CORE.ASSIGN	■ CORE.List
■ CORE.NUMber	■ CORE.ReMove	■ CORE.SHOWACTIVE	■ CORE.SINGLE
■ MACHINE.select	■ TASK.select	❑ CORE()	

Format:	**CORE.SHOWACTIVE**

Opens a window with a color legend, displaying individual colors and numbers for the cores assigned to TRACE32:

- Gray indicates that a core is inactive.

  An inactive core is not executing any code. The debugger can neither control nor talk to this core. A core is inactive if it is not clocked or not powered or held in reset.

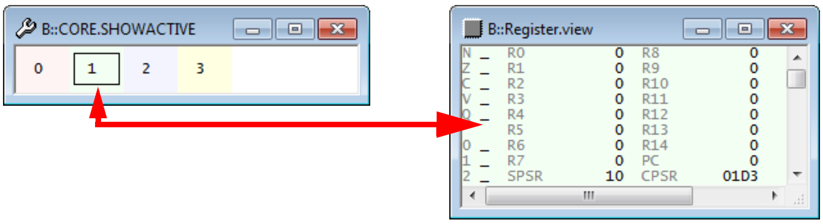
- Colors other than gray (e.g. orange, green, yellow) indicate that a core is active.

  An active core is executing code and the debugger has full control. A core is active if it is clocked, powered and not in reset.

Clicking a number switches the debugger view to the selected core. The window background is highlighted in the same color as the selected core.

For example, when you click **1** in the **CORE.SHOWACTIVE** window, the **Register.view** window updates accordingly. The green background color tells you that this register information refers to core 1 (see screenshots below):

Core 1 = green **Register.view** window = green = Core 1



```
N _  R0      0  R8      0
Z _  R1      0  R9      0
C _  R2      0  R10     0
V _  R3      0  R11     0
Q _  R4      0  R12     0
     R5      0  R13     0
0 _  R6      0  R14     0
1 _  R7      0  PC      0
2 _  SPSR   10  CPSR 01D3
```

Example: Let's assume a multicore chip has 6 cores, and just 4 cores of them are assigned to the TRACE32 PowerView GUI. The **CORE.SHOWACTIVE** window lets you switch between the assigned 4 cores. If you want to pin a window to a particular core, append **/CORE <number>** to the window command (see source example below):

```
;- The cores 1, 2, 4, 5 (= four cores) are assigned to the TRACE32
;  PowerView GUI
;- The cores 3 and 6 are skipped (= two cores)
CORE.ASSIGN 1. 2. 4. 5.
SYStem.Up

;Open the CORE.SHOWACTIVE window. It has four entries because
;four cores were assigned to the TRACE32 PowerView GUI via CORE.ASSIGN
CORE.SHOWACTIVE  ;To select a core, click the core number you want

;alternatively, use this command to select the core you want:
CORE.select 1    ;e.g. select core 1

Register.view       ;displays register information and source listing
Data.List func1     ;from the core currently selected in the
                    ;CORE.SHOWACTIVE window, i.e. core 1

Register.view /CORE 3.    ;displays register information from core 3
Data.List func1 /CORE 3.  ;and source listing from core 3,
                          ;independently of the core currently selected
                          ;in the CORE.SHOWACTIVE window
```

See also

■ CORE	■ CORE.select	■ CmdPOS	■ FramePOS
■ SETUP.COLOR	❏ CORE.ISACTIVE()		

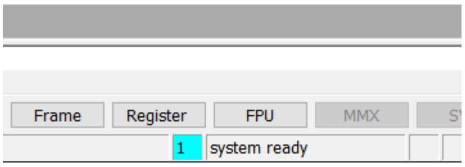
▲ 'PowerView - Screen Display' in 'PowerView User's Guide'

CORE.SINGLE Select single core for debugging

[build 137288 - DVD 09/2021]

Format: **CORE.SINGLE** *<logical_core>*

Selects single core for debugging on SMP systems. As a result the debugger view is changed to *<logical_core>* and all commands, as **Go** and **Step**, are only valid for this core. The core number field in the TRACE32 state line will display the number of the selected core with a turquoise background color.



The command **CORE.select** can be used to revert this selection when the CPU is stopped.

See also

■ CORE	■ CORE.select

Count

Count

Universal counter

See also

■ Count.AutoInit	■ Count.Gate	■ Count.GO	■ Count.Init
■ Count.Mode	■ Count.OUT	■ Count.PROfile	■ Count.RESet
■ Count.Select	■ Count.state	❏ Count.Frequency()	

▲ ’Count’ in ’EPROM/FLASH Simulator’
▲ ’Count Functions’ in ’General Function Reference’
▲ ’Release Information’ in ’Legacy Release History’

Overview Count

Counter of TRACE32-ICD

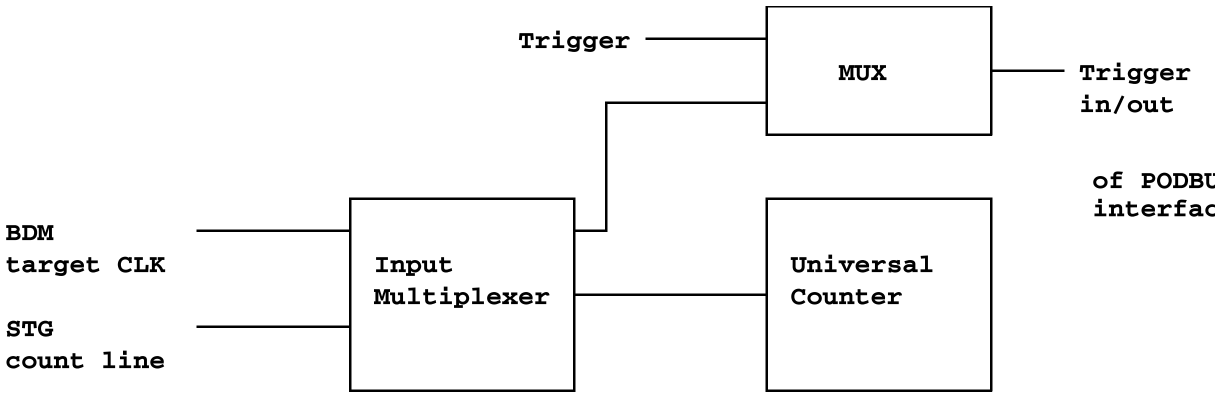
The universal counter system TRACE32-ICD can measure the frequency of the target clock (if the target clock is connected to the debug cable) or the signal on the count line of the **Stimuli Generator** (see **“Stimuli Generator User’s Guide”** (stg_user.pdf)).

The input multiplexer enables the target clock line if a debug module is used and **Count.Select** is entered while the device **B:** (TRACE32-ICD) is selected.

The input multiplexer enables the count line of the Stimuli Generator if a Stimuli Generator is connected and **Count.Select** is entered while the device **ESI:** (EPROM Simulator) is selected.

If only the debug module or only the Stimuli Generator is connected, the input multiplexer enables the present input signal independent of the device selection.

Using the **Count.OUT** command the input signal is issued to the trigger connector on the PODBUS interface. By that the trigger output is disabled.



Counter Functions

To use the result of the measurement in automatic test programs, some functions are defined to get the counter state. The functions are valid only if the **Count.Go** command is executed.

Count.Frequency()

The result of a frequency measurement

Count.LEVEL()

The actual level of the counter signal (Low = 0, High = 1)

Count.Time()

The result of a period or pulse duration measurement

Count.VALUE()

The result of a event count measurement

Count.AutoInit

Automatic counter reset

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

If AutoInit is selected, the counter is initialized when emulation is started (**Go** or **Step**).

See also

■ Count ■ Count.state

▲ 'Count' in 'EPROM/FLASH Simulator'

Count.Gate

Gate time

Format:	**Count.Gate** [<*time*>]
<*time*>:	**0.01s** … **10.0s** **0.** (= infinite gate time)

The gate time has two functions. On measuring frequencies it defines the sample time (gate time). The precision of the measurement increases with the gate time. If pulse measurement is selected, the gate time is the max. time for the pulse width. To measure very long pulses the gate time must be set to infinite.

```
Count.Gate 0.1s                  ; set gate time to 0.1 s

Count.Gate 0.                    ; infinite gate time
```

See also

■ Count ■ Count.state

Count.GO

Start measurement

Format: **Count.GO**

Start single measurement of the frequency counter. This command is usually used only in PRACTICE scripts.

```
Count.Select Cycle
Count.Mode Frequency
Count.Gate 0.1s
Count.GO                        ; start measurement
PRINT COUNT.VALUE()             ; print value
```

See also

■ Count	■ Count.state	❑ Count.Frequency()	❑ Count.Time()
❑ Count.VALUE()			

▲ 'Count' in 'EPROM/FLASH Simulator'

Count.Init

Reset counter

Format: **Count.Init**

The counter is reset (counter value to zero), running measurement cycles are stopped. The counter modes and the channel selection are not changed.

See also

■ Count ■ Count.state

▲ 'Count' in 'EPROM/FLASH Simulator'

Format:	**Count.Mode** [*<mode>*]
<mode>:	**Frequency** **Period** **PulsLow** **PulsHigh** **EventLow** **EventHigh** **EventHOld**

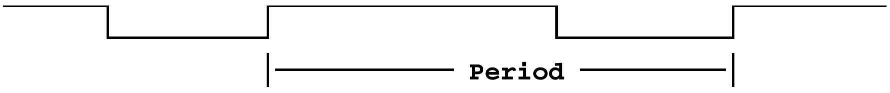
Select mode of the counter.

Frequency

Frequency measurement. The range is up to 20 MHz on external signals and up to 80 MHz for CLOCK and VCO measurement. Depending on the gate time the resolution is from 0.2 Hz to 800 Hz, which is displayed behind the result in the display window.

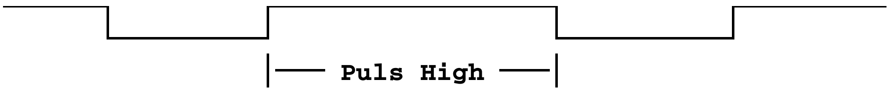
Period

Period time. The resolution is 100 ns, the maximum range up to 300 days



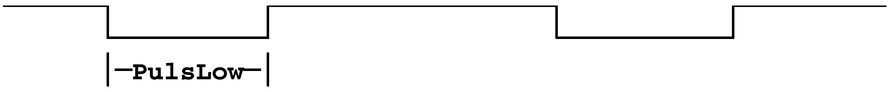
PulsHigh

Measurement of time between the rising and the falling edge



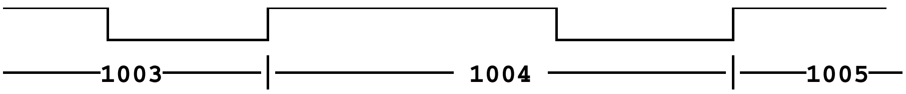
PulsLow

Measurement of time between the falling and the rising edge



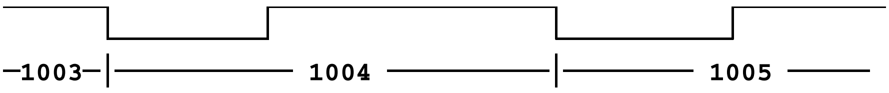
EventHigh

Event count on rising edges



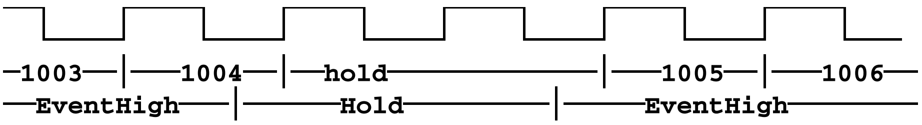
EventLow

Event count on falling edges



EventHOld

The event count is stopped. On starting the previous event count mode, the counter is not cleared.



```
Count.Mode PulsHigh           ; pulse time high

Count.Mode Period             ; period duration

Count.Mode EventHigh          ; event count rising edge
Count.Mode EventHOld          ; stop event count
Count.Mode EventHigh          ; continue event count
```

See also

■ Count ■ Count.state
▲ 'Count' in 'EPROM/FLASH Simulator'

Count.OUT — Forward counter input signal to trigger system/output

Format:	**Count.OUT** [**ON** | **OFF**]

Default: OFF.

When enabled, the input signal of the counter module is forwarded to the Podbus Trigger system. From there it can be used with other devices connected to the Podbus chain. It is also possible to forward the signal to the trigger connector on the debug interface. This is done with **TrBus.Connect Out**.

See also

■ Count ■ Count.state

▲ 'Count' in 'EPROM/FLASH Simulator'

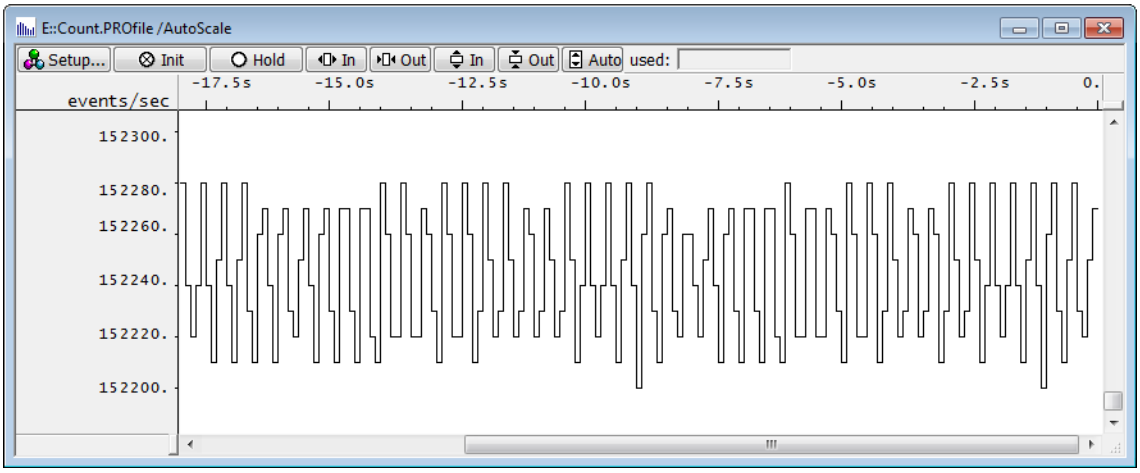
Count.PROfile — Graphic counter display

Format:	**Count.PROfile** [*<gate>*] [*<scale>*]
<gate>:	**0.1s** | **1.0s** | **10.0s**
<scale>:	**1 … 32768.**

The count rate is displayed in graphic mode. The counter mode must be **EventHigh** or **EventLow**. The display is updated and shift every 100 ms or slower. The profiler system is a very effective subsystem to show transfer or interrupt rates in a running system (see also **Analyzer.PROfile**). An opened window may be zoomed by the function keys. An auto zooming feature displays the results always with the best vertical scaling. 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.

NOTE:	Open windows that make dualport memory access may influence the profiling window!

```
;---- profile interrupt rate ----------------------------------

Break.Set INT_routine /Alpha         ; set address mark on beginning of
                                     ; interrupt routine
TrEvent.Select Alpha                 ; set event selector to breakpoint
                                     ; alpha

Count.Mode EventLow                  ; event measurement
Count.Select Event
Go                                   ; start emulation
Count.PROfile                        ; display window

;---- profile data transfer rate -------------------------

Break.Set V.RANGE(buffer1) /Alpha    ; mark buffer area
TrEvent.Select Alpha                 ; set event selector to breakpoint
                                     ; alpha

Count.Mode EventLow                  ; event measurement
Count.Select Event
Go                                   ; start emulation
Count.PROfile                        ; display window
```



See also

■ Count ■ Count.state

▲ 'Count' in 'EPROM/FLASH Simulator'

Count.RESet

Reset command

Format:	**Count.RESet**

The counter system is initialized to the reset state after power up.

See also

■ Count ■ Count.state

▲ 'Count' in 'EPROM/FLASH Simulator'

Count.Select

Select input source

Format:	**Count.Select** [*<signal>*]
<signal>:	**VCO** **Clock** **CYcle** **ExtComp** **EXT** **Event** **PODBUS** **Port** **AlphaBreak** **BetaBreak** **CharlyBreak** **OUTD** **RESet** **Halt** **BusReq** **BusErr** **Vpa** **VCC** **BusGrant** **BusGrantAck** **E0** | **E1** | **E2** | **E3** | **E4** | **E5** | **E6** | **E7** **T0** | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** **T8** | **T9** | **T10** | **T11** | **T12** | **T13** | **T14** | **T15** **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7**

Count.Select controls the input multiplexer of the universal counter. The selected signal (named SIG) may be used as trigger source too. To see this signal on the EVENT output on the rear of the ECU box, use the **TriggerEvent.Select** command.

Clock	Clock frequency of the emulation CPU (external or internal)
CYcle	OUT.D Signal of the trigger unit (additional event counter or profiler).
RESet, Halt, Bus-Req, BusGrant, …	Cycle signal of emulation CPU. Normally generated by the data strobe.
PODBUS	Signal selected by the external PODBUS probes.
Port	This signal is the channel selected by the port analyzer.
B0, B1, B2, B3, B4, B5, B6, B7	Inputs lines on BANK probe
T0, T1, T2, T3, T4, T5, T6, T7	Input lines on TRIGGER probe

See also

■ Count ■ Count.state

▲ 'Count' in 'EPROM/FLASH Simulator'

Count.state State display

Format:	**Count.state**

Displays the measurement value and setup of the frequency counter. The number of channels and the configuration depends on the development tool and the CPU used.

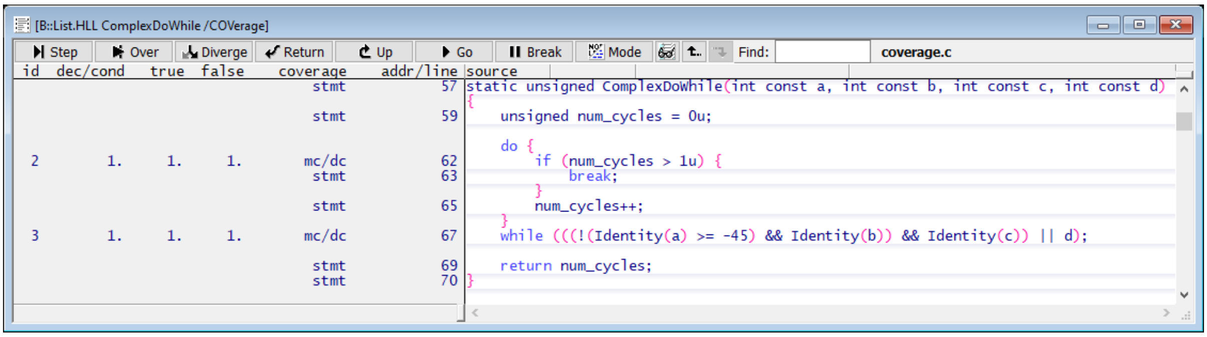
See also

■ Count	■ Count.AutoInit	■ Count.Gate	■ Count.GO
■ Count.Init	■ Count.Mode	■ Count.OUT	■ Count.PROfile
■ Count.RESet	■ Count.Select	❏ Count.Frequency()	❏ Count.LEVEL()
❏ Count.Time()	❏ Count.VALUE()		

▲ 'Count' in 'EPROM/FLASH Simulator'

COVerage — Trace-based code coverage

The **COVerage** command group uses the program flow information from the trace for a detailed code coverage analysis. The manual **"Application Note for Trace-Based Code Coverage"** (app_code_coverage.pdf) gives a detailed introduction to the topic.



```
[B::List.HLL ComplexDoWhile /COVerage]
id  dec/cond  true  false   coverage  addr/line source
                              stmt         57 static unsigned ComplexDoWhile(int const a, int const b, int const c, int const d)
                                              {
                              stmt         59     unsigned num_cycles = 0u;

                                                  do {
 2       1.      1.     1.    mc/dc        62         if (num_cycles > 1u) {
                              stmt         63             break;
                                                      }
                              stmt         65         num_cycles++;
                                                  }
 3       1.      1.     1.    mc/dc        67     while (((!(Identity(a) >= -45) && Identity(b)) && Identity(c)) || d);

                              stmt         69     return num_cycles;
                              stmt         70 }
```

A demo script is included in your TRACE32 installation. To access the script, run this command:

```
ChDir.PSTEP ~~/demo/t32cast/eca/measure_mcdc.cmm
```

See also

■ COVerage.ACCESS	■ COVerage.ADD	■ COVerage.Delete	■ COVerage.EXPORT
■ COVerage.Init	■ COVerage.List	■ COVerage.ListCalleEs	■ COVerage.ListCalleRs
■ COVerage.ListFunc	■ COVerage.ListInlineBlock	■ COVerage.ListLine	■ COVerage.ListModule
■ COVerage.ListVar	■ COVerage.LOAD	■ COVerage.MAP	■ COVerage.METHOD
■ COVerage.Mode	■ COVerage.OFF	■ COVerage.ON	■ COVerage.Option
■ COVerage.RESet	■ COVerage.SAVE	■ COVerage.Set	■ COVerage.state
■ COVerage.StaticInfo	■ COVerage.TreeWalkSETUP	■ BookMark	■ RTS
❏ COVerage.BDONE()	❏ COVerage.SCOPE()	❏ COVerage.SourceMetric()	❏ COVerage.TreeWalk()

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Introduction' in 'Application Note for t32cast'
▲ 'COVerage Functions' in 'General Function Reference'

COVerage.ACCESS   Set the memory access mode

Format:	**COVerage.ACCESS** [**auto** | **VM** | **DualPort**] (deprecated) **Use Trace.ACCESS instead.**

See also

■ COVerage ■ COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.ADD   Add trace contents to database

Format:	**COVerage.ADD** [**/**<*option*>] <*trace*>**.COVerage.add** (deprecated)
<*option*>:	**FILE** **FlowTrace** | **BusTrace**

The trace contents is processed and added to the coverage database.

FILE	Takes trace memory contents loaded by **Trace.FILE**.
FlowTrace	The trace works as a program flow trace. This option is usually not required.
BusTrace	Trace works as a bus trace. This option is usually not required.

Example:

```
Analyzer.Mode Stack             ; clear trace buffer and use stack mode
Go sieve                        ; run a part of the application
…
COVerage.ADD                    ; process the trace buffer
```

See also

■ COVerage ■ COVerage.state

▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'

COVerage.Delete

Modify coverage

Format:	**COVerage.Delete** [*<address>* | *<range>*]

Marks the defined range as '**never**' executed.

See also

■ COVerage ■ COVerage.state

▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'

COVerage.EXPORT Export code coverage information to an XML file

Using the **COVerage.EXPORT** commands, you can export code coverage information for all HLL functions, lines, modules, or variables to an XML file.

In addition, TRACE32 provides an XSL transformation template for formatting the XML file. The formatting is automatically applied to the XML file when it is opened in an external browser window. Prerequisite: The XSL file is placed in the same folder as the XML file.

For an export example and demo scripts, see **COVerage.EXPORT.ListFunc**.

See also

■ COVerage.EXPORT.CBA	■ COVerage.EXPORT.CSV
■ COVerage.EXPORT.JSON	■ COVerage.EXPORT.JSONE
■ COVerage.EXPORT.ListCalleEs	■ COVerage.EXPORT.ListCalleEs.<sub_cmd>
■ COVerage.EXPORT.ListCalleRs	■ COVerage.EXPORT.ListCalleRs.<sub_cmd>
■ COVerage.EXPORT.ListFunc	■ COVerage.EXPORT.ListFunc.<sub_cmd>
■ COVerage.EXPORT.ListInlineBlock	■ COVerage.EXPORT.ListInlineBlock.<sub_cmd>
■ COVerage.EXPORT.ListLine	■ COVerage.EXPORT.ListLine.<sub_cmd>
■ COVerage.EXPORT.ListModule	■ COVerage.EXPORT.ListModule.<sub_cmd>
■ COVerage.EXPORT.ListVar	■ COVerage.EXPORT.ListVar.<sub_cmd>
■ COVerage	■ COVerage.state
■ ISTATistic.EXPORT	■ List.EXPORT
■ SETUP.XSLTSTYLESHEET	

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.EXPORT.CBA Export HLL lines in CBA format

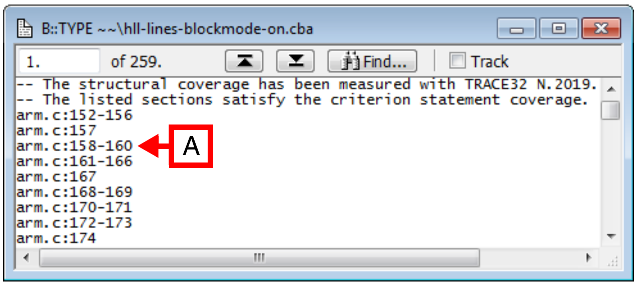
Format:	**COVerage.EXPORT.CBA** *<file>* [**/Append**]

Exports coverage information about HLL lines to a file in CBA format for import to VectorCAST/CBA.

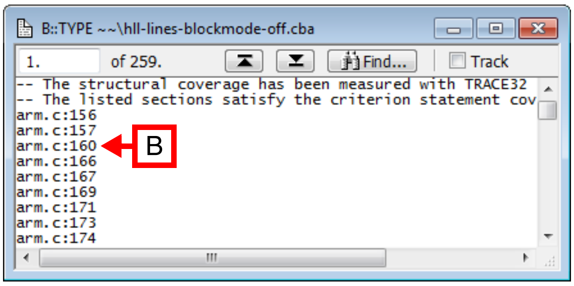
<file>	The default extension of the file name is ***.cba**. If you omit the extension, it is added automatically on file creation.

Example:

```
COVerage.state
COVerage.Option.SourceMetric Statement
COVerage.Option.BLOCKMode ON  ;for a comparison of ON and OFF, see below

COVerage.EXPORT.CBA ~~\hll-lines-blockmode-on.cba
```



```
-- The structural coverage has been measured with TRACE32 N.2019.
-- The listed sections satisfy the criterion statement coverage.
arm.c:152-156
arm.c:157
arm.c:158-160
arm.c:161-166
arm.c:167
arm.c:168-169
arm.c:170-171
arm.c:172-173
arm.c:174
```



```
-- The structural coverage has been measured with TRACE32
-- The listed sections satisfy the criterion statement cov
arm.c:156
arm.c:157
arm.c:160
arm.c:166
arm.c:167
arm.c:169
arm.c:171
arm.c:173
arm.c:174
```

A With **COVerage.Option.BLOCKMode ON**, the line number range for each entry is printed.

B With **COVerage.Option.BLOCKMode OFF**, only the number of the last line is printed.

See also

■ COVerage.EXPORT
■ COVerage.Option.BLOCKMode
▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.EXPORT.CSV

Export coverage data in CSV format

Format:	**COVerage.EXPORT.CSV** *<file>* [*<string>* | *<range>*] [**/***<option>*]
<option>:	**Append**

Exports coverage information in CSV format.

<file>	The default extension of the file name is ***.csv**. If you omit the extension, it is added automatically on file creation.

For a description of the command line arguments, a screenshot of an export example, and PRACTICE script examples, see **COVerage.EXPORT.ListFunc**.

Append	Appends the coverage information to an existing CSV file - without overwriting the current file contents.

See also

■ COVerage.EXPORT
▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.EXPORT.JSON — Export code coverage in JSON format

Format:	**COVerage.EXPORT.JSON** *<file>*

Exports coverage information about functions and lines to a file in JSON format compatible to Gcov.

<file>	The default extension of the file name is ***.json**. If you omit the extension, it is added automatically on file creation.

Example:

```
; Process trace data for coverage
COVerage.Add
; Process trace for ISTATistic (needed for export of execution count)
ISTATistic.Add
; Export to JSON
COVerage.EXPORT.JSON ~~\result.json
```

See also

■ COVerage.EXPORT.JSONE ■ COVerage.EXPORT

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.EXPORT.JSONE — Export code coverage in JSONE format

Format:	**COVerage.EXPORT.JSONE** *<file>*

Exports coverage information about functions and lines to a file in JSONE format compatible to Gcov.

<file>	The default extension of the file name is ***.jsone**. If you omit the extension, it is added automatically on file creation.

Example:

```
; Process trace data for coverage
COVerage.Add
; Process trace for ISTATistic (needed for export of execution count)
ISTATistic.Add
; Export to JSONE
COVerage.EXPORT.JSONE ~~\result.jsone
```

See also

■ COVerage.EXPORT.JSON ■ COVerage.EXPORT

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

See also

- ■ COVerage.EXPORT.ListCalleEs.<sub_cmd>
- ■ COVerage.EXPORT
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
- ▲ 'Release Information' in 'Legacy Release History'

Format:	**COVerage.EXPORT.ListCalleEs.**<*sub_cmd*>
<*sub_cmd*>:	**ADDRESS** [<*file*>] [<*source_file*>...] [**/**<*option*>] **preset** [<*file*>] [%<*format*>] [<*filter*>] [**/**<*option*>] **SOURCE** [<*file*>] [<*source_file*>...] [**/**<*option*>] **sYmbol** [<*file*>] [<*symbol*>…] [**/**<*option*>]
<*option*>:	**Append**

Exports coverage information for function callees to an XML file.

The following <*sub_cmd*> are possible:

ADDRESS	Uses addresses to control which coverage information for function callees to export.
preset	If the command contains no parameters, then all function callees are exported. The <*filter*> parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Uses the names of source files to control which coverage information for function callees to export. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the HLL function callees to export.
<*file*>, <*option*>	For descriptions, see **COVerage.EXPORT.ListFunc**.
<*source_file*>, <*symbol*>	You can use one or more items as filter criteria. The wildcards '*****' and '**?**' are supported. Only items matching the filter criteria are displayed.

See also

- COVerage.EXPORT.ListCalleEs
- COVerage.ListCalleEs
- COVerage.EXPORT

See also

■ COVerage.EXPORT.ListCalleRs.<sub_cmd> ■ COVerage.EXPORT

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.EXPORT.ListCalleRs.<sub_cmd>

Format:	**COVerage.EXPORT.ListCalleRs.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<file>*] [*<source_file>*...] [**/***<option>*] **preset** [*<file>*] [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<file>*] [*<source_file>*...] [**/***<option>*] **sYmbol** [*<file>*] [*<symbol>*…] [**/***<option>*]
<option>:	**Append**

Exports coverage information for function callers to an XML file.

The following *<sub_cmd>* are possible:

ADDRESS	Uses addresses to control which coverage information for function callers to export.
preset	If the command contains no parameters, then all function callers are exported. The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Uses the names of source files to control which coverage information for function callers to export. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the HLL function callers to export.
<file>, *<option>*	For descriptions, see **COVerage.EXPORT.ListFunc**.
<source_file>, *<symbol>*	You can use one or more items as filter criteria. The wildcards '`*`' and '`?`' are supported. Only items matching the filter criteria are displayed.

See also

- COVerage.EXPORT.ListCalleRs
- COVerage.ListCalleRs
- COVerage.EXPORT

[Examples]

See also

- ■ COVerage.EXPORT.ListFunc.<sub_cmd>
- ■ COVerage.EXPORT
- ■ COVerage.ListFunc
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.EXPORT.ListFunc.<sub_cmd> Export HLL function information

Format:	**COVerage.EXPORT.ListFunc.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<file>*] [*<source_file>*...] [**/***<option>*] **preset** [*<file>*] [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<file>*] [*<source_file>*...] [**/***<option>*] **sYmbol** [*<file>*] [*<symbol>*…] [**/***<option>*]
<option>:	**Append**

Exports coverage information about the HLL function to an XML file.

The following *<sub_cmd>* are possible:

ADDRESS	Exports code coverage information for functions filtered by source file.
preset	If the command contains no parameters, then all the HLL function are exported. The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Exports code coverage information for source code lines filtered by source file. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the HLL function to export.
<file>, *<option>*	For descriptions, see **COVerage.EXPORT.ListFunc**.
<source_file>, *<symbol>*	You can use one or more items as filter criteria. The wildcards '*' and '**?**' are supported. Only items matching the filter criteria are displayed.

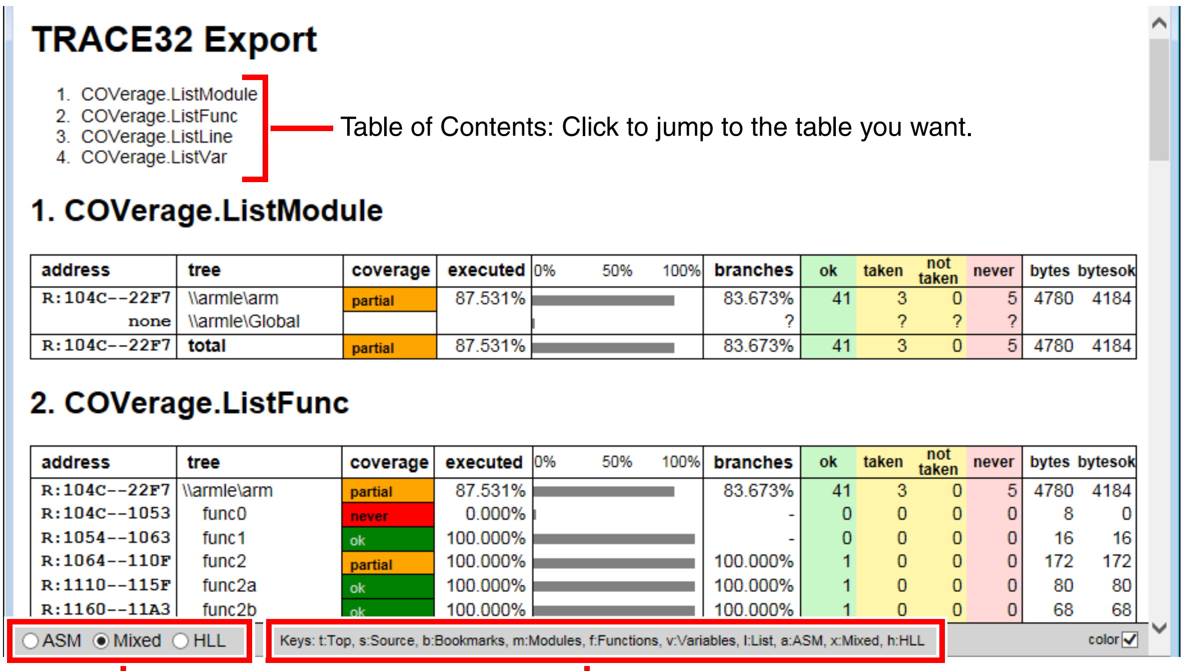
Examples:

```
COVerage.EXPORT.ListFunc.ADDRESS output.xml P:0x0000000--0x9000000
```

```
COVerage.EXPORT.ListFunc.SOURCE output.xml "*sieve.c"
```

```
;In this script line, only the symbol main as well as symbols matching
;the patterns func? and *eve* are exported
COVerage.EXPORT.ListFunc.sYmbol ~~\coverage.xml   main   func?   *eve*
```

Example of an XML export file opened in an external browser window:



TRACE32 Export

1. COVerage.ListModule
2. COVerage.ListFunc
3. COVerage.ListLine
4. COVerage.ListVar

Table of Contents: Click to jump to the table you want.

1. COVerage.ListModule

address	tree	coverage	executed	0% 50% 100%	branches	ok	taken	not taken	never	bytes	bytesok
R:104C--22F7	\\armle\arm	partial	87.531%		83.673%	41	3	0	5	4780	4184
none	\\armle\Global				?		?	?	?		
R:104C--22F7	total	partial	87.531%		83.673%	41	3	0	5	4780	4184

2. COVerage.ListFunc

address	tree	coverage	executed	0% 50% 100%	branches	ok	taken	not taken	never	bytes	bytesok
R:104C--22F7	\\armle\arm	partial	87.531%		83.673%	41	3	0	5	4780	4184
R:104C--1053	func0	never	0.000%		-	0	0	0	0	8	0
R:1054--1063	func1	ok	100.000%		-	0	0	0	0	16	16
R:1064--110F	func2	partial	100.000%		100.000%	1	0	0	0	172	172
R:1110--115F	func2a	ok	100.000%		100.000%	1	0	0	0	80	80
R:1160--11A3	func2b	ok	100.000%		100.000%	1	0	0	0	68	68

ASM [✓ Mixed] HLL

Keys: t:Top, s:Source, b:Bookmarks, m:Modules, f:Functions, v:Variables, l:List, a:ASM, x:Mixed, h:HLL

color ✓

Click to toggle the display of the listing.

Press these keys to jump to the table you want.

<file>

Name of the XML file that stores the code coverage information. The file extension *.xml can be omitted.

<string>

Defines a filter for the source files that you want to export. The filter consists of the file path and refers only to source files that are listed in the **tree** column of a **COVerage.ListFunc**, **COVerage.ListModule**, etc. window.

Example for <string>:

```
;export the code coverage information for all HLL functions with
;a source path that matches the pattern "*/gnu/sub/*"
COVerage.EXPORT.ListFunc C:\t32\coverage.xml "*/gnu/sub/*"

;export the code coverage information for all modules with a file path
;that matches the pattern "*crt0.s"
COVerage.EXPORT.ListModule C:\t32\coverage.xml "*crt0.s"  /Append
```

<range>

Filter for exporting the specified address range or symbol range.

The address range can be specified as follows:

- Start and end address.
- Only start address. Exports items from the start address up to the maximum address of the current address space.

The symbol range can be specified as program, module, or function.

Example: This script line exports code coverage information for three symbol ranges.

```
;export the code coverage information for three symbol ranges
COVerage.EXPORT.ListFunc C:\t32\coverage.xml \\myprog\func13  func10 \
                                                                  \\prog2
```

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

APPEND

Appends the coverage information to an existing XML file - without overwriting the current file contents.

SOrder, TOrder

SOrder	Sort in source line order.
TOrder	Sort by address.

Example 1:

Prerequisite: The debug symbols have been loaded and trace data has been recorded.

This script shows how to export code coverage information for all modules, HLL functions, lines, and variables to the same XML file. The formatted file is then opened in an external browser window.

```
COVerage.ADD                ;update the coverage database
COVerage.ListModule         ;display coverage of all modules
COVerage.ListFunc           ;display coverage of all functions
COVerage.ListLine           ;display coverage of all source lines
COVerage.ListVar            ;display coverage of all variables

;export the code coverage information for all modules of
;program “armle”
COVerage.EXPORT.ListModule "~~/coverage.xml" \\armle

;export the code coverage information for all HLL functions of the
;module “arm” and append to an existing file
COVerage.EXPORT.ListFunc   "~~/coverage.xml" \arm   /Append

;export the code coverage information for all HLL lines of the
;function “sieve” and append to an existing file
COVerage.EXPORT.ListLine   "~~/coverage.xml" sieve  /Append

;export the code coverage information for HLL variables
;and append to an existing file
COVerage.EXPORT.ListVar    "~~/coverage.xml" ,      /Append

;for demo purposes: let's open the unformatted result in TRACE32
EDIT "~~/coverage.xml"

;place the transformation template in the same folder as the XML file
COPY "~~/demo/coverage/single_file_report/t32transform.xsl" \
     "~~/t32transform.xsl"

;you can now open the formatted result in an external browser window
OS.Command start iexplore.exe "file:///C:/t32/coverage.xml"
```

The tildes ~~ expand to your TRACE32 system directory, (e.g. C:\T32).

Example 2:

A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```
CD.PSTEP ~~/demo/coverage/example.cmm
```

This demo script also tells you how to include a listing in the XML export file.

See also

- COVerage.EXPORT.ListFunc
- COVerage.ListFunc
- COVerage.EXPORT

See also

- ■ COVerage.EXPORT.ListInlineBlock.<sub_cmd>
- ■ COVerage.EXPORT
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

Format:	**COVerage.EXPORT.ListInlineBlock.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<file>*] [*<source_file>*...] [**/***<option>*] **preset** [*<file>*] [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<file>*] [*<source_file>*...] [**/***<option>*] **sYmbol** [*<file>*] [*<symbol>*…] [**/***<option>*]
<option>:	**Append**

Exports coverage information about inlined code blocks to an XML file.

The following *<sub_cmd>* are possible:

ADDRESS	Uses addresses to control which coverage information for inlined code blocks to export.
preset	If the command contains no parameters, then all inlined code blocks are exported. The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Uses the names of source files to control which coverage information for inlined code blocks to export. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the inlined code blocks to export.
<file>, *<option>*	For descriptions, see **COVerage.EXPORT.ListFunc**.
<source_file>, *<symbol>*	You can use one or more items as filter criteria. The wildcards '*****' and '**?**' are supported. Only items matching the filter criteria are displayed.

See also

- COVerage.EXPORT.ListInlineBlock
- COVerage.ListInlineBlock
- COVerage.EXPORT

See also

- ■ COVerage.EXPORT.ListLine.<sub_cmd>
- ■ COVerage.EXPORT
- ■ COVerage.ListLine

▲ ’Introduction’ in ’Application Note for Trace-Based Code Coverage’

Format:	**COVerage.EXPORT.ListLine.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<file>*] [*<source_file>*...] [**/***<option>*] **preset** [*<file>*] [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<file>*] [*<source_file>*...] [**/***<option>*] **sYmbol** [*<file>*] [*<symbol>*…] [**/***<option>*]
<option>:	**Append**

Exports coverage information about HLL lines to an XML file.

The following *<sub_cmd>* are possible:

ADDRESS	Uses addresses to control which coverage information for source code lines to export.
preset	If the command contains no parameters, then all HLL lines are exported. The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Uses the names of source files to control which coverage information for source code lines to export. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the HLL lines to export.
<file>, *<option>*	For descriptions, see **COVerage.EXPORT.ListFunc**.
<source_file>, *<symbol>*	You can use one or more items as filter criteria. The wildcards '*****' and '**?**' are supported. Only items matching the filter criteria are displayed.

See also

■ COVerage.EXPORT.ListLine ■ COVerage.EXPORT ■ COVerage.ListLine

See also

- ■ COVerage.EXPORT.ListModule.<sub_cmd>
- ■ COVerage.ListModule
- ■ COVerage.EXPORT
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.EXPORT.ListModule.<sub_cmd> Export modules information

Format:	**COVerage.EXPORT.ListModule.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<file>*] [*<source_file>*...] [**/***<option>*] **preset** [*<file>*] [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<file>*] [*<source_file>*...] [**/***<option>*] **sYmbol** [*<file>*] [*<symbol>*…] [**/***<option>*]
<option>:	**Append**

Exports coverage information bout modules to an XML file.

The following *<sub_cmd>* are possible:

ADDRESS	Uses addresses to control which coverage information for modules to export.
preset	If the command contains no parameters, then all modules are exported. The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Uses the names of source files to control which coverage information for modules to export. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the modules to export.
<file>, *<option>*	For descriptions, see **COVerage.EXPORT.ListFunc**.
<source_file>, *<symbol>*	You can use one or more items as filter criteria. The wildcards '*****' and '**?**' are supported. Only items matching the filter criteria are displayed.

See also

- ■ COVerage.EXPORT.ListModule
- ■ COVerage.ListModule
- ■ COVerage.EXPORT

See also

- ■ COVerage.EXPORT.ListVar.<sub_cmd>
- ■ COVerage.EXPORT
- ■ COVerage.ListVar

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.EXPORT.ListVar.<sub_cmd> Export HLL variables information

Format:	**COVerage.EXPORT.ListVar.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<file>*] [*<source_file>*...] [**/***<option>*] **preset** [*<file>*] [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<file>*] [*<source_file>*...] [**/***<option>*] **sYmbol** [*<file>*] [*<symbol>*…] [**/***<option>*]
<option>:	**Append**

Exports coverage information for HLL variables to an XML file.

The following *<sub_cmd>* are possible:

ADDRESS	Uses addresses to control which coverage information for variables to export.
preset	If the command contains no parameters, then all HLL variables are exported. The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis.
SOURCE	Uses the names of source files to control which coverage information for variables to export. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols of the HLL variables to export.
<file>, <option>	For descriptions, see **COVerage.EXPORT.ListFunc**.
<source_file>, <symbol>	You can use one or more items as filter criteria. The wildcards '`*`' and '`?`' are supported. Only items matching the filter criteria are displayed.

See also

- ■ COVerage.EXPORT.ListVar
- ■ COVerage.EXPORT
- ■ COVerage.ListVar

COVerage.Init — Clear coverage database

Format: **COVerage.Init**
<trace>.**COVerage.Init** (deprecated)

Deletes all code coverage information for HLL source code statements, assembly instructions and data values.

See also

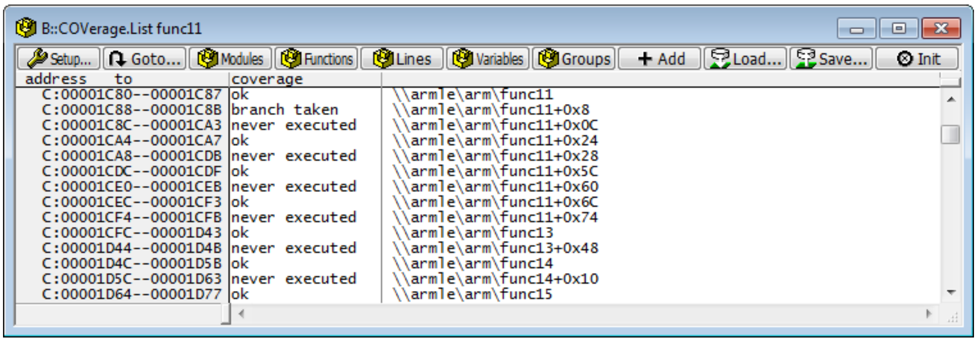
■ COVerage ■ COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.List — Coverage display

Format: **COVerage.List** [*<address>* | *<range>*]
<trace>.**COVerage.List** (deprecated)

Displays the results of the coverage analysis.



```
address    to          coverage
   C:00001C80--00001C87 ok                \\arml e\arm\func11
   C:00001C88--00001C8B branch taken      \\arml e\arm\func11+0x8
   C:00001C8C--00001CA3 never executed    \\arml e\arm\func11+0x0C
   C:00001CA4--00001CA7 ok                \\arml e\arm\func11+0x24
   C:00001CA8--00001CDB never executed    \\arml e\arm\func11+0x28
   C:00001CDC--00001CDF ok                \\arml e\arm\func11+0x5C
   C:00001CE0--00001CEB never executed    \\arml e\arm\func11+0x60
   C:00001CEC--00001CF3 ok                \\arml e\arm\func11+0x6C
   C:00001CF4--00001CFB never executed    \\arml e\arm\func11+0x74
   C:00001CFC--00001D43 ok                \\arml e\arm\func13
   C:00001D44--00001D4B never executed    \\arml e\arm\func13+0x48
   C:00001D4C--00001D5B ok                \\arml e\arm\func14
   C:00001D5C--00001D63 never executed    \\arml e\arm\func14+0x10
   C:00001D64--00001D77 ok                \\arml e\arm\func15
```

Double-clicking a line opens a **List** window, showing the context of and more details about the covered code.

See also

■ COVerage.ListCalleEs ■ COVerage.ListFunc ■ COVerage.ListLine ■ COVerage.ListModule
■ COVerage.ListVar ■ COVerage ■ COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

See also

- COVerage.ListCalleRs
- COVerage.ListFunc
- COVerage.ListLine
- COVerage.ListVar
- COVerage
- COVerage.state
- COVerage.List
- COVerage.ListInlineBlock
- COVerage.ListModule
- COVerage.LOAD
- COVerage.EXPORT.ListCalleEs.<sub_cmd>

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.ListCalleEs.<sub_cmd> Display coverage for callees function

Format:	**COVerage.ListCalleEs.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [%*<format>*] [*<address>* | *<address_range>*] [**/***<option>*] **preset** [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [%*<format>*] [*<source>...*] [**/***<option>*] **sYmbol** [%*<format>*] [*<symbol>…*] [**/***<option>*]
<format>:	**SINGLE** | **MULTI** | **DO178**
<option>:	**SOrder** | **TOrder**

Displays the results of the code coverage analysis related to function callees. If the metric Call is set (see **COVerage.Option SourceMetric Call**) callee details are part of the report generated with the help of the TRACE32 Coverage Report Utility.

The following <sub_cmd> are possible:

ADDRESS	Allows to restrict the displayed function callees to a specified address range.
preset	If the command contains no parameters, then all function callees are displayed (see example 1). The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis (see example 2).
SOURCE	Allows to restrict the displayed function callees to the specified source files. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window (see example 3).

sYmbol	Allows to restrict the displayed callees to the specified symbol ranges. The symbol names are oriented towards the **symbol** column in the **sYmbol.Browse.Function** or **sYmbol.Browse.Module** window (see example 4).
<symbol>, <source>	You can use one or more items as filter criteria. The wildcards '`*`' and '`?`' are supported. Only items matching the filter criteria are displayed. See commands **sYmbol.FILTER.ADD.sYmbol** and **sYmbol.FILTER.ADD.SOURCE**.

Format Parameters SINGLE, MULTI, DO178

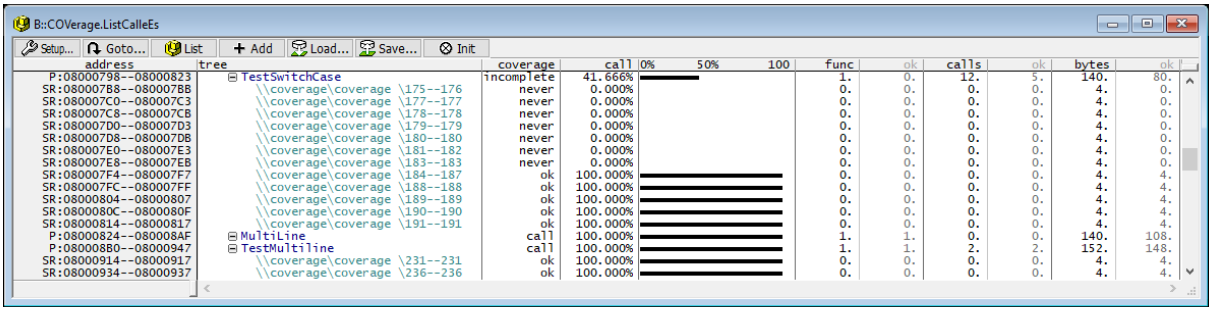
SINGLE	The code coverage results are displayed only for the selected metric.
MULTI	The code coverage results are displayed for the selected metric and all included metrics. E.g. the MCDC metric includes also CONDition, Decision and Statement coverage.
DO178	The code coverage results are displayed for the selected metric and all included metrics that are relevant for DO178. E.g. the MCDC metric includes also Decision and Statement coverage.

Option SOrder, TOrder

SOrder	Display the source code lines belonging to a function in **source order**.
TOrder	Display the source code lines belonging to a function in **target order** (default).

Example 1:

```
COVerage.Option SourceMetric Call

…

COVerage.ListCalleEs
```



```
         address        tree                                      coverage       call 0%   50%   100   func    ok   calls    ok   bytes    ok
  P:08000798--08000823      ⊟ TestSwitchCase                      incomplete   41.666%                  1.     0.     12.     5.   140.   80.
 SR:080007B8--080007BB         \\coverage\coverage \175--176          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007C0--080007C3         \\coverage\coverage \177--177          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007C8--080007CB         \\coverage\coverage \178--178          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007D0--080007D3         \\coverage\coverage \179--179          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007D8--080007DB         \\coverage\coverage \180--180          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007E0--080007E3         \\coverage\coverage \181--182          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007E8--080007EB         \\coverage\coverage \183--183          never    0.000%                  0.     0.      0.     0.     4.    0.
 SR:080007F4--080007F7         \\coverage\coverage \184--187             ok  100.000%                  0.     0.      0.     0.     4.    4.
 SR:080007FC--080007FF         \\coverage\coverage \188--188             ok  100.000%                  0.     0.      0.     0.     4.    4.
 SR:08000804--08000807         \\coverage\coverage \189--189             ok  100.000%                  0.     0.      0.     0.     4.    4.
 SR:0800080C--0800080F         \\coverage\coverage \190--190             ok  100.000%                  0.     0.      0.     0.     4.    4.
 SR:08000814--08000817         \\coverage\coverage \191--191             ok  100.000%                  0.     0.      0.     0.     4.    4.
  P:08000824--080008AF      ⊟ MultiLine                                 call  100.000%                  1.     1.      0.     0.   140.  108.
  P:080008B0--08000947      ⊟ TestMultiline                             call  100.000%                  1.     1.      2.     2.   152.  148.
 SR:08000914--08000917         \\coverage\coverage \231--231             ok  100.000%                  0.     0.      0.     0.     4.    4.
 SR:08000934--08000937         \\coverage\coverage \236--236             ok  100.000%                  0.     0.      0.     0.     4.    4.
```

Double-clicking a line displays the function or call and detailed information about the code coverage in a **List** window.

Example 2:

```
COVerage.Option SourceMetric Call

…

sYmbol.Browse.Module

sYmbol.FILTER.ADD.sYmbol jd_modules \jdcolor \jdmarker \jdtrans

COVerage.ListCalleEs.preset jd_modules
```

Example 3:

```
sYmbol.Browse.SOURCE

COVerage.ListCalleEs.SOURCE \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdapistd.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdinput.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdpostct.c"
```

```
COVerage.ListCalleEs.SOURCE \"*jdc*.c"
```

Example 4:

```
sYmbol.Browse.Module

COVerage.ListCalleEs.sYmbol \jdapistd \jdmaster \jidctred
```

```
COVerage.ListCalleEs.sYmbol \jda*
```

See also

- COVerage.ListCalleEs
- COVerage.EXPORT.ListCalleRs.<sub_cmd>
- COVerage
- COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.ListCalleRs.<sub_cmd> Display coverage for callers function

Format:	**COVerage.ListCalleRs.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [%*<format>*] [*<address>* | *<address_range>*] [**/***<option>*] **preset** [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [%*<format>*] [*<source>...*] [**/***<option>*] **sYmbol** [%*<format>*] [*<symbol>…*] [**/***<option>*]
<format>:	**SINGLE** | **MULTI** | **DO178**
<option>:	**SOrder** | **TOrder**

Displays the results of the code coverage analysis related to function callees. If the metric Call is set (see **COVerage.Option SourceMetric Call**) callee details are part of the report generated with the help of the TRACE32 Coverage Report Utility.

The following <sub_cmd> are possible:

ADDRESS	Allows to restrict the displayed function callers to a specified address range.
preset	If the command contains no parameters, then all function callers are displayed (see example 1). The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis (see example 2).
SOURCE	Allows to restrict the displayed function callers to the specified source files. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window (see example 3).

sYmbol	Allows to restrict the displayed callers to the specified symbol ranges. The symbol names are oriented towards the **symbol** column in the **sYmbol.Browse.Function** or **sYmbol.Browse.Module** window (see example 4).
<symbol>, <source>	You can use one or more items as filter criteria. The wildcards '`*`' and '`?`' are supported. Only items matching the filter criteria are displayed. See commands **sYmbol.FILTER.ADD.sYmbol** and **sYmbol.FILTER.ADD.SOURCE**.

Format Parameters SINGLE, MULTI, DO178

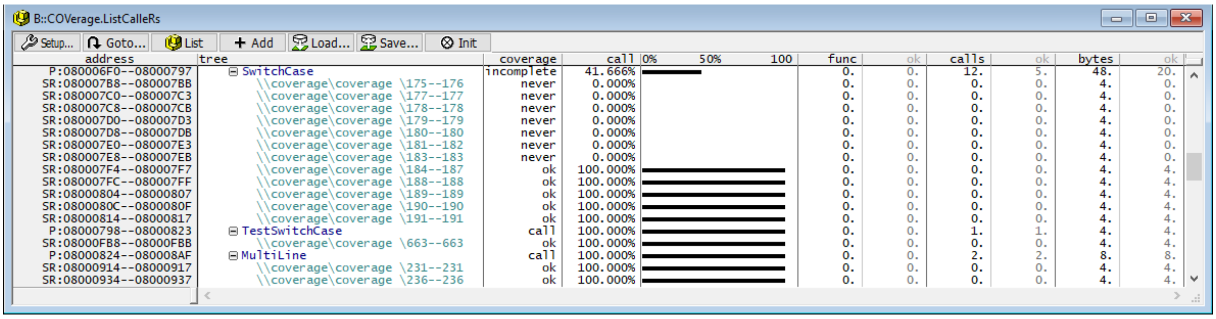
SINGLE	The code coverage results are displayed only for the selected metric.
MULTI	The code coverage results are displayed for the selected metric and all included metrics. E.g. the MCDC metric includes also CONDition, Decision and Statement coverage.
DO178	The code coverage results are displayed for the selected metric and all included metrics that are relevant for DO178. E.g. the MCDC metric includes also Decision and Statement coverage.

Option SOrder, TOrder

SOrder	Display the source code lines belonging to a function in **source order**.
TOrder	Display the source code lines belonging to a function in **target order** (default).

Example 1:

```
COVerage.Option SourceMetric Call

...

COVerage.ListCalleRs
```



```
            address  tree                                 coverage     call 0%      50%     100     func      ok    calls      ok    bytes      ok
  P:080006F0--08000797  ⊟ SwitchCase                      incomplete  41.666%                        0.      0.      12.      5.      48.     20.
 SR:080007B8--080007BB     \\coverage\coverage \175--176      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007C0--080007C3     \\coverage\coverage \177--177      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007C8--080007CB     \\coverage\coverage \178--178      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007D0--080007D3     \\coverage\coverage \179--179      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007D8--080007DB     \\coverage\coverage \180--180      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007E0--080007E3     \\coverage\coverage \181--182      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007E8--080007EB     \\coverage\coverage \183--183      never    0.000%                        0.      0.       0.      0.       4.      0.
 SR:080007F4--080007F7     \\coverage\coverage \184--187         ok  100.000%                        0.      0.       0.      0.       4.      4.
 SR:080007FC--080007FF     \\coverage\coverage \188--188         ok  100.000%                        0.      0.       0.      0.       4.      4.
 SR:08000804--08000807     \\coverage\coverage \189--189         ok  100.000%                        0.      0.       0.      0.       4.      4.
 SR:0800080C--0800080F     \\coverage\coverage \190--190         ok  100.000%                        0.      0.       0.      0.       4.      4.
 SR:08000814--08000817     \\coverage\coverage \191--191         ok  100.000%                        0.      0.       0.      0.       4.      4.
  P:08000798--08000823  ⊟ TestSwitchCase                        call  100.000%                        0.      0.       1.      1.       4.      4.
 SR:08000FB8--08000FBB     \\coverage\coverage \663--663         ok  100.000%                        0.      0.       0.      0.       4.      4.
  P:08000824--080008AF  ⊟ MultiLine                             call  100.000%                        0.      0.       2.      2.       8.      8.
 SR:08000914--08000917     \\coverage\coverage \231--231         ok  100.000%                        0.      0.       0.      0.       4.      4.
 SR:08000934--08000937     \\coverage\coverage \236--236         ok  100.000%                        0.      0.       0.      0.       4.      4.
```

Double-clicking a line displays the function or call and detailed information about the code coverage in a **List** window.

Example 2:

```
COVerage.Option SourceMetric Call

…

sYmbol.Browse.Module

sYmbol.FILTER.ADD.sYmbol jd_modules \jdcolor \jdmarker \jdtrans

COVerage.ListCalleRs.preset jd_modules
```

Example 3:

```
sYmbol.Browse.SOURCE

COVerage.ListCalleRs.SOURCE \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdapistd.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdinput.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdpostct.c"
```

```
COVerage.ListCalleRs.SOURCE \"*jdc*.c"
```

Example 4:

```
sYmbol.Browse.Module

COVerage.ListCalleRs.sYmbol \jdapistd \jdmaster \jidctred
```

```
COVerage.ListCalleRs.sYmbol \jda*
```

See also

- COVerage.ListFunc.<sub_cmd>
- COVerage.ListCalleEs
- COVerage.ListModule
- COVerage.EXPORT.ListFunc
- COVerage.state
- COVerage.List
- COVerage.ListLine
- COVerage
- COVerage.EXPORT.ListFunc.<sub_cmd>

▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.ListFunc.<sub_cmd> Display coverage for HLL function

Format:	**COVerage.ListFunc.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [%*<format>*] [*<address>* | *<address_range>*] [**/***<option>*] **preset** [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [%*<format>*] [*<source>*...] [**/***<option>*] **sYmbol** [%*<format>*] [*<symbol>*...] [**/***<option>*]
<format>:	**SINGLE** | **MULTI** | **DO178** | **OBC**
<option>:	**SOrder** | **TOrder**

Displays the results of the code coverage analysis related to HLL functions based on the selected metric (see **COVerage.Option SourceMetric**).

The following <sub_cmd> are possible:

ADDRESS **COVerage.ListFunc** (deprecated)	Allows to restrict the displayed functions to a specified address range.
preset **COVerage.ListFunc** (deprecated)	If the command contains no parameters, then all HLL functions are displayed (see example 1). The *<filter>* parameter allows to reduce the number of functions to that which is in the focus of the code coverage analysis (see example 2).
SOURCE	Allows to restrict the displayed functions to the specified source files. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window (see example 3).

sYmbol	Allows to restrict the displayed functions to the specified symbol ranges. The symbol names are oriented towards the **symbol** column in the **sYmbol.Browse.Function** or **sYmbol.Browse.Module** window (see example 4).
<symbol>, <source>	You can use one or more items as filter criteria. The wildcards '`*`' and '`?`' are supported. Only items matching the filter criteria are displayed. See commands **sYmbol.FILTER.ADD.sYmbol** and **sYmbol.FILTER.ADD.SOURCE**.

Format Parameters SINGLE, MULTI, DO178, OBC

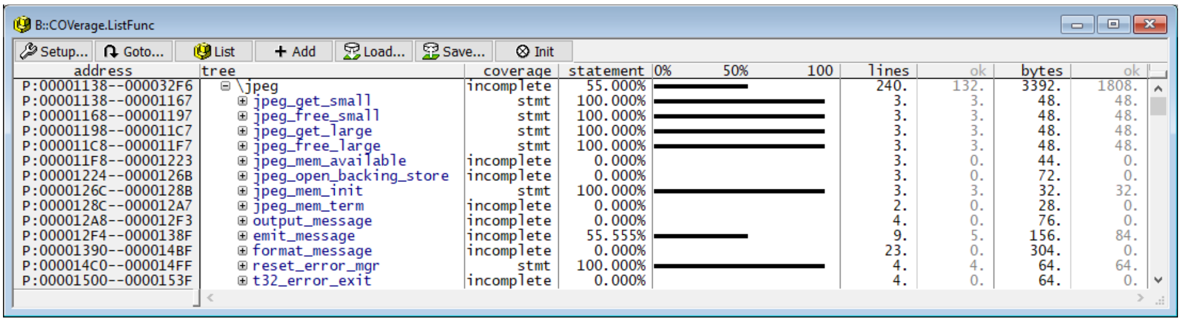
SINGLE	The code coverage results are displayed only for the selected metric.
MULTI	The code coverage results are displayed for the selected metric and all included metrics. E.g. the MCDC metric includes also CONDition, Decision and Statement coverage.
DO178	The code coverage results are displayed for the selected metric and all included metrics that are relevant for DO178. E.g. the MCDC metric includes also Decision and Statement coverage.
OBC	Includes object code branch coverage results if **COVerage.Option.SourceMetric** **Statement** is set.

Option SOrder, TOrder

SOrder	Display the source code lines belonging to a function in **source order**.
TOrder	Display the source code lines belonging to a function in **target order** (default).

Example 1:

```
COVerage.Option SourceMetric Statement

…

COVerage.ListFunc
```

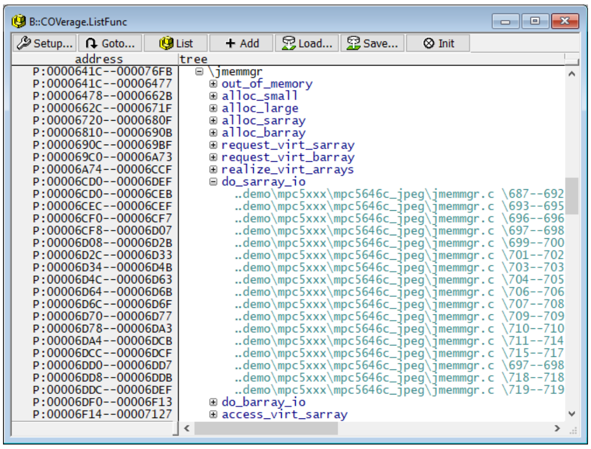


```
B::COVerage.ListFunc
Setup...  Goto...  List  Add  Load...  Save...  Init
          address         tree                      coverage     statement 0%   50%   100    lines    ok   bytes     ok
P:00001138--000032F6      \jpeg                     incomplete    55.000%                   240.   132.  3392.   1808.
P:00001138--00001167        jpeg_get_small                stmt   100.000%                     3.     3.    48.     48.
P:00001168--00001197        jpeg_free_small               stmt   100.000%                     3.     3.    48.     48.
P:00001198--000011C7        jpeg_get_large                stmt   100.000%                     3.     3.    48.     48.
P:000011C8--000011F7        jpeg_free_large               stmt   100.000%                     3.     3.    48.     48.
P:000011F8--00001223        jpeg_mem_available      incomplete     0.000%                     3.     0.    44.      0.
P:00001224--0000126B        jpeg_open_backing_store incomplete     0.000%                     3.     0.    72.      0.
P:0000126C--0000128B        jpeg_mem_init                 stmt   100.000%                     3.     3.    32.     32.
P:0000128C--000012A7        jpeg_mem_term           incomplete     0.000%                     2.     0.    28.      0.
P:000012A8--000012F3        output_message          incomplete     0.000%                     4.     0.    76.      0.
P:000012F4--0000138F        emit_message            incomplete    55.555%                     9.     5.   156.     84.
P:00001390--000014BF        format_message          incomplete     0.000%                    23.     0.   304.      0.
P:000014C0--000014FF        reset_error_mgr               stmt   100.000%                     4.     4.    64.     64.
P:00001500--0000153F        t32_error_exit          incomplete     0.000%                     4.     0.    64.      0.
```

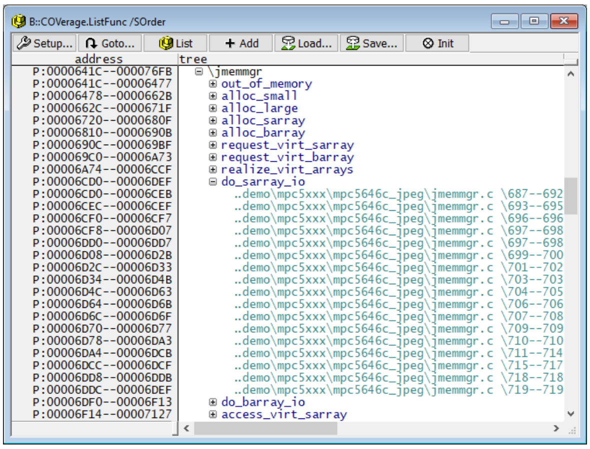
Double-clicking a line displays the function or call and detailed information about the code coverage in a **List** window.

Example 2:

```
COVerage.Option SourceMetric Statement

…

sYmbol.Browse.Module

sYmbol.FILTER.ADD.sYmbol jd_modules \jdcolor \jdmarker \jdtrans

COVerage.ListFunc.preset jd_modules
```



```
B::COVerage.ListFunc
Setup...  Goto...  List  Add  Load...  Save...  Init
          address         tree
P:0000641C--000076FB      \jmemmgr
P:0000641C--00006477        out_of_memory
P:00006478--0000662B        alloc_small
P:0000662C--0000671F        alloc_large
P:00006720--0000680F        alloc_sarray
P:00006810--0000690B        alloc_barray
P:0000690C--000069BF        request_virt_sarray
P:000069C0--00006A73        request_virt_barray
P:00006A74--00006CCF        realize_virt_arrays
P:00006CD0--00006DEF        do_sarray_io
P:00006CD0--00006CEB          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \687--692
P:00006CEC--00006CEF          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \693--695
P:00006CF0--00006CF7          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \696--696
P:00006CF8--00006D07          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \697--698
P:00006D08--00006D2B          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \699--700
P:00006D2C--00006D33          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \701--702
P:00006D34--00006D4B          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \703--703
P:00006D4C--00006D63          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \704--705
P:00006D64--00006D6B          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \706--706
P:00006D6C--00006D6F          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \707--708
P:00006D70--00006D77          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \709--709
P:00006D78--00006DA3          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \710--710
P:00006DA4--00006DCB          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \711--714
P:00006DCC--00006DCF          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \715--717
P:00006DD0--00006DD7          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \697--698
P:00006DD8--00006DDB          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \718--718
P:00006DDC--00006DEF          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \719--719
P:00006DF0--00006F13        do_barray_io
P:00006F14--00007127        access_virt_sarray
```



```
B::COVerage.ListFunc /SOrder
Setup...  Goto...  List  Add  Load...  Save...  Init
          address         tree
P:0000641C--000076FB      \jmemmgr
P:0000641C--00006477        out_of_memory
P:00006478--0000662B        alloc_small
P:0000662C--0000671F        alloc_large
P:00006720--0000680F        alloc_sarray
P:00006810--0000690B        alloc_barray
P:0000690C--000069BF        request_virt_sarray
P:000069C0--00006A73        request_virt_barray
P:00006A74--00006CCF        realize_virt_arrays
P:00006CD0--00006DEF        do_sarray_io
P:00006CD0--00006CEB          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \687--692
P:00006CEC--00006CEF          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \693--695
P:00006CF0--00006CF7          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \696--696
P:00006CF8--00006D07          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \697--698
P:00006DD0--00006DD7          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \697--698
P:00006D08--00006D2B          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \699--700
P:00006D2C--00006D33          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \701--702
P:00006D34--00006D4B          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \703--703
P:00006D4C--00006D63          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \704--705
P:00006D64--00006D6B          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \706--706
P:00006D6C--00006D6F          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \707--708
P:00006D70--00006D77          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \709--709
P:00006D78--00006DA3          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \710--710
P:00006DA4--00006DCB          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \711--714
P:00006DCC--00006DCF          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \715--717
P:00006DD8--00006DDB          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \718--718
P:00006DDC--00006DEF          ..demo\mpc5xxx\mpc5646c_jpeg\jmemmgr.c \719--719
P:00006DF0--00006F13        do_barray_io
P:00006F14--00007127        access_virt_sarray
```

Example 3:

```
sYmbol.Browse.SOURCE

COVerage.ListFunc.SOURCE \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdapistd.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdinput.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdpostct.c"
```

```
COVerage.ListFunc.SOURCE \"*jdc*.c"
```

Example 4:

```
sYmbol.Browse.Module

COVerage.ListFunc.sYmbol \jdapistd \jdmaster \jidctred
```

```
COVerage.ListFunc.sYmbol \jda*
```

See also

- COVerage.ListFunc

See also

- ■ COVerage.ListCalleEs
- ■ COVerage.EXPORT.ListInlineBlock.<sub_cmd>
- ■ COVerage
- ■ COVerage.state
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.ListInlineBlock.<sub_cmd> Display coverage for inlined block

Format:	**COVerage.ListInlineBlock.**<*sub_cmd*>
<*sub_cmd*>:	**ADDRESS** [%<*format*>] [<*address*> | <*address_range*>] [**/**<*option*>] **preset** [%<*format*>] [<*filter*>] [**/**<*option*>] **SOURCE** [%<*format*>] [<*source*>...] [**/**<*option*>] **sYmbol** [%<*format*>] [<*symbol*>…] [**/**<*option*>]
<*format*>:	**SINGLE** | **MULTI** | **DO178**
<*option*>:	**SOrder** | **TOrder**

Displays the result of the code coverage analysis related to inlined code blocks based on the selected metric (see**COVerage.Option SourceMetric**). The command **sYmbol.List.InlineBlock** provides a list of all inlined code blocks.

The following <sub_cmd> are possible:

ADDRESS	Allows to restrict the displayed blocks to a specified address range.
preset	When compiling with optimization the compiler may insert functions or parts of a function directly instead of adding a call to the function. This command lists all parts of the code where function parts have been inlined by the compiler and displays the code coverage result for the individual blocks. If the command contains no parameters, then all inline blocks are displayed (see example1). The commands **sYmbol.FILTER.ADD.SOURCE** and **sYmbol.FILTER.ADD.sYmbol** allow to combine source files/symbols of interest under a <*filter*>. The <*filter*> parameter allows to reduce the number of inlined blocks to that which is in the focus of the code coverage analysis. This is especially useful for very large projects (see example 2).

SOURCE	Allows to restrict the displayed inlined blocks to the specified source files. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window (see example 3).
sYmbol	Allows to restrict the displayed inlined blocks to the specified symbol ranges. The symbol names are oriented towards the **symbol** column in the **sYmbol.Browse.Function** or **sYmbol.Browse.Module** window (see example 4).
<symbol>, <source>	Instead of listing the sources individually, they can also be combined under a filter name. See commands **sYmbol.FILTER.ADD.sYmbol** and **sYmbol.FILTER.ADD.SOURCE**. The wildcards '*****' and '**?**' are supported.

Format Parameters SINGLE, MULTI, DO178

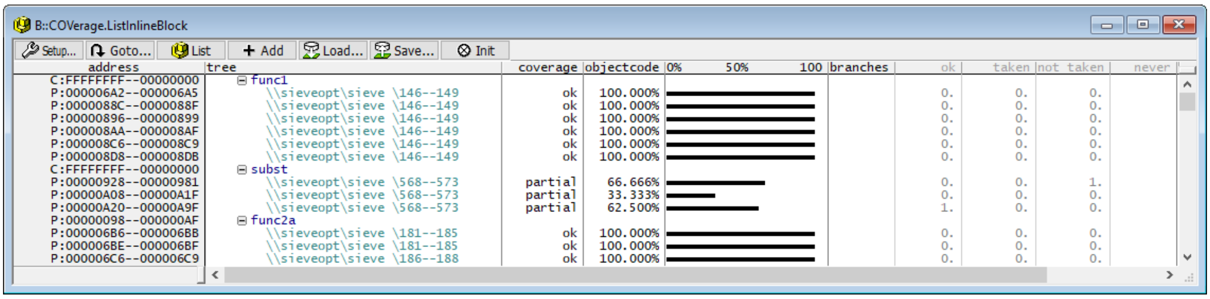
SINGLE	The code coverage results are displayed only for the selected metric.
MULTI	The code coverage results are displayed for the selected metric and all included metrics. E.g. the MCDC metric includes also CONDition, Decision and Statement coverage.
DO178	The code coverage results are displayed for the selected metric and all included metrics that are relevant for DO178. E.g. the MCDC metric includes also Decision and Statement coverage.

Option SOrder, TOrder

SOrder	Display the source code lines belonging to a function in **source order**.
TOrder	Display the source code lines belonging to a function in **target order** (default).

Example 1:

```
COVerage.Option SourceMetric Statement

...

COVerage.ListInlineBlock
```



```
              address  tree                                coverage objectcode 0%       50%      100 branches   ok   taken  not taken   never
 C:FFFFFFFF--00000000      ⊟ func1
 P:000006A2--000006A5         \\sieveopt\sieve \146--149        ok   100.000%                                 0.      0.        0.
 P:0000088C--0000088F         \\sieveopt\sieve \146--149        ok   100.000%                                 0.      0.        0.
 P:00000896--00000899         \\sieveopt\sieve \146--149        ok   100.000%                                 0.      0.        0.
 P:000008AA--000008AF         \\sieveopt\sieve \146--149        ok   100.000%                                 0.      0.        0.
 P:000008C6--000008C9         \\sieveopt\sieve \146--149        ok   100.000%                                 0.      0.        0.
 P:000008D8--000008DB         \\sieveopt\sieve \146--149        ok   100.000%                                 0.      0.        0.
 C:FFFFFFFF--00000000      ⊟ subst
 P:00000928--00000981         \\sieveopt\sieve \568--573   partial    66.666%                                 0.      0.        1.
 P:00000A08--00000A1F         \\sieveopt\sieve \568--573   partial    33.333%                                 0.      0.        0.
 P:00000A20--00000A9F         \\sieveopt\sieve \568--573   partial    62.500%                                 1.      0.        0.
 P:00000098--000000AF      ⊟ func2a
 P:000006B6--000006BB         \\sieveopt\sieve \181--185        ok   100.000%                                 0.      0.        0.
 P:000006BE--000006BF         \\sieveopt\sieve \181--185        ok   100.000%                                 0.      0.        0.
 P:000006C6--000006C9         \\sieveopt\sieve \186--188        ok   100.000%                                 0.      0.        0.
```

Double-clicking a line displays the block and detailed information about the code coverage in a **List** window.

Example 2:

```
COVerage.Option SourceMetric Statement

…

sYmbol.Browse.Module

sYmbol.FILTER.ADD.sYmbol jd_modules \jdcolor \jdmarker \jdtrans

COVerage.ListInlineBlock.preset jd_modules
```

Example 3:

```
sYmbol.Browse.SOURCE

COVerage.ListInlineBlock.SOURCE \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdapistd.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdinput.c" \
\"D:/work/demo/mpc5xxx/mpc5646c_jpeg/jdpostct.c"
```

```
COVerage.ListInlineBlock.SOURCE \"*jdc*.c"
```

Example 4:

```
sYmbol.Browse.Module

COVerage.ListInlineBlock.sYmbol \jdapistd \jdmaster \jidctred
```

```
COVerage.ListInlineBlock.sYmbol \jda*
```

See also

- COVerage.ListLine.<sub_cmd>
- COVerage.ListCalleEs
- COVerage.ListModule
- COVerage.EXPORT.ListLine
- COVerage.state
- COVerage.List
- COVerage.ListFunc
- COVerage
- COVerage.EXPORT.ListLine.<sub_cmd>

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.ListLine.<sub_cmd> Display coverage for HLL lines

Format:	**COVerage.ListLine.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [%*<format>*] [*<address>* | *<address_range>*] [**/***<option>*] **preset** [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<source_file>...*] [**/***<option>*] **sYmbol** [%*<format>*] [*<symbol>*…] [**/***<option>*]
<format>:	**SINGLE** | **MULTI** | **DO178** | **OBC**
<option>:	**SOrder** | **TOrder**

Displays the result of the code coverage analysis related to HLL lines based on the selected metric (see**COVerage.Option SourceMetric**).

The following <sub_cmd> are possible:

ADDRESS **COVerage.ListLine** (deprecated)	Allows to restrict the displayed lines to a specified address range.
preset	If the command contains no arguments, then all HLL lines are displayed. If the *<filter>* argument is passed, then only items matching the filter criteria are displayed (see example 1).
SOURCE	Lists lines using <source_file> as filter criterion. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.

sYmbol	Defines a filter for the symbols of the HLL lines to view.
<symbol>, <source_file>	You can use one or more items as filter criteria. The wildcards '*' and '?' are supported. Only items matching the filter criteria are displayed.

Format Parameters SINGLE, MULTI, DO178, OBC

SINGLE	The code coverage results are displayed only for the selected metric.
MULTI	The code coverage results are displayed for the selected metric and all included metrics. E.g. the MCDC metric includes also CONDition, Decision and Statement coverage.
DO178	The code coverage results are displayed for the selected metric and all included metrics that are relevant for DO178. E.g. the MCDC metric includes also Decision and Statement coverage.
OBC	Includes object code branch coverage results if **COVerage.Option.SourceMetric Statement** is set.

Option SOrder, TOrder

SOrder	Display the source code lines belonging to a function in **source order**.
TOrder	Display the source code lines belonging to a function in **target order** (default).

Example 1:

```
COVerage.ADD                        ;Update the coverage database

COVerage.ListLine "*chario.c"       ;Display all items which contain the
                                    ;file chario.c
COVerage.ListLine main              ;Display coverage for function main
```

Example 2:

```
COVerage.ListLine.SOURCE "*sieve.c"
```

See also

■ COVerage.ListLine

See also

- ■ COVerage.ListModule.<sub_cmd>
- ■ COVerage.ListCalleEs
- ■ COVerage.ListLine
- ■ COVerage.EXPORT.ListModule
- ■ COVerage.state
- ■ COVerage.List
- ■ COVerage.ListFunc
- ■ COVerage
- ■ COVerage.EXPORT.ListModule.<sub_cmd>

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.ListModule.<sub_cmd> Display coverage for modules

Format:	**COVerage.ListModule.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [%*<format>*] [*<address>* | *<address_range>*] [**/***<option>*] **preset** [%*<format>*] [*<filter>*] [**/***<option>*] **SOURCE** [*<source_file>...*] [**/***<option>*] **sYmbol** [%*<format>*] [*<symbol>…*] [**/***<option>*]
<format>:	**SINGLE** | **MULTI** | **DO178** | **OBC**
<option>:	**SOrder** | **TOrder**

Displays the result of the code coverage analysis related to modules based on the selected metric (see**COVerage.Option SourceMetric**).

The following <sub_cmd> are possible:

ADDRESS **COVerage.ListMod-ule** (deprecated)	Allows to restrict the displayed modules to a specified address range.
preset	Displays the results of the coverage analysis related to modules. Double-clicking a line displays the function and detailed information about the coverage. If the command contains no arguments, then all modules are displayed. If *<filter>* argument is passed, then only items matching the filter criteria are displayed (see example 1).
SOURCE	Lists modules using <source_file> as filter criterion. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.

sYmbol	Defines a filter for the symbols of the modules to view.
<symbol>, *<source_file>*	You can use one or more items as filter criteria. The wildcards '*' and '**?**' are supported. Only items matching the filter criteria are displayed.

Format Parameters SINGLE, MULTI, DO178, OBC

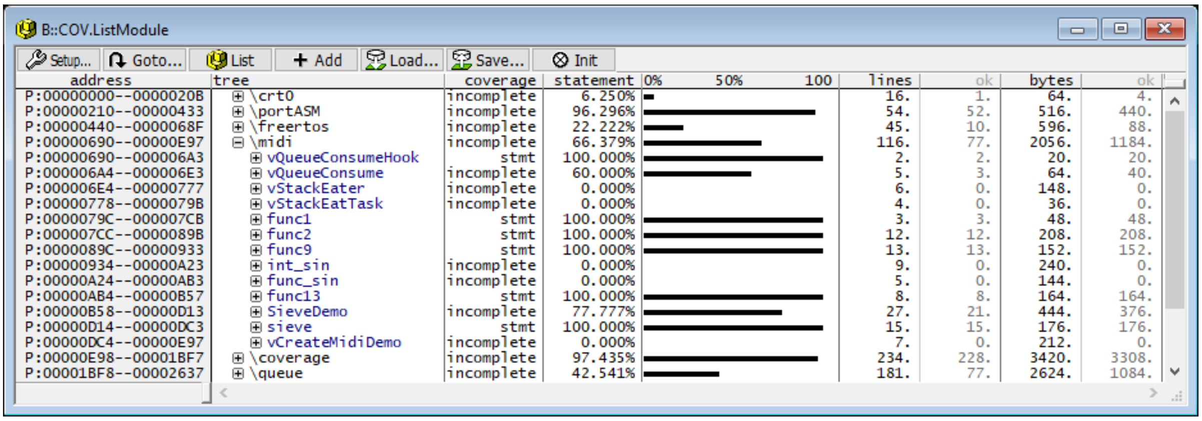
SINGLE	The code coverage results are displayed only for the selected metric.
MULTI	The code coverage results are displayed for the selected metric and all included metrics. E.g. the MCDC metric includes also CONDition, Decision and Statement coverage.
DO178	The code coverage results are displayed for the selected metric and all included metrics that are relevant for DO178. E.g. the MCDC metric includes also Decision and Statement coverage.
OBC	Includes object code branch coverage results if **COVerage.Option.SourceMetric Statement** is set.

Option SOrder, TOrder

SOrder	Display the source code lines belonging to a function in **source order**.
TOrder	Display the source code lines belonging to a function in **target order** (default).

Example 1:

```
COVerage.Option SourceMetric Statement

…

COVerage.ListModule
```

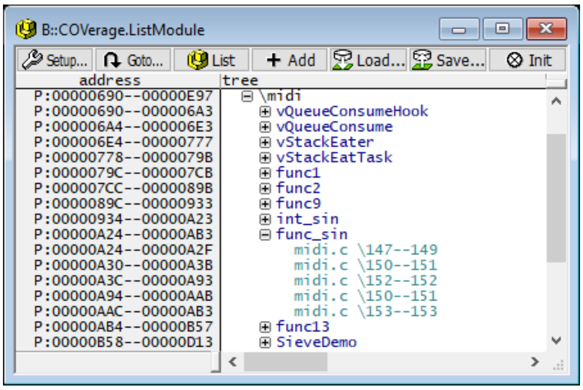


```
address                  tree                   coverage   statement 0%   50%   100  lines   ok  bytes    ok
P:00000000--0000020B  ⊞ \crt0                  incomplete    6.250%                  16.   1.    64.    4.
P:00000210--00000433  ⊞ \portASM               incomplete   96.296%                  54.  52.   516.  440.
P:00000440--0000068F  ⊞ \freertos              incomplete   22.222%                  45.  10.   596.   88.
P:00000690--00000E97  ⊟ \midi                  incomplete   66.379%                 116.  77.  2056. 1184.
P:00000690--000006A3    ⊞ vQueueConsumeHook          stmt  100.000%                   2.   2.    20.   20.
P:000006A4--000006E3    ⊞ vQueueConsume        incomplete   60.000%                   5.   3.    64.   40.
P:000006E4--00000777    ⊞ vStackEater          incomplete    0.000%                   6.   0.   148.    0.
P:00000778--0000079B    ⊞ vStackEatTask        incomplete    0.000%                   4.   0.    36.    0.
P:0000079C--000007CB    ⊞ func1                      stmt  100.000%                   3.   3.    48.   48.
P:000007CC--0000089B    ⊞ func2                      stmt  100.000%                  12.  12.   208.  208.
P:0000089C--00000933    ⊞ func9                      stmt  100.000%                  13.  13.   152.  152.
P:00000934--00000A23    ⊞ int_sin              incomplete    0.000%                   9.   0.   240.    0.
P:00000A24--00000AB3    ⊞ func_sin             incomplete    0.000%                   5.   0.   144.    0.
P:00000AB4--00000B57    ⊞ func13                     stmt  100.000%                   8.   8.   164.  164.
P:00000B58--00000D13    ⊞ SieveDemo            incomplete   77.777%                  27.  21.   444.  376.
P:00000D14--00000DC3    ⊞ sieve                      stmt  100.000%                  15.  15.   176.  176.
P:00000DC4--00000E97    ⊞ vCreateMidiDemo      incomplete    0.000%                   7.   0.   212.    0.
P:00000E98--00001BF7  ⊞ \coverage              incomplete   97.435%                 234. 228.  3420. 3308.
P:00001BF8--00002637  ⊞ \queue                 incomplete   42.541%                 181.  77.  2624. 1084.
```

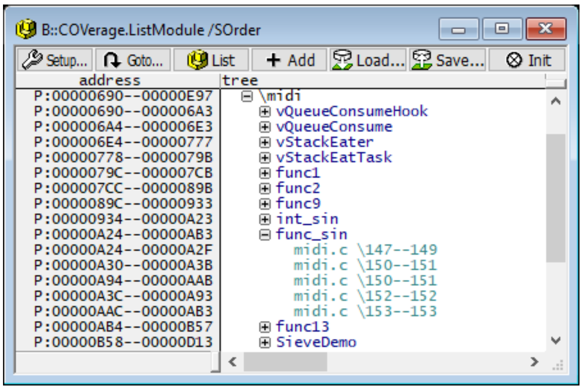
Double-clicking a line displays the module and detailed information about the code coverage in a **List** window.

Example 2:

```
COVerage.Option SourceMetric Statement

…

sYmbol.Browse.Module

sYmbol.FILTER.ADD.sYmbol jd_modules \crt0 \freertos \midi

COVerage.ListModule.preset jd_modules
```



```
address                  tree
P:00000690--00000E97  ⊟ \midi
P:00000690--000006A3    ⊞ vQueueConsumeHook
P:000006A4--000006E3    ⊞ vQueueConsume
P:000006E4--00000777    ⊞ vStackEater
P:00000778--0000079B    ⊞ vStackEatTask
P:0000079C--000007CB    ⊞ func1
P:000007CC--0000089B    ⊞ func2
P:0000089C--00000933    ⊞ func9
P:00000934--00000A23    ⊞ int_sin
P:00000A24--00000AB3    ⊟ func_sin
P:00000A24--00000A2F        midi.c \147--149
P:00000A30--00000A3B        midi.c \150--151
P:00000A3C--00000A93        midi.c \152--152
P:00000A94--00000AAB        midi.c \150--151
P:00000AAC--00000AB3        midi.c \153--153
P:00000AB4--00000B57    ⊞ func13
P:00000B58--00000D13    ⊞ SieveDemo
```


B::COVerage.ListModule /SOrder
Setup... Goto... List Add Load... Save... Init

```
address                  tree
P:00000690--00000E97  ⊟ \midi
P:00000690--000006A3    ⊞ vQueueConsumeHook
P:000006A4--000006E3    ⊞ vQueueConsume
P:000006E4--00000777    ⊞ vStackEater
P:00000778--0000079B    ⊞ vStackEatTask
P:0000079C--000007CB    ⊞ func1
P:000007CC--0000089B    ⊞ func2
P:0000089C--00000933    ⊞ func9
P:00000934--00000A23    ⊞ int_sin
P:00000A24--00000AB3    ⊟ func_sin
P:00000A24--00000A2F        midi.c \147--149
P:00000A30--00000A3B        midi.c \150--151
P:00000A94--00000AAB        midi.c \150--151
P:00000A3C--00000A93        midi.c \152--152
P:00000AAC--00000AB3        midi.c \153--153
P:00000AB4--00000B57    ⊞ func13
P:00000B58--00000D13    ⊞ SieveDemo
```

Example 3:

```
COVerage.ListModule.sYmbol \main
```

See also

■ COVerage.ListModule

COVerage.ListVar — Display coverage for variable

[Example]

See also

- ■ COVerage.ListVar.<sub_cmd>
- ■ COVerage.ListCalleEs
- ■ COVerage.EXPORT.ListVar
- ■ COVerage.state
- ■ COVerage.List
- ■ COVerage
- ■ COVerage.EXPORT.ListVar.<sub_cmd>

▲ 'Data Coverage' in 'Application Note for Trace-Based Code Coverage'

COVerage.ListVar.<sub_cmd> — Display coverage for variables

Format:	**COVerage.ListVar.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<address>* | *<address_range>*] **preset** [*<filter>*] **SOURCE** [*<source_file>*...] **sYmbol** [*<symbol>*…]

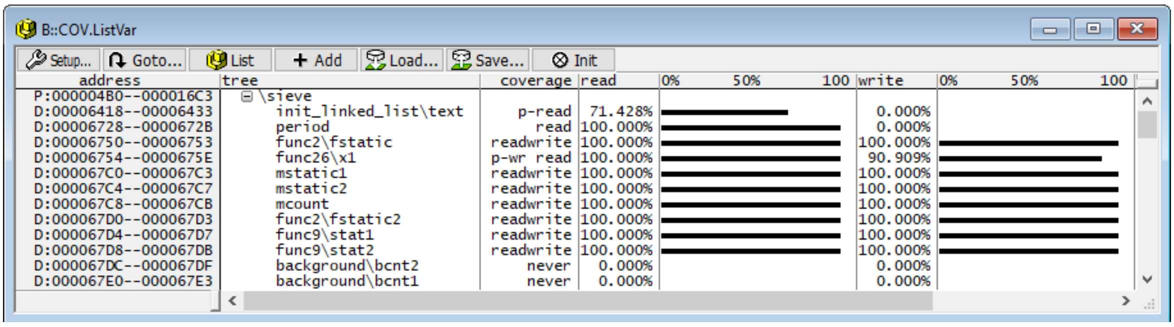
Displays the result of the data coverage analysis for source code variables if the source metric ObjectCode is set (**COVerage.Option SourceMetric ObjectCode**).

Since off-chip trace ports usually do not have enough bandwidth to make all read/write accesses (and the program flow) visible, they are rather unsuitable for data coverage. For test phases in which testing in the target environment is not yet required, a TRACE32 Instruction Set Simulator can be used well for data coverage.

!	If the program and data flow is broadcast via an offchip trace port (e.g. ARM-ETM or NEXUS), **COVerage.ListVar** displays an accurate result only if the trace does not contain **FIFOFULLs**.

The following *<sub_cmd>* are possible:

ADDRESS **COVerage.ListVar** (deprecated)	Allows to restrict the displayed variables to a specified address range.
preset	If the command contains no arguments, then all variables are displayed. If *<filter>* argument is passed, then only items matching the filter criteria are displayed (see examples).
SOURCE	Lists variables using <source_file> as filter criterion. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	List variable by using a module or program name as filter criterion.
<symbol>, *<source_file>*	You can use one or more items as filter criteria. The wildcards '*****' and '**?**' are supported. Only items matching the filter criteria are displayed.



```
        address          tree                          coverage  read     0%    50%    100  write    0%    50%    100
P:000004B0--000016C3     ⊟ \sieve
D:00006418--00006433        init_linked_list\text        p-read   71.428%                    0.000%
D:00006728--0000672B        period                         read  100.000%                    0.000%
D:00006750--00006753        func2\fstatic             readwrite  100.000%                  100.000%
D:00006754--0000675E        func26\x1                 p-wr read  100.000%                   90.909%
D:000067C0--000067C3        mstatic1                  readwrite  100.000%                  100.000%
D:000067C4--000067C7        mstatic2                  readwrite  100.000%                  100.000%
D:000067C8--000067CB        mcount                    readwrite  100.000%                  100.000%
D:000067D0--000067D3        func2\fstatic2            readwrite  100.000%                  100.000%
D:000067D4--000067D7        func9\stat1               readwrite  100.000%                  100.000%
D:000067D8--000067DB        func9\stat2               readwrite  100.000%                  100.000%
D:000067DC--000067DF        background\bcnt2              never    0.000%                    0.000%
D:000067E0--000067E3        background\bcnt1              never    0.000%                    0.000%
```

Examples:

```
Trace.FLOWPROCESS                ; Process the whole trace

Trace.Find FIFOFULL /ALL         ; Display the number of FIFOFULLs
PRINT %Decimal FOUND.COUNT()

COVerage.ADD                     ; Add the trace contents to the
                                 ; coverage system

COVerage.ListVar
```

A filter allows to limit the result to the variables of interest.

```
sYmbol.Filter.ADD.sYmbol vardiabc \diabc        ; create a filter that
                                                ; represents the module
                                                ; \diabc

COVerage.ADD                                    ; Add the trace contents
                                                ; to the coverage system

COVerage.ListVar vardiabc                       ; display data coverage
                                                ; only for the variables
                                                ; of the module \diabc
```

See also

■ COVerage.ListVar

COVerage.LOAD

Load coverage database from file

Format:	**COVerage.LOAD** *<file>* [**/***<option>*] *<trace>*.**COVerage.LOAD** (deprecated)
<option>:	**Replace** **ADD** **SUBtract**

Loads the code coverage information from a file. The currently available code coverage information is discarded.

<file>	Name of the file with a previously saved code coverage data set. The default extension of the file name is ***.acd**. The file extension *.acd can be omitted.
Replace (default)	Removes the current coverage information of TRACE32 and replaces it with the stored coverage data set of the file.
ADD	Keeps the current coverage information of TRACE32 and updates it with the stored coverage data set of the file.
SUBtract	Removes all coverage information of TRACE32 that is also present in the stored coverage data set of the file.

See also

■ COVerage.ListCalleEs ■ COVerage ■ COVerage.state

▲ 'Assemble Multiple Test Runs' in 'Application Note for Trace-Based Code Coverage'

Format:	**COVerage.MAP** *<source> <destination>* [**/***<option>*]
<option>:	**Replace** **ADD** **SUBtract**

Allows to summarize the coverage of a code section that is available several times in a program, e.g. a shared library that is used more the once.

Maps the code coverage of a source range to a destination range. Both ranges have to have the same length.

<source>	The address range whose code coverage is mapped to another one.
<destination>	The address range whose code coverage is updated.
Replace	Removes the current coverage information of the destination range and replaces it with the coverage data of the source range.
ADD	Keeps the current coverage information of the destination range, but updates it with the coverage data of the source range.
SUBtract	Removes all coverage information of the destination range that is also present in the coverage data set of the source range.

See also

■ COVerage ■ COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.METHOD

Format:	**COVerage.METHOD INCremental | SPY | RTS | ART | Hardware**

TRACE32 supports various code coverage methods. The code coverage method **INCremental** is supported for all processor architectures, as long as information about the executed instructions is recorded by a TRACE32 trace tool or by an onchip trace buffer. All other methods are subject to restrictions.

INCremental	INCRemental code coverage is based on the trace recording. After the trace recording stopped the command **COVerage.ADD** can be used to add the current trace recording to the code coverage database. Incremental code coverage is the preferred method for the **Trace.Modes** Fifo, Stack and Leash, but it can also be used in conjunction with the **Trace.Mode** STREAM.
SPY	SPY code coverage is based on the trace recording. It can only be selected if the **Trace.Mode STREAM** is active. While trace data is being recorded, streaming to the host is automatically interrupted at regular intervals in order to update the coverage database. SPY code coverage is only recommended if the processor/trace protocol in use is not supported by RTS. For setup details, refer to the chapter **“SPY Mode Code Coverage”** in Application Note for Trace-Based Code Coverage, page 27 (app_code_coverage.pdf). SPY code coverage is only possible for static code and is otherwise subject to the same restrictions as **Trace.Mode STREAM**.
RTS	RTS stands for Real-time Processing. The COVerage.METHOD RTS is automatically enabled if **RTS.ON**. Trace data are processed while recording and a live display of the code coverage results is possible. For details refer to the examples given in the description of the **RTS** command group. RTS code coverage is subject to the same restrictions as the **RTS** command group.
ART	ART code coverage is based on the assembler single steps recorded to the TRACE32 Advanced Register Trace **ART**. The code coverage database is updated after every single step. ART code coverage is only supported for a limited number of processor architectures. If you processor architecture is not supported, the ART method will be grayed out in the **COVerage** window and the **COVerage.METHOD ART** command will return a “command locked” error. Please contact in this case the Lauterbach technical support.

See also

■ COVerage	■ COVerage.state	■ Analyzer.Mode	■ ART
■ RTS.ON			

▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

COVerage.Mode — Activate code coverage for virtual targets

Format:	**COVerage.Mode** *<mode>*
<mode>:	**FastCOVerage** [**ON** | **OFF**]

Activates code coverage for virtual targets with minimal trace activation.

FastCOVerage	Code coverage via the MCD interface. TRACE32 instructs a virtual target via the MCD interface to perform a code coverage analysis. Upon completion of the coverage analysis, the coverage information is imported to the TRACE32 coverage database with the **COVerage.ADD** command. Prerequisite: **COVerage.METHOD.INCremental** is selected in the **COVerage.state** window.

See also

■ COVerage	■ COVerage.state

▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'

COVerage.OFF — Deactivate coverage

Format:	**COVerage.OFF**

Coverage data will not be recorded.

See also

■ COVerage	■ COVerage.state

▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'

Format:	**COVerage.ON**

Activates the currently selected **COVerage.METHOD**.

See also

- ■ COVerage ■ COVerage.state
- ▲ 'Trace Data Collection Modes' in 'Application Note for Trace-Based Code Coverage'

COVerage.Option — Set coverage options

Using the **COVerage.Option** command group, you can configure how TRACE32 processes or displays code coverage data.

See also

- ■ COVerage.Option.BLOCKMode
- ■ COVerage.Option.SourceMetric
- ■ COVerage
- ■ COVerage.Option.ITrace
- ■ COVerage.Option.StaticInfo
- ■ COVerage.state
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.Option.BLOCKMode — Enable/disable line block mode

Format:	**COVerage.Option.BLOCKMode** [**ON** | **OFF**]

Changes how code coverage measurements are applied to source code lines.

ON	The code coverage result is applied to all associated source code lines.
OFF	The code coverage result is applied only to the last source code line.

Example: Please refer to **COVerage.EXPORT.CBA**.

See also

- ■ COVerage.Option
- ■ COVerage.EXPORT.CBA
- ▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.Option.ITrace — Enable instruction trace processing

Format:	**COVerage.Option.ITrace** [**ON** | **OFF**]

TRACE32 does not record trace information about conditional instructions in the simulator. If a trace, which has been recorded on real hardware, should be loaded in the simulator, the additional info is processed.

ON	Conditional instruction trace is processed.
OFF	Only the simulator bus trace is processed.

See also

■ COVerage.Option

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.Option.SourceMetric — Select code coverage metric

Format:	**COVerage.Option.SourceMetric** *<criterion>*
<criterion>:	**Call** **CONDition** **Decision** **Function** **MCDC** **ObjectCode** **Statement**

Code coverage for the selected metric is performed based on the trace data.

ObjectCode	ObjectCode coverage is performed. See **"Object Code Coverage"** in Application Note for Trace-Based Code Coverage, page 40 (app_code_coverage.pdf).
Statement	Indicates if an HLL line has achieved the code coverage criterion *statement coverage*. See **"Statement Coverage"** in Application Note for Trace-Based Code Coverage, page 47 (app_code_coverage.pdf).
Decision	Indicates if an HLL line has achieved the code coverage criterion *decision coverage*. See **"Full Decision Coverage"** in Application Note for Trace-Based Code Coverage, page 51 (app_code_coverage.pdf).

MCDC	Modified condition/decision coverage (MC/DC). Indicates if an HLL line has achieved the code coverage criterion *modified condition/decision coverage*. See **“Modified Condition/Decision Coverage (MC/DC)”** in Application Note for Trace-Based Code Coverage, page 72 (app_code_coverage.pdf).
Function	Indicates if an HLL line has achieved the code coverage criterion *function coverage*. See **“Function Coverage”** in Application Note for Trace-Based Code Coverage, page 79 (app_code_coverage.pdf).
Call	Indicates if an HLL line has achieved the code coverage criterion *call coverage*. See **“Call Coverage”** in Application Note for Trace-Based Code Coverage, page 84 (app_code_coverage.pdf).

For selections other than **ObjectCode** the available debug symbol information is used to improve the accuracy of the code coverage. E.g. literal pools and alignment padding blocks are excluded from the code coverage calculations.

Blocks of assembly instructions are not affected by this option.

For more information about all the metrics, please refer to the chapter **“Supported Code Coverage Metrics”** in Application Note for Trace-Based Code Coverage, page 38 (app_code_coverage.pdf).

See also

■ COVerage.Option ❏ COVerage.SourceMetric()

▲ ’Supported Code Coverage Metrics’ in ’Application Note for Trace-Based Code Coverage’
▲ ’Release Information’ in ’Legacy Release History’

COVerage.Option.StaticInfo

Perform code coverage precalculations

Format:	**COVerage.Option.StaticInfo** [**ON** | **OFF**]

Performs the following precalculations for the code coverage if **ON**:

- **Object code coverage**
  - IT block preprocessing to improve the coverage results for ARM Thumb code.
  - Counting the conditional branches for the conditional branch analysis.
- **Statement and decision coverage**
  - Detection of literal pools and alignment padding blocks.
  - Counting the instructions for modules/functions without source code information.

ON (default)	Perform precalculations.
OFF	Do not perform precalculations (recommended in the case of issues with the code coverage).

See also

■ COVerage.Option

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.RESet — Clear coverage database

Format:	**COVerage.RESet**
	<trace>.**COVerage.RESet** (deprecated)

Discards the complete code coverage information and restores the default code coverage settings.

See also

■ COVerage ■ COVerage.state

▲ ’Trace Data Collection Modes’ in ’Application Note for Trace-Based Code Coverage’

COVerage.SAVE — Save coverage database to file

Format:	**COVerage.SAVE** *<file>*
	<trace>.**COVerage.SAVE** (deprecated)

Saves the code coverage information to a file.

<file>	The default extension of the file name is ***.acd**.

See also

■ COVerage ■ COVerage.state

▲ ’Assemble Multiple Test Runs’ in ’Application Note for Trace-Based Code Coverage’

Format:	**COVerage.Set** [*<address>* | *<range>*] *<state>* *<trace>*.**COVerage.Set** (deprecated)
<state>	**NOTTAKEN** **TAKEN** **NOTEXEC** **ONLYEXEC** **OK**

Marks the defined range with the specified execution state. If the instruction is already marked with an execution state the new state is added incrementally.

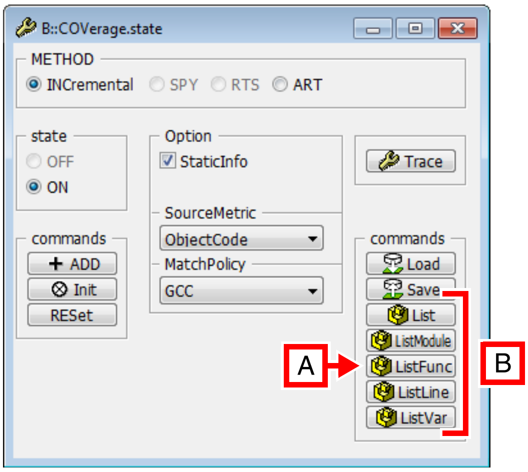
See also

■ COVerage ■ COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

Format:	**COVerage.state**

Opens the **COVerage.state** window, where you can configure the code coverage analysis and display the results.



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

B Click to display the results of the code coverage analysis.

See also

■ COVerage	■ COVerage.ACCESS	■ COVerage.ADD	■ COVerage.Delete
■ COVerage.EXPORT	■ COVerage.Init	■ COVerage.List	■ COVerage.ListCalleEs
■ COVerage.ListCalleRs	■ COVerage.ListFunc	■ COVerage.ListInlineBlock	■ COVerage.ListLine
■ COVerage.ListModule	■ COVerage.ListVar	■ COVerage.LOAD	■ COVerage.MAP
■ COVerage.METHOD	■ COVerage.Mode	■ COVerage.OFF	■ COVerage.ON
■ COVerage.Option	■ COVerage.RESet	■ COVerage.SAVE	■ COVerage.Set
■ COVerage.StaticInfo	■ COVerage.TreeWalkSETUP	■ RTS.OFF	

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'
▲ 'Release Information' in 'Legacy Release History'

Format:	**COVerage.StaticInfo** (deprecated) **Use sYmbol.ECA.BINary.PROCESS instead**

See also

■ COVerage ■ COVerage.state

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

See also

- ■ COVerage.TreeWalkSETUP.<sub_cmd>
- ■ COVerage.state
- ■ COVerage
- ❏ COVerage.TreeWalk()

▲ 'Introduction' in 'Application Note for Trace-Based Code Coverage'

COVerage.TreeWalkSETUP.<sub_cmd> Prepare a coverage symbol tree

Format:	**COVerage.TreeWalkSETUP.***<sub_cmd>*
<sub_cmd>:	**ADDRESS** [*<address>* | *<address_range>*] **preset** *[<filter>] [*/<option>] **SOURCE** [*<source_file>*] **sYmbol** [*<symbol>*…]

Prepares a tree with modules, functions, and HLL lines. The tree can be traversed with the PRACTICE function **COVerage.TreeWalk()**.

ADDRESS	Defines a filter for the addresses you want to include in the tree.
preset	If the command contains no parameters, then all symbols are included in the tree. The <filter> parameter allows to reduce the number of symbols.
SOURCE	Defines a filter for the source files you want to include in the tree. The syntax of the pathname is oriented towards the **symbol** and **path** columns in the **sYmbol.Browse.SOURCE** window.
sYmbol	Defines a filter for the symbols you want to include in the tree.
<symbol>, <source>	You can use one or more items as filter criteria. The wildcards '*****' and '**?**' are supported. Only items matching the filter criteria are displayed.

Example:

```
PRIVATE &node

; create a tree with all symbols starting with “func”
COVerage.TreeWalkSETUP.sYmbol func*

&node=COVerage.TreeWalk("Init")              ; get the first tree element
WHILE "&node"!=""
(
   IF STRing.SCAN("&node","\",0.)==0.        ; element is a module
   (
      PRINT "The next module is: &node"
   )
   ELSE IF STRing.SCAN("&node","--",0.)>-1. ; element is an HLL line
   (
      PRINT "The next HLL line is: &node"
   )
   ELSE                                      ; element is a function
   (
      PRINT "The next function is: &node"
   )
   &node=COVerage.TreeWalk("Recurse")        ; get the next tree element
)
```

See also

■ COVerage.TreeWalkSETUP

CTS — Context tracking system (CTS)

CTS (**C**ontext **T**racking **S**ystem) is a technique that allows the context of the target system to be reconstructed for each single record sampled to the trace buffer. Context of the target system means here the contents of the CPU registers, the memories, the caches and TLBs (for selected architectures only).

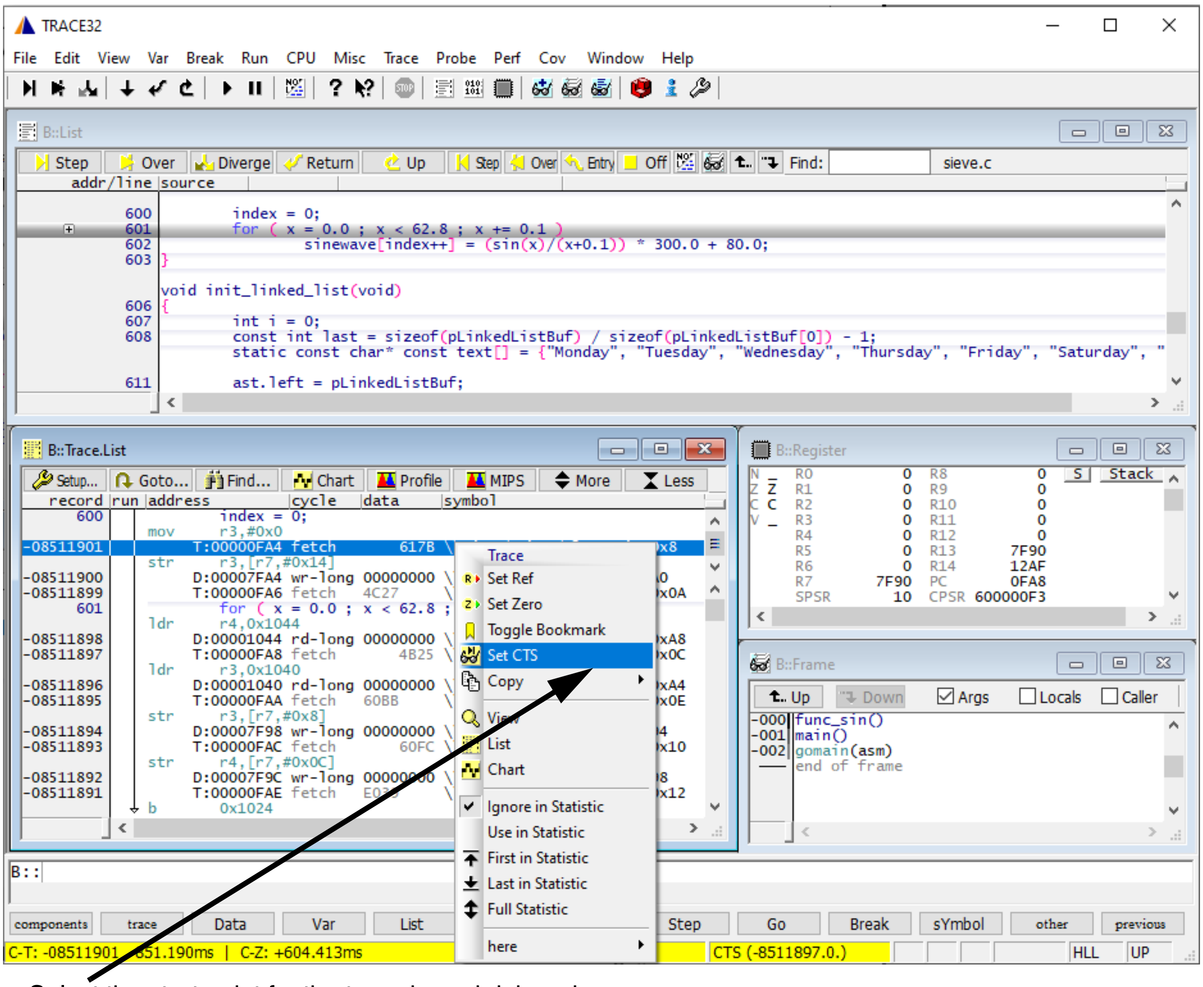
See also

■ CTS.CACHE	■ CTS.CAPTURE	■ CTS.Chart.sYmbol	■ CTS.EXPORT
■ CTS.FixedControl	■ CTS.GOTO	■ CTS.INCremental	■ CTS.Init
■ CTS.List	■ CTS.ListNesting	■ CTS.Mode	■ CTS.OFF
■ CTS.ON	■ CTS.PROCESS	■ CTS.PROfileChart	■ CTS.RESet
■ CTS.SELectiveTrace	■ CTS.SKIP	■ CTS.SmartTrace	■ CTS.state
■ CTS.STATistic	■ CTS.TAKEOVER	■ CTS.UNDO	■ CTS.UseCACHE
■ CTS.UseConst	■ CTS.UseMemory	■ CTS.UseReadCycle	■ CTS.UseRegister
■ CTS.UseSIM	■ CTS.UseVM	■ CTS.UseWriteCycle	■ Go.Back

▲ 'Release Information' in 'Legacy Release History'

Trace-based Debugging

The main application for **CTS** is the so-called trace-based debugging. Trace-based debugging allows to re-run the program and data flow sampled to the trace buffer on the TRACE32 screen. Precondition to perform a full-featured trace-based debugging is that the complete program and data flow until the stop of the program execution is sampled to the trace buffer. Otherwise CTS has to be configured to give correct results (See **CTS.state**).



TRACE32
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov Window Help
B::List
Step Over Diverge Return Up Step Over Entry Off Find: sieve.c
B::Trace.List
Setup... Goto... Find... Chart Profile MIPS More Less
Trace
Set Ref
Set Zero
Toggle Bookmark
Set CTS
Copy
View
List
Chart
Ignore in Statistic
Use in Statistic
First in Statistic
Last in Statistic
Full Statistic
here
B::Register
B::Frame
Up Down Args Locals Caller
components trace Data Var List Step Go Break sYmbol other previous
HLL UP

```
addr/line source
      600         index = 0;
      601         for ( x = 0.0 ; x < 62.8 ; x += 0.1 )
      602                 sinewave[index++] = (sin(x)/(x+0.1)) * 300.0 + 80.0;
      603 }

          void init_linked_list(void)
      606 {
      607         int i = 0;
      608         const int last = sizeof(pLinkedListBuf) / sizeof(pLinkedListBuf[0]) - 1;
                  static const char* const text[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "

      611         ast.left = pLinkedListBuf;
```

```
   record run address            cycle     data       symbol
      600               index = 0;
             mov        r3,#0x0
-08511901        T:00000FA4 fetch         617B \
             str        r3,[r7,#0x14]
-08511900        D:00007FA4 wr-long 00000000 \
-08511899        T:00000FA6 fetch   4C27     \
      601               for ( x = 0.0 ; x < 62.8 ;
             ldr        r4,0x1044
-08511898        D:00001044 rd-long 00000000 \
-08511897        T:00000FA8 fetch         4B25 \
             ldr        r3,0x1040
-08511896        D:00001040 rd-long 00000000 \
-08511895        T:00000FAA fetch   60BB     \
             str        r3,[r7,#0x8]
-08511894        D:00007F98 wr-long 00000000 \
-08511893        T:00000FAC fetch         60FC \
             str        r4,[r7,#0x0C]
-08511892        D:00007F9C wr-long 00000000 \
-08511891        T:00000FAE fetch   E03      \
             b          0x1024
```

```
N _   R0        0   R8        0
Z Z   R1        0   R9        0
C C   R2        0   R10       0
V _   R3        0   R11       0
      R4        0   R12       0
      R5        0   R13    7F90
      R6        0   R14    12AF
      R7     7F90   PC     0FA8
      SPSR     10   CPSR 600000F3
```

```
-000 func_sin()
-001 main()
-002 gomain(asm)
     end of frame
```

```
B::
C-T: -08511901  851.190ms | C-Z: +604.413ms     CTS (-8511897.0.)
```

Select the start point for the trace-based debugging

After selecting the start point for the trace-based debugging TRACE32 does the following:

- The TRACE32 screen displays the context of the processor as it was when the selected start point was recorded to the trace buffer (e.g. CPU registers, source listing, variables etc.).
- The yellow CTS field in the state line indicates that the TRACE32 screen no longer displays the current state of the CPU.
- All run-time control buttons in the **List** window are yellow, to indicate that trace-based debugging is enabled.

If trace-based debugging in on, you can use all run-time control commands to re-run the information sampled to the trace buffer on the TRACE32 screen (e.g. **Step.single**, **Step.Back**, **Go.Return**, **Var.Step.Till** etc.).

Trace-based debugging can be switched off by either using the **Off** button in the **List** window or by entering **CTS.OFF** into the command line.

Full High-Level Language Trace Display

If the complete program and data flow until the stop of the program execution is sampled to the trace buffer TRACE32 can display a full High-Level Language trace containing also register and stack variables. See the command **CTS.List**.

Reconstruction of Trace Gaps (TRACE32-ICD)

CTS.List can also be used to reconstruct trace information:

- trace information lost through an overload of the trace port can be reconstructed in most cases.
- if only read cycles are sampled to prevent an overload of the trace port, CTS can reconstruct all write cycles.

CTS.CACHE

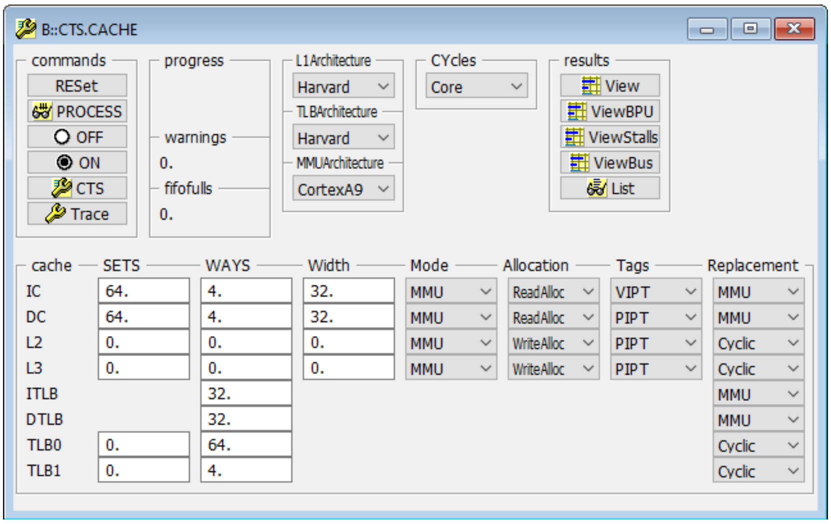
CTS cache analysis

Format:	**CTS.CACHE**

TRACE32 allows to perform a cache analysis using CTS technology, i.e. based on the program execution captured in a trace recording.

The cache analysis requires detailed knowledge of the structure of the CPU's cache. For most CPUs TRACE32 is aware of the cache structure.

To check if TRACE32 has the correct information for the cache structure of your CPU, open the **CTS.CACHE.state** window. To define the cache structure for TRACE32, use the TRACE32 command line or adjust the settings in the **CTS.CACHE.state** window.



progress

warnings 0.

fifofulls 0.

L1Architecture: Harvard

TLBArchitecture: Harvard

MMUArchitecture: CortexA9

CYcles: Core

cache	SETS	WAYS	Width	Mode	Allocation	Tags	Replacement
IC	64.	4.	32.	MMU	ReadAlloc	VIPT	MMU
DC	64.	4.	32.	MMU	ReadAlloc	PIPT	MMU
L2	0.	0.	0.	MMU	WriteAlloc	PIPT	Cyclic
L3	0.	0.	0.	MMU	WriteAlloc	PIPT	Cyclic
ITLB		32.					MMU
DTLB		32.					MMU
TLB0	0.	64.					Cyclic
TLB1	0.	4.					Cyclic

After **CTS** is switched to **ON** and **CTS.Mode CACHE** is selected, the contents of the caches and TLBs can be reconstructed. The cache analysis can be used for the following tasks:

- To support you to improve the cache hit rate by changing code and data locations
- To verify the cache hit rates after code changes
- To identify candidates for TCMs (tightly coupled memories) or faster memories
- To support you to find performance or bus bottlenecks
- To support you to improve the system performance and to reduce the power consumption
- To support you to try and verify different cache strategies
- To support you to identify optimum cache configuration and sizes for new silicons

The command group **CTS.CACHE** provides also the following advanced performance analysis features:

- Analysis of the branch prediction unit
- Analysis of the external bus interface
- Analysis of idle/stall operations

Even if these commands analyze different aspects of a microcontroller they are summarized here.

See also

■ CTS.CACHE.Allocation	■ CTS.CACHE.CYcles
■ CTS.CACHE.DefineBus	■ CTS.CACHE.L1Architecture
■ CTS.CACHE.LFSR	■ CTS.CACHE.ListAddress
■ CTS.CACHE.ListFunc	■ CTS.CACHE.ListLine
■ CTS.CACHE.ListModules	■ CTS.CACHE.ListRequests
■ CTS.CACHE.ListSet	■ CTS.CACHE.ListVar
■ CTS.CACHE.MMUArchitecture	■ CTS.CACHE.Mode
■ CTS.CACHE.Replacement	■ CTS.CACHE.RESet
■ CTS.CACHE.SETS	■ CTS.CACHE.Sort
■ CTS.CACHE.state	■ CTS.CACHE.Tags
■ CTS.CACHE.TLBArchitecture	■ CTS.CACHE.View
■ CTS.CACHE.ViewBPU	■ CTS.CACHE.ViewBus
■ CTS.CACHE.ViewStalls	■ CTS.CACHE.WAYS
■ CTS.CACHE.Width	■ CTS
■ CTS.PROfileChart.CACHE	■ CTS.state

▲ 'Release Information' in 'Legacy Release History'

Format:	**CTS.CACHE.Allocation** *<cache>* **ReadAlloc** | **WriteAlloc**
<cache>:	**IC** **DC** **L2** **L3**

The command **CTS.CACHE.Allocation** describes how the CPU deals with a cache miss on a data store/write access.

ReadAlloc	The data from a memory address is only loaded to the cache on read/load accesses.
WriteAlloc	The data from a memory address is loaded to the cache on a store/write access and the new data is written in the cache line. Please note that this also depends on the cache mode (write-through or copy-back).

```
CTS.CACHE.Allocation IC ReadAlloc     ; the instruction cache is a
                                      ; read allocate cache
```

See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.CYcles — Define counting method for cache analysis

Format:	**CTS.CACHE.CYcle Core** | **Bus** | **NonSequential**

Defines which method is used to count the cache hit/cache miss rate.

Core	The hit or miss counter is incremented on every core cycle.
Bus	The hit or miss counter is incremented on every bus cycle.
NonSequential	The hit or miss counter is only incremented if the CPU accesses a new cache line or performs a non-sequential access.

See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.DefineBus — Define bus interface

Format:	**CTS.CACHE.DefineBus** *<bus> <range> <bus_type> <frequency> <unknown> <read> <readreq> <readline> <write> <writeseq> <writeline> <writehalf>*
<bus>:	**BUS0**
	BUS1
	BUS2
	BUS3
<bus_type>:	**SIMPLE32**
	SIMPLE32I
	SIMPLE32D
	SIMPLE64
	SIMPLE64I
	SIMPLE64D

Defines the bus interface that is the base for the analysis of the bus utilization by the command **CTS.CACHE.ViewBus**.

SIMPLE indicates that the number of clock cycles required by each type of memory access can be directly given.

<range>	Memory range addressed by the bus. The physical address has to be specified (memory class A:)
<frequency>	Bus frequency.
<unknown>	Average number of clock cycles required by a memory access that is categorized as unknown by the cache analysis.
<read>	Number of clock cycles required by a memory read access.
<readseq>	Number of clock cycles required by a subsequent memory read access (e.g. burst access).
<readline>	Number of clock cycles required by a cache line fill.
<write>	Number of clock cycles required by a memory write access.
<writeseq>	Number of clock cycles required by a subsequent memory write access (e.g. burst access).
<writeline>	Number of clock cycles required to write the contents of a cache line back to memory (copy back).
<writehalf>	Number of clock cycles required to write the contents of half a cache line back to memory (copy back).

```
CTS.CACHE.DefineBus BUS0 A:0++0xffffffff SIMPLE64 100.MHz
              1. 1. 1. 4. 1. 1. 4. 2.

CTS.CACHE.DefineBus BUS1 A:0x80000000++0x1ffffff SIMPLE32 100.MHz
              5. 8. 1. 6. 7. 1. 8. 4.
```

See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.L1Architecture — Define architecture for L1 cache

Format:	**CTS.CACHE.L1Architecture Harvard** | **Unified** | **UnifiedSplit**

Defines the CACHE structure. This command defines the architecture of the level 1 cache.

Harvard	The L1 cache has Harvard architecture, which means that there is an instruction cache and a data cache available.
Unified	The L1 cache is a unified cache, which means that the same cache is used for instruction fetches and data loads/stores.
UnifiedSplit	The L1 cache is a unified cache, which means that the same cache is used for instruction fetches and data loads/stores. TRACE32 splits however the unified cache in an instruction and data cache for the cache analysis. The splitting is based on the cycle type (e.g. read, write, ptrace, exec).

See also

■ CTS.CACHE ■ CTS.CACHE.state

CTS.CACHE.LFSR — Linear-feedback shift register for random generator

Format:	**CTS.CACHE.LFSR IC** | **DC** | **L2** | **L3** *<lfsr>*

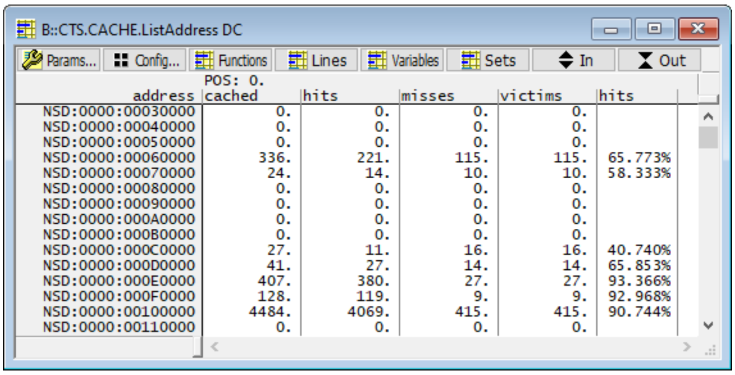
Set the start value of the linear-feedback shift register for random replacement strategy.

See also

■ CTS.CACHE ■ CTS.CACHE.state

Format:	**CTS.CACHE.ListAddress IC** | **DC** | **L2** | **L3** *<range>*

Performs a cache analysis based on addresses.



```
                  POS: 0.
         address  cached     hits        misses      victims     hits
NSD:0000:00030000       0.          0.          0.          0.
NSD:0000:00040000       0.          0.          0.          0.
NSD:0000:00050000       0.          0.          0.          0.
NSD:0000:00060000     336.        221.        115.        115.   65.773%
NSD:0000:00070000      24.         14.         10.         10.   58.333%
NSD:0000:00080000       0.          0.          0.          0.
NSD:0000:00090000       0.          0.          0.          0.
NSD:0000:000A0000       0.          0.          0.          0.
NSD:0000:000B0000       0.          0.          0.          0.
NSD:0000:000C0000      27.         11.         16.         16.   40.740%
NSD:0000:000D0000      41.         27.         14.         14.   65.853%
NSD:0000:000E0000     407.        380.         27.         27.   93.366%
NSD:0000:000F0000     128.        119.          9.          9.   92.968%
NSD:0000:00100000    4484.       4069.        415.        415.   90.744%
NSD:0000:00110000       0.          0.          0.          0.
```

cached	Number of accesses to cached memory.
hits	Number of cache hits. (percentage based on all cached accesses)
misses	Number of cache misses. (percentage based on all cached accesses)
victims	Number of cache lines that were thrown out of the cache after a cache miss occurred.

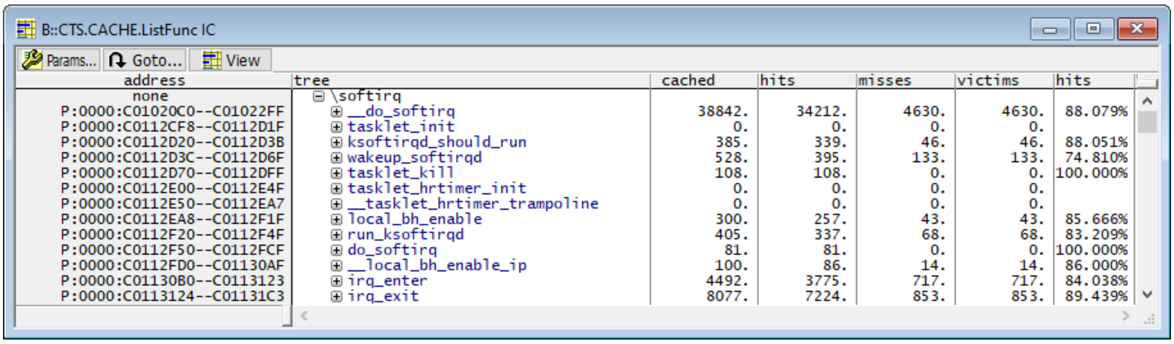
```
CTS.CACHE.ListAddress IC 0x8000--0x12000
```

See also

- CTS.CACHE
- CTS.CACHE.state

Format: **CTS.CACHE.ListFunc IC** | **DC** | **L2** | **L3** [*<range>* | *<address>*]

Performs a function-based cache analysis.



address	tree	cached	hits	misses	victims	hits
none	\softirq					
P:0000:C01020C0--C01022FF	__do_softirq	38842.	34212.	4630.	4630.	88.079%
P:0000:C0112CF8--C0112D1F	tasklet_init	0.	0.	0.	0.	
P:0000:C0112D20--C0112D3B	ksoftirqd_should_run	385.	339.	46.	46.	88.051%
P:0000:C0112D3C--C0112D6F	wakeup_softirqd	528.	395.	133.	133.	74.810%
P:0000:C0112D70--C0112DFF	tasklet_kill	108.	108.	0.	0.	100.000%
P:0000:C0112E00--C0112E4F	tasklet_hrtimer_init	0.	0.	0.	0.	
P:0000:C0112E50--C0112EA7	__tasklet_hrtimer_trampoline	0.	0.	0.	0.	
P:0000:C0112EA8--C0112F1F	local_bh_enable	300.	257.	43.	43.	85.666%
P:0000:C0112F20--C0112F4F	run_ksoftirqd	405.	337.	68.	68.	83.209%
P:0000:C0112F50--C0112FCF	do_softirq	81.	81.	0.	0.	100.000%
P:0000:C0112FD0--C01130AF	__local_bh_enable_ip	100.	86.	14.	14.	86.000%
P:0000:C01130B0--C0113123	irq_enter	4492.	3775.	717.	717.	84.038%
P:0000:C0113124--C01131C3	irq_exit	8077.	7224.	853.	853.	89.439%

```
CTS.CACHE.ListFunc IC 8000++0fff       ; perform a function based cache
                                       ; cache analysis for the specified
                                       ; address range
```

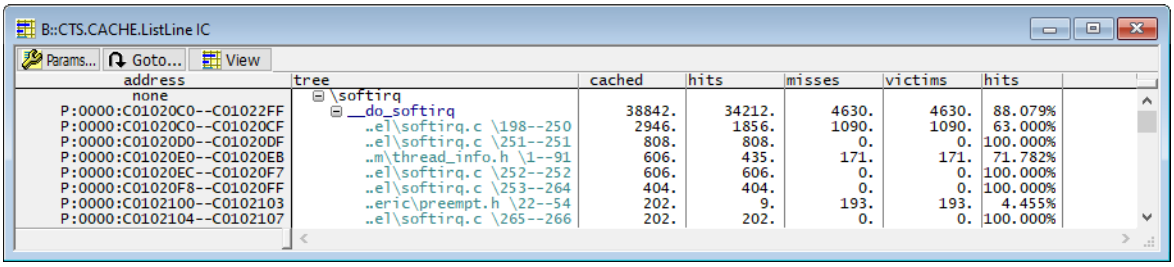
See also

■ CTS.CACHE ■ CTS.CACHE.state

CTS.CACHE.ListLine

Format: **CTS.CACHE.ListLine IC** | **DC** | **L2** | **L3** [*<range>* | *<address>*]

Performs an HLL-line-based cache analysis.



```
          address          |tree                          cached    |hits      |misses    |victims   |hits
           none            |  ⊟ \softirq
 P:0000:C01020C0--C01022FF |    ⊟ __do_softirq             38842.     34212.      4630.      4630.   88.079%
 P:0000:C01020C0--C01020CF |       ..el\softirq.c \198--250  2946.      1856.      1090.      1090.   63.000%
 P:0000:C01020D0--C01020DF |       ..el\softirq.c \251--251   808.       808.         0.         0. 100.000%
 P:0000:C01020E0--C01020EB |       ..m\thread_info.h \1--91   606.       435.       171.       171.  71.782%
 P:0000:C01020EC--C01020F7 |       ..el\softirq.c \252--252   606.       606.         0.         0. 100.000%
 P:0000:C01020F8--C01020FF |       ..el\softirq.c \253--264   404.       404.         0.         0. 100.000%
 P:0000:C0102100--C0102103 |       ..eric\preempt.h \22--54   202.         9.       193.       193.   4.455%
 P:0000:C0102104--C0102107 |       ..el\softirq.c \265--266   202.       202.         0.         0. 100.000%
```

```
CTS.CACHE.ListLine IC dosomethingbad
```

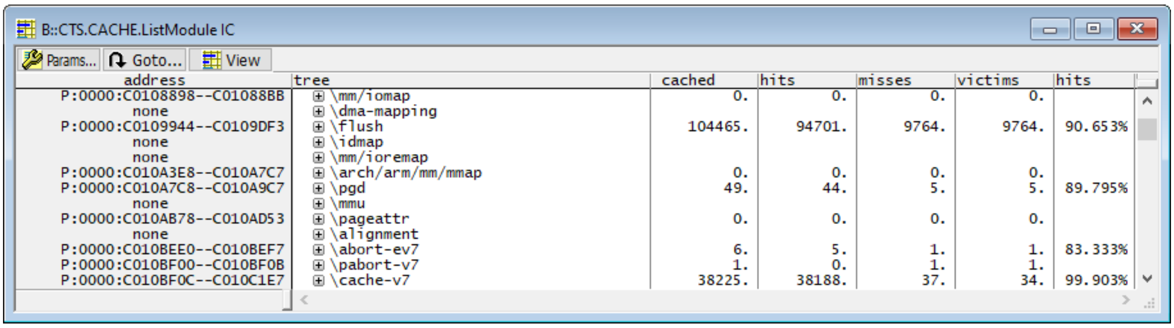
See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.ListModules

Format: **CTS.CACHE.ListModules IC** | **DC** | **L2** | **L3** [*<range>* | *<address>*]

Performs a module-based cache analysis.



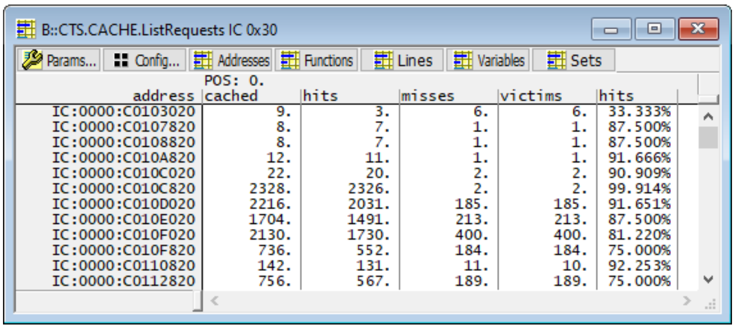
```
          address          |tree                          cached    |hits      |misses    |victims   |hits
 P:0000:C0108898--C01088BB |  ⊞ \mm/iomap                      0.         0.         0.         0.
           none            |  ⊞ \dma-mapping
 P:0000:C0109944--C0109DF3 |  ⊞ \flush                   104465.     94701.      9764.      9764.   90.653%
           none            |  ⊞ \idmap
           none            |  ⊞ \mm/ioremap
 P:0000:C010A3E8--C010A7C7 |  ⊞ \arch/arm/mm/mmap               0.         0.         0.         0.
 P:0000:C010A7C8--C010A9C7 |  ⊞ \pgd                           49.        44.         5.         5.   89.795%
           none            |  ⊞ \mmu
 P:0000:C010AB78--C010AD53 |  ⊞ \pageattr                      0.         0.         0.         0.
           none            |  ⊞ \alignment
 P:0000:C010BEE0--C010BEF7 |  ⊞ \abort-ev7                     6.         5.         1.         1.   83.333%
 P:0000:C010BF00--C010BF0B |  ⊞ \pabort-v7                     1.         0.         1.         1.
 P:0000:C010BF0C--C010C1E7 |  ⊞ \cache-v7                  38225.     38188.        37.        34.   99.903%
```

See also

- CTS.CACHE
- CTS.CACHE.state

Format: **CTS.CACHE.ListRequests IC** | **DC** | **L2** | **L3** *<address>*

Display which addresses compete for the same cache line.



```
                 POS: 0.
          address cached     |hits      |misses    |victims   |hits
  IC:0000:C0103020       9.        3.         6.         6.   33.333%
  IC:0000:C0107820       8.        7.         1.         1.   87.500%
  IC:0000:C0108820       8.        7.         1.         1.   87.500%
  IC:0000:C010A820      12.       11.         1.         1.   91.666%
  IC:0000:C010C020      22.       20.         2.         2.   90.909%
  IC:0000:C010C820    2328.     2326.         2.         2.   99.914%
  IC:0000:C010D020    2216.     2031.       185.       185.   91.651%
  IC:0000:C010E020    1704.     1491.       213.       213.   87.500%
  IC:0000:C010F020    2130.     1730.       400.       400.   81.220%
  IC:0000:C010F820     736.      552.       184.       184.   75.000%
  IC:0000:C0110820     142.      131.        11.        10.   92.253%
  IC:0000:C0112820     756.      567.       189.       189.   75.000%
```

```
CTS.CACHE.ListRequests IC 0x30    ; Display which addresses compete for
                                  ; the cache line 0x30 of the instruction
                                  ; cache
```

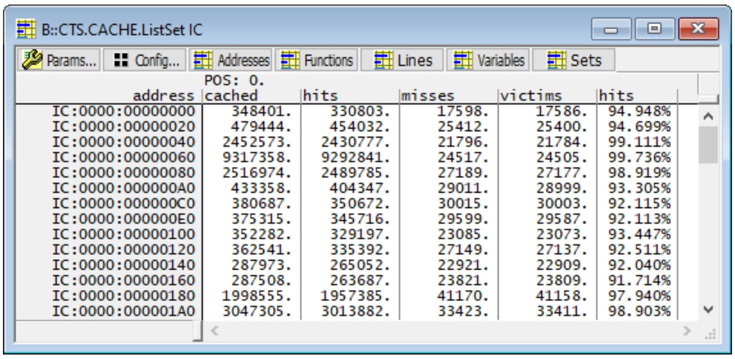
See also

■ CTS.CACHE ■ CTS.CACHE.state

CTS.CACHE.ListSet
Cache set based cache analysis

Format: **CTS.CACHE.ListSet IC | DC | L2 | L3**

Performs a cache analysis based on cache sets.



```
                    POS: 0.
             address cached      hits        misses      victims     hits
 IC:0000:00000000     348401.     330803.      17598.      17586.   94.948%
 IC:0000:00000020     479444.     454032.      25412.      25400.   94.699%
 IC:0000:00000040    2452573.    2430777.      21796.      21784.   99.111%
 IC:0000:00000060    9317358.    9292841.      24517.      24505.   99.736%
 IC:0000:00000080    2516974.    2489785.      27189.      27177.   98.919%
 IC:0000:000000A0     433358.     404347.      29011.      28999.   93.305%
 IC:0000:000000C0     380687.     350672.      30015.      30003.   92.115%
 IC:0000:000000E0     375315.     345716.      29599.      29587.   92.113%
 IC:0000:00000100     352282.     329197.      23085.      23073.   93.447%
 IC:0000:00000120     362541.     335392.      27149.      27137.   92.511%
 IC:0000:00000140     287973.     265052.      22921.      22909.   92.040%
 IC:0000:00000160     287508.     263687.      23821.      23809.   91.714%
 IC:0000:00000180    1998555.    1957385.      41170.      41158.   97.940%
 IC:0000:000001A0    3047305.    3013882.      33423.      33411.   98.903%
```

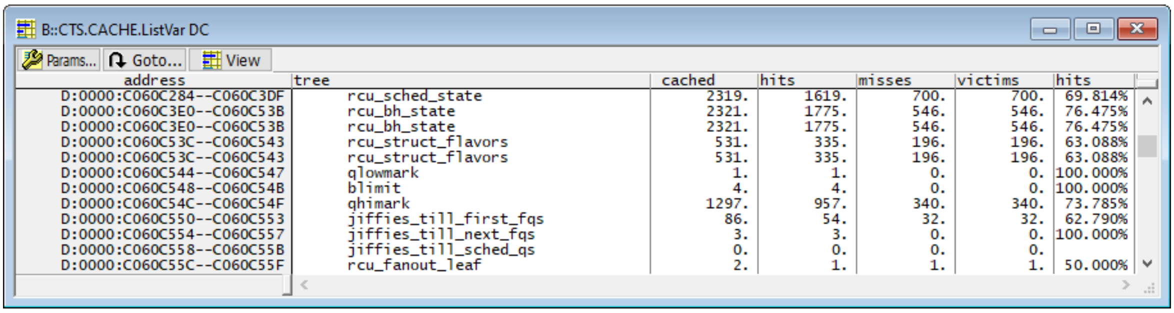
See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.ListVar
Variable based cache analysis

Format: **CTS.CACHE.ListVar IC | DC | L2 | L3** [*<range>* | *<address>*]

Performs a cache analysis based on variables.



```
              address          tree                                cached      hits      misses     victims    hits
  D:0000:C060C284--C060C3DF           rcu_sched_state               2319.      1619.        700.       700.   69.814%
  D:0000:C060C3E0--C060C53B           rcu_bh_state                  2321.      1775.        546.       546.   76.475%
  D:0000:C060C3E0--C060C53B           rcu_bh_state                  2321.      1775.        546.       546.   76.475%
  D:0000:C060C53C--C060C543           rcu_struct_flavors             531.       335.        196.       196.   63.088%
  D:0000:C060C53C--C060C543           rcu_struct_flavors             531.       335.        196.       196.   63.088%
  D:0000:C060C544--C060C547           qlowmark                         1.         1.          0.         0.  100.000%
  D:0000:C060C548--C060C54B           blimit                           4.         4.          0.         0.  100.000%
  D:0000:C060C54C--C060C54F           qhimark                       1297.       957.        340.       340.   73.785%
  D:0000:C060C550--C060C553           jiffies_till_first_fqs          86.        54.         32.        32.   62.790%
  D:0000:C060C554--C060C557           jiffies_till_next_fqs            3.         3.          0.         0.  100.000%
  D:0000:C060C558--C060C55B           jiffies_till_sched_qs            0.         0.          0.         0.
  D:0000:C060C55C--C060C55F           rcu_fanout_leaf                  2.         1.          1.         1.   50.000%
```

See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.MMUArchitecture   Define MMU architecture for cache control

Format:	**CTS.CACHE.MMUArchitecture** *<control>*
<control>:	**NONE** **ARM920T | ARM922T | ARM925T | ARM926EJ | ARM946E** **ARM1136J | ARM1156T2 | ARM1176JZ | ARM11MPCORE** **CortexA5 | CortexA7 | CortexA8 | CortexA9** **CortexR4 | CortexR5 | CortexR7 | CortexR8** **MXPLMEM** **SCORPION** **E200MMU | E200MPU | E200FLASH | E200FLASH2** **M340** **MCF5272** **SC140E** **NIOS2E | NIOS2S | NIOS2F** **TC1766 | TC1796**

If the MMU architecture is set, the cache analysis takes **all manipulations on the cache control registers** into account for the cache analysis:

- Cache flushes
- Switch-on and switch-off of the caches
- Cache locks

If **CTS.CACHE.MMUArchitecture** is set to **NONE**, the manipulations on the cache control registers are not taken into account for the cache analysis.

See also

■ CTS.CACHE   ■ CTS.CACHE.state

Format:	**CTS.CACHE.Mode IC** | **DC** | **L2** | **L3** *<mode>*
<mode>:	**CopyBack** **WriteThrough** **MMU**

This command defines the strategy used for the memory coherency for each cache.

CopyBack	Copy back strategy guarantees memory coherency. When a cache hit occurred for a data store/write, the cache contents is updated and the corresponding cache line is marked as dirty. The data value is copied back to memory when the contents of the cache line is evicted.
WriteThrough	Write Through strategy guarantees memory coherency. When a cache hit occurs for a data store/write, the cache contents is updated and the data is also stored/written to memory.
MMU	The strategy for memory coherency is taken from the MMU.

See also

- CTS.CACHE
- CTS.CACHE.state

Format:	**CTS.CACHE.Replacement** *<cache> <replace>*
<cache>:	**IC** | **DC** | **L2** | **L3** | **ITLB** | **DTLB** | **TLB0** | **TLB1**
<replace>:	**Cyclic** **FreeCyclic** **PseudoCyclic** **FreePseudoCyclic** **Random** **FreeRandom** **LRU** **MMU**

This command defines the replacement strategy for each cache.

Cyclic	Cyclic (round-robin) replacement strategy is used. One round robin counter for each cache set.
FreeCyclic	Cyclic (round-robin) replacement strategy is used, but if an empty cache line is found it is filled first.
PseudoCyclic	Cyclic (round-robin) replacement strategy is used. But there is only one round robin counter for all cache sets.
FreePseudoCyclic	Cyclic (round-robin) replacement strategy is used • but if an empty cache line is found it is filled first • but there is only one round robin counter for all cache sets
Random	Random replacement strategy is used.
FreeRandom	Random replacement strategy is used, but if an empty cache line is found it is filled first.
LRU	Last recently used replacement strategy is used.
MMU	The replacement strategy is defined by the CPU. Please use **CTS.CACHE.Replacement MMU** is your CPU uses a not listed replacement strategy.

See also

■ CTS.CACHE ■ CTS.CACHE.state

CTS.CACHE.RESet — Reset settings of CTS cache window

Format:	**CTS.CACHE.RESet**

Resets the settings of the **CTS.CACHE** window.

See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.SETS — Define the number of cache sets

Format:	**CTS.CACHE.SETS** *<cache> <sets>*
<cache>:	**IC** | **DC** | **L2** | **L3** | **ITLB** | **DTLB** | **TLB0** | **TLB1**

This command defines the number of cache sets for each cache.

```
CTS.CACHE.SETS IC 4.                    ; The instruction CACHE has 4 sets
CTS.CACHE.SETS DC 4.                    ; The data CACHE has 4 sets
```

See also

- CTS.CACHE
- CTS.CACHE.state

CTS.CACHE.Sort — Define sorting for all list commands

Format:	**CTS.CACHE.Sort OFF** | **Address** | **Victims**

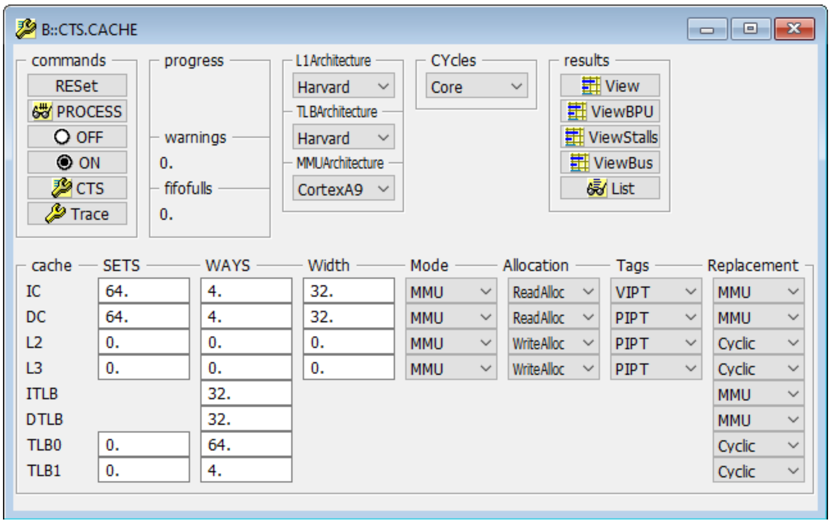
Defines the sorting for all list commands.

See also

- CTS.CACHE
- CTS.CACHE.state

Format:	**CTS.CACHE.state**

Displays the cache structure of your CPU in the **CTS.CACHE.state** window. For background information, see **CTS.CACHE**.



progress

warnings
0.

fifofulls
0.

L1Architecture: Harvard

TLBArchitecture: Harvard

MMUArchitecture: CortexA9

CYcles: Core

cache	SETS	WAYS	Width	Mode	Allocation	Tags	Replacement
IC	64.	4.	32.	MMU	ReadAlloc	VIPT	MMU
DC	64.	4.	32.	MMU	ReadAlloc	PIPT	MMU
L2	0.	0.	0.	MMU	WriteAlloc	PIPT	Cyclic
L3	0.	0.	0.	MMU	WriteAlloc	PIPT	Cyclic
ITLB		32.					MMU
DTLB		32.					MMU
TLB0	0.	64.					Cyclic
TLB1	0.	4.					Cyclic

See also

- CTS.CACHE
- CTS.CACHE.CYcles
- CTS.CACHE.L1Architecture
- CTS.CACHE.ListAddress
- CTS.CACHE.ListLine
- CTS.CACHE.ListRequests
- CTS.CACHE.ListVar
- CTS.CACHE.Mode
- CTS.CACHE.RESet
- CTS.CACHE.Sort
- CTS.CACHE.TLBArchitecture
- CTS.CACHE.ViewBPU
- CTS.CACHE.ViewStalls
- CTS.CACHE.Width
- CTS.CACHE.Allocation
- CTS.CACHE.DefineBus
- CTS.CACHE.LFSR
- CTS.CACHE.ListFunc
- CTS.CACHE.ListModules
- CTS.CACHE.ListSet
- CTS.CACHE.MMUArchitecture
- CTS.CACHE.Replacement
- CTS.CACHE.SETS
- CTS.CACHE.Tags
- CTS.CACHE.View
- CTS.CACHE.ViewBus
- CTS.CACHE.WAYS
- CTS.state

Format:	**CTS.CACHE.Tags IC** | **DC** | **L2** | **L3** *<tag>*
<tag>:	**VIVT** **PIPT** **VIPT** **AVIVT**

Defines the cache structure.

VIVT	Virtual Index, Virtual Tag The logical address is used as tag for a cache line.
PIPT	Physical Index, Physical Tag The physical address is used as tag for a cache line.
VIPT	Virtual Index, Physical Tag
AVIVT	Address Space ID + Virtual Index, Virtual Tag

See also

■ CTS.CACHE ■ CTS.CACHE.state

CTS.CACHE.TLBArchitecture

Define architecture for the TLB

Format:	**CTS.CACHE.TLBArchitecture Harvard | Unified | UnifiedSplit**

This command defines the architecture for the TLB cache.

Harvard	The TLB cache has Harvard architecture, that means there is an instruction TLB and a data TLB available.
Unified	The TLB cache is a unified cache, that means the same TLB is used for instruction fetches and data loads/stores.
UnifiedSplit	The TLB cache is a unified cache, that means the same TLB is used for instruction fetches and data loads/stores. But TRACE32 splits the unified cache in an instruction and data TLB for the cache analysis. The splitting is based on the cycles type (e.g. read/write/ptrace/exec). (not implemented yet)

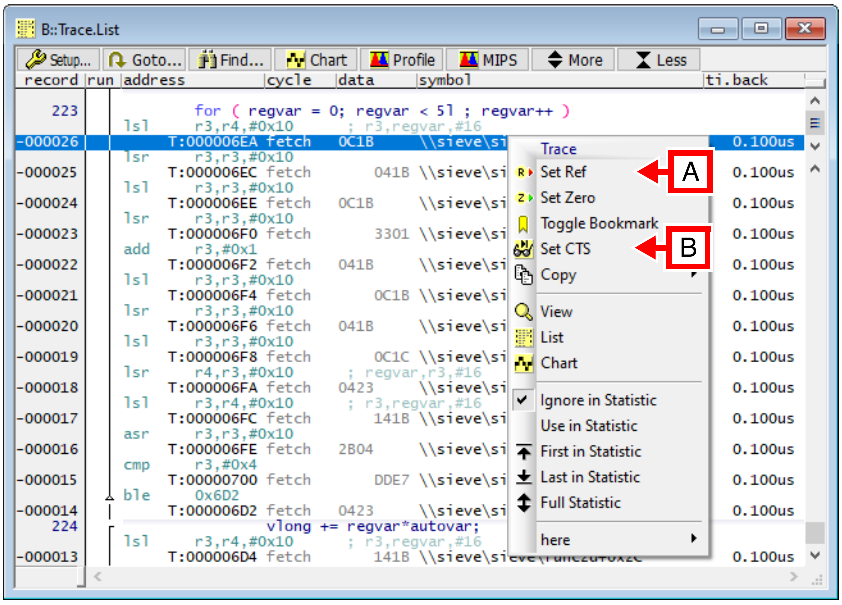
See also

- CTS.CACHE
- CTS.CACHE.state

Format:	**CTS.CACHE.View**

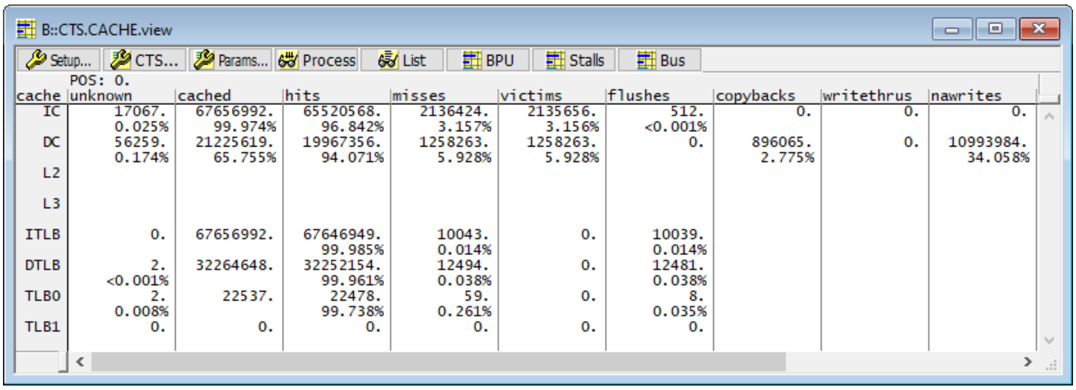
Displays the results for the cache analysis. **CTS.Mode CACHE** has to be selected before any calculation can be started. The calculation of the results for the cache analysis can be activated as follows:

- By using the command **CTS.PROCESS**. That way the complete trace contents is analyzed.
- By selecting a part of the trace contents e.g. a function. The starting point for the analysis is selected by setting a reference point (command **Analyzer.REF**) to the relevant trace record **[A]**. The endpoint for the analysis is selected by setting the CTS point (command **CTS.GOTO**) to the relevant trace record **[B]**.

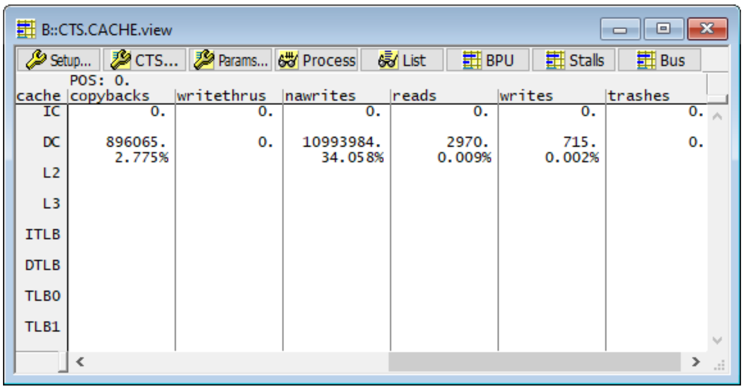


```
record |run |address          |cycle  |data      |symbol                         |ti.back

    223           for ( regvar = 0; regvar < 51 ; regvar++ )
         lsl      r3,r4,#0x10       ; r3,regvar,#16
-000026       T:000006EA fetch   0C1B      \\sieve\si                         0.100us
         lsr      r3,r3,#0x10
-000025       T:000006EC fetch       041B \\sieve\si                          0.100us
         lsl      r3,r3,#0x10
-000024       T:000006EE fetch   0C1B      \\sieve\si                         0.100us
         lsr      r3,r3,#0x10
-000023       T:000006F0 fetch       3301 \\sieve\si                          0.100us
         add      r3,#0x1
-000022       T:000006F2 fetch   041B      \\sieve\si                         0.100us
         lsl      r3,r3,#0x10
-000021       T:000006F4 fetch       0C1B \\sieve\si                          0.100us
         lsr      r3,r3,#0x10
-000020       T:000006F6 fetch   041B      \\sieve\si                         0.100us
         lsl      r3,r3,#0x10
-000019       T:000006F8 fetch       0C1C \\sieve\si                          0.100us
         lsr      r4,r3,#0x10       ; regvar,r3,#16
-000018       T:000006FA fetch   0423      \\sieve\si                         0.100us
         lsl      r3,r4,#0x10       ; r3,regvar,#16
-000017       T:000006FC fetch       141B \\sieve\si                          0.100us
         asr      r3,r3,#0x10
-000016       T:000006FE fetch   2B04      \\sieve\si                         0.100us
         cmp      r3,#0x4
-000015       T:00000700 fetch       DDE7 \\sieve\si                          0.100us
         ble      0x6D2
-000014       T:000006D2 fetch   0423      \\sieve\si                         0.100us
    224                   vlong += regvar*autovar;
         lsl      r3,r4,#0x10       ; r3,regvar,#16
-000013       T:000006D4 fetch       141B \\sieve\sieve\func2d+0x2c           0.100us
```

The result:

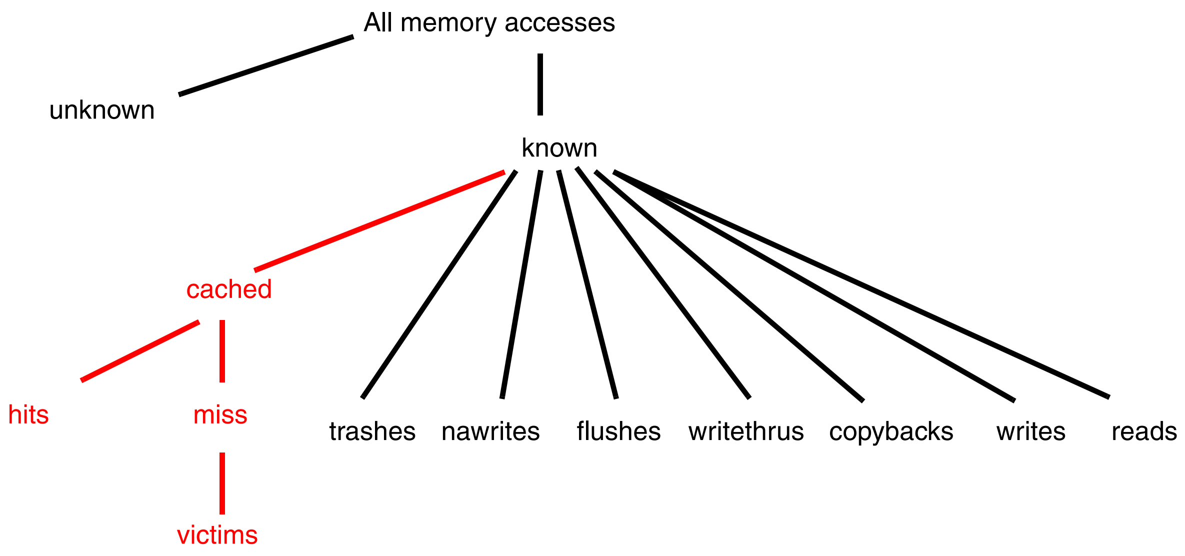


```
      POS: 0.
cache |unknown   |cached    |hits      |misses    |victims   |flushes   |copybacks |writethrus|nawrites
   IC     17067.  67656992.  65520568.   2136424.   2135656.       512.         0.         0.         0.
          0.025%    99.974%    96.842%     3.157%     3.156%   <0.001%
   DC     56259.  21225619.  19967356.   1258263.   1258263.         0.    896065.         0.  10993984.
          0.174%    65.755%    94.071%     5.928%     5.928%               2.775%               34.058%
   L2

   L3

 ITLB         0.  67656992.  67646949.     10043.         0.     10039.
                               99.985%     0.014%                0.014%
 DTLB         2.  32264648.  32252154.     12494.         0.     12481.
         <0.001%               99.961%     0.038%                0.038%
 TLB0         2.     22537.     22478.        59.         0.         8.
          0.008%               99.738%     0.261%                0.035%
 TLB1         0.         0.         0.         0.         0.         0.
```



```
       POS: 0.
cache |copybacks  |writethrus |nawrites   |reads      |writes     |trashes
   IC          0.          0.          0.          0.          0.          0.

   DC     896065.          0.   10993984.       2970.        715.          0.
          2.775%                  34.058%      0.009%      0.002%
   L2

   L3

 ITLB

 DTLB

 TLB0

 TLB1
```

Interpretation of the result:



Description of Buttons in the CTS.CACHE.View Window

[Back to Top]

Setup	Display a **Trace** configuration window.
CTS	Display **CTS** settings window.
Params	Display information about the cache structure (**CTS.CACHE.state**).
Process	Initiate calculation for cache analysis (**CTS.PROCESS**).
List	Display a **CTS** listing (**CTS.List**).
BPU	Display a statistic for branch prediction unit (**CTS.CACHE.ViewBPU**).
Stalls	Display a statistic for idles/stalls (**CTS.CACHE.ViewStalls**).
Bus	Display a statistic for bus utilization (**CTS.CACHE.ViewBus**).

Description of Columns in the CTS.CACHE.View Window

[Back to Top]

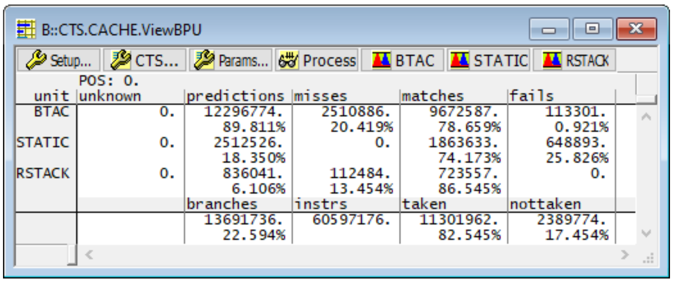
unknown	All accesses for which TRACE32 has no information The cache analysis is based on the memory addresses recorded in the trace buffer. Before the first memory address is mapped to a specific cache line the contents of this cache line is unknown. Other reasons for unknown are: gaps in the trace recording, missing address information etc. (percentage is based on all memory accesses)
cached	Number of accesses to cached addresses (percentage is based on all memory accesses)
hits	Number of cache hits (percentage is based on all cached accesses)
miss	Number of cache misses (percentage is based on all cached accesses)
victims	Number of cache victims (percentage is based on all cached accesses)
flushes	Number of cache lines that were flushed (percentage is based on all memory accesses)
copybacks	Number of cache lines that were copied back to memory (percentage is based on all memory accesses)

writethrus	Number of cache lines that were written through to memory (percentage is based on all memory accesses)
nawrites	Writes in a read-allocated cache (percentage is based on all memory accesses)
reads	Number of not-cached reads (percentage is based on all memory accesses)
writes	Number of not-cached writes (percentage is based on all memory accesses)
trashes	Discarded accesses (ARM11 only) (percentage is based on all memory accesses)

See also

- CTS.CACHE
- CTS.CACHE.state

Format:	**CTS.CACHE.ViewBPU**



```
          POS: 0.
  unit |unknown         |predictions |misses      |matches     |fails
  BTAC |              0.|  12296774. |   2510886. |   9672587. |   113301.
       |                |   89.811%  |   20.419%  |   78.659%  |    0.921%
STATIC |              0.|   2512526. |         0. |   1863633. |   648893.
       |                |   18.350%  |            |   74.173%  |   25.826%
RSTACK |              0.|    836041. |    112484. |    723557. |        0.
       |                |    6.106%  |   13.454%  |   86.545%  |
       |                |branches    |instrs      |taken       |nottaken
       |                |  13691736. |  60597176. |  11301962. |  2389774.
       |                |   22.594%  |            |   82.545%  |   17.454%
```

BTAC	Branch Target Address Cache / Branch Folding
STATIC	Static Branch Predictor
RSTACK	Return Stack

For details about the program flow prediction please refer to your processor manual.

instrs	Total number of instructions.
branches	Total number of branches.
taken	Number of taken branches.
nottaken	Number of not-taken branches.

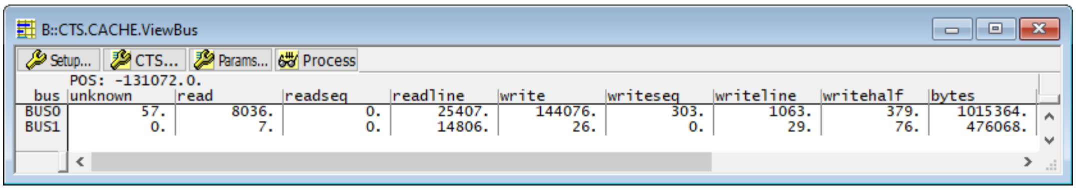
predictions	Total number of branch predictions.
unknown	Since the contents of Branch Target Address Cache is unknown at the beginning of the analysis, the first *<size_of_branch_target_address_cache>* predictions are unknown.
misses	No entry was found in the Branch Target Address Cache for the branch source address.
hits	An entry for the branch source address was found in the Branch Target Address Cache and the prediction was correct.
fails	An entry for the branch source address was found in the Branch Target Address Cache, but the prediction failed.

See also

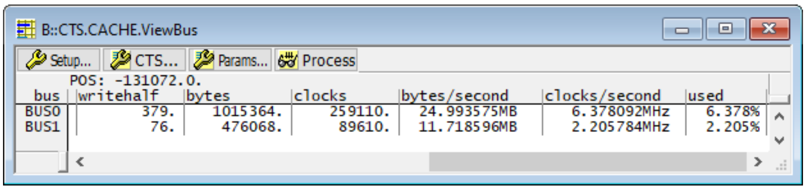
- CTS.CACHE
- CTS.CACHE.state

Format:	**CTS.CACHE.ViewBus**

Displays a detailed analysis of the bus utilization.



```
      POS: -131072.0.
  bus  unknown     read        readseq     readline    write       writeseq    writeline   writehalf   bytes
 BUS0        57.       8036.          0.      25407.     144076.        303.       1063.        379.    1015364.
 BUS1         0.          7.          0.      14806.         26.          0.         29.         76.     476068.
```



```
      POS: -131072.0.
  bus  writehalf   bytes       clocks      bytes/second     clocks/second    used
 BUS0        379.    1015364.     259110.   24.993575MB      6.378092MHz      6.378%
 BUS1         76.     476068.      89610.   11.718596MB      2.205784MHz      2.205%
```

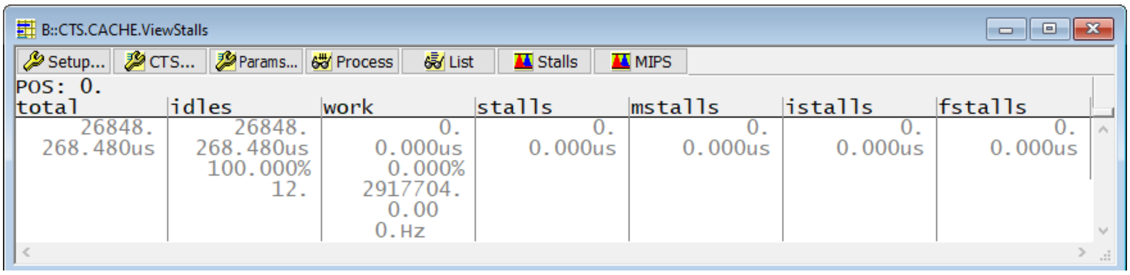
unknown	Number of clock cycles consumed by memory accesses that are categorized as unknown by the cache analysis
read	Number of clock cycles consumed by memory read accesses
readseq	Number of clock cycles consumed by subsequent memory read accesses (e.g. burst access)
readline	Number of clock cycles consumed by cache line fill operations
write	Number of clock cycles consumed by memory write accesses
writeseq	Number of clock cycles consumed by subsequent memory write accesses (e.g. burst access)
writeline	Number of clock cycles consumed by writing the contents of a cache line back to memory (copy back)
writehalf	Number of clock cycles consumed by writing the contents of half a cache line back to memory (copy back)
bytes	Number of bytes transferred via the external bus interface
clocks	Number of clock cycles the external bus was busy
bytes/s	Transmission rate
clocks/s	Transmission frequency
used	Bus load in percentage

See also

■ CTS.CACHE ■ CTS.CACHE.state

Format:	**CTS.CACHE.ViewStalls**

Analyses over the measurement interval how much cycles/time was taken by idles/stalls and how much cycles/time the CPU was really working.



```
POS: 0.
total      |idles      |work       |stalls     |mstalls    |istalls    |fstalls
    26848.      26848.         0.          0.          0.          0.          0.
 268.480us   268.480us    0.000us     0.000us     0.000us     0.000us     0.000us
             100.000%     0.000%
                  12.   2917704.
                           0.00
                           0.Hz
```

total	Number of analyzed clock cycles measurement time
idles	Number of idles cycles (the CPU is not executing instructions) time the CPU was in idle mode percentage of time/clocks the CPU was in idle mode number of time the CPU was in idle state The number of idles states is calculated as follows: • number of times the CPU went in power-down or sleep mode (e.g. for the ARM architecture the number of times a Wait for Interrupt CP15 operation was performed) • number of times a single instruction last more the 1000. clock cycles
work	Number of cycles the processor was working time the CPU was working percentage of time the processor was working number of instructions that were executed by the processor
stalls	Number of stalls time the CPU was stalled percentage of time the CPU was stalled
mstalls	Number of memory stalls time taken by memory stalls percentage of time taken by memory stalls Memory stalls are caused by e.g. cache misses, TLB misses, accesses to slow memory …

istalls	Number of interlock stalls time taken by interlock stalls percentage of time taken by interlock stalls Interlock stalls are caused by e.g. resource conflicts between instructions, data dependencies …
fstalls	Number of fetch stalls time taken by fetch stalls percentage of time taken by fetch stalls Fetch stalls are caused by e.g. pipeline reload etc.

See also

■ CTS.CACHE ■ CTS.CACHE.state

▲ 'Release Information' in 'Legacy Release History'

CTS.CACHE.WAYS Define number of cache ways

Format:	**CTS.CACHE.WAYS** *<cache> <ways>*
<cache>:	**IC** | **DC** | **L2** | **L3** | **ITLB** | **DTLB** | **TLB0** | **TLB1**

This command defines the number of cache ways (blocks) for each cache.

```
CTS.CACHE.WAYS IC 4.              ; The instruction CACHE has 4 blocks

CTS.CACHE.WAYS DC 4.              ; The data CACHE has 4 blocks
```

See also

■ CTS.CACHE ■ CTS.CACHE.state

CTS.CACHE.Width — Define width of cache line

Format: **CTS.CACHE.Width IC | DC | L2 | L3** *<width>*

This command define the width of a single cache line in bytes.

```
CTS.CACHE.Width IC 32.              ; A cache line for the instruction cache
                                    ; is 32. byte
```

See also

- ■ CTS.CACHE
- ■ CTS.CACHE.state

CTS.CAPTURE — Copy real memory to the virtual memory for CTS

Format: **CTS.CAPTURE**

Copies “real” memory to the TRACE32 virtual memory (VM:) for all places where VM: is already mapped.

```
...
; Capture a snapshot of the system for the analysis.
CTS.CAPTURE
Go ; Start the analysis.
Break ; Stop the analysis.
...
```

See also

- ■ CTS
- ■ CTS.state
- ■ CTS.UseCACHE
- ■ CTS.UseVM
- ▲ ’Release Information’ in ’Legacy Release History’

CTS.Chart.ChildTREE   Display callee context of a function as chart

Format: **CTS.Chart.ChildTREE** *<address>*

Show call tree and run-time of all functions called by the specified functions based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

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

See also

- <trace>.Chart.ChildTREE

CTS.Chart.Func   Function activity chart

Format: **CTS.Chart.Func** [*<trace_area>*] [/*<option>*]

Displays the time spent in different functions as chart based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

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

See also

- <trace>.Chart.Func

CTS.Chart.INTERRUPT   Display interrupt chart

Format: **CTS.Chart.INTERRUPT** [*<trace_area>*] [/*<option>*]

Displays the time spent in different interrupts as time chart based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

Refer to **<trace>.Chart.INTERRUPT** for a description of the parameters and options.

See also

- <trace>.Chart.INTERRUPT

CTS.Chart.INTERRUPTTREE — Display interrupt nesting

Format: **CTS.Chart.INTERRUPTTREE** [*<trace_area>*] [/*<option>*]

Displays the interrupt nesting as time chart based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

Refer to **<Trace>.Chart.INTERRUPTTREE** for a description of the parameters and options.

See also

- <trace>.Chart.INTERRUPTTREE

CTS.Chart.Nesting — Show function nesting at cursor position

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

Shows the function call stack as a time chart based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

Refer to **<trace>.Chart.Nesting** for a description of the parameters and options.

See also

- <trace>.Chart.Nesting

CTS.Chart.RUNNABLE — Runnable activity chart

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

The time spent in different AUTOSAR Runnables is displayed graphically. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **“OS Awareness Manual NORTi”** (rtos_norti.pdf) for more information.

Refer to **<trace>.Chart.Nesting** for a description of the parameters and options.

See also

- <trace>.Chart.RUNNABLE

CTS.Chart.sYmbol Execution time at different symbols as chart

Format:	**CTS.Chart.sYmbol** [*<trace_area>*] [/*<option>*]

Displays the distribution of program execution time at different symbols as a time chart based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

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

See also

- CTS
- CTS.PROfileChart
- CTS.state
- <trace>.Chart.sYmbol

CTS.Chart.TASK Task activity chart

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

Displays the time spent in different tasks based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature is only available if TRACE32 has been set for OS-aware debugging.

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

See also

- <trace>.Chart.TASK

CTS.Chart.TASKINFO — Chart for context ID special messages

Format: **CTS.Chart.TASKINFO** [<*trace_area*>] [/<*option*>]

Displays a graphical chart based on the CTS data for special messages written to the context ID register (ETM trace).

Refer to **<trace>.Chart.TASKINFO** for a description of the parameters and options.

See also

- <trace>.Chart.TASKINFO

CTS.Chart.TASKINTR — Display ISR2 time chart (ORTI)

Format: **CTS.Chart.TASKINTR** [<*trace_area*>] [**/**<*option*>]

Displays an ORTI based ISR2 time chart based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **“OS Awareness Manual NORTi”** (rtos_norti.pdf) for more information.

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

See also

- <trace>.Chart.TASKINTR

CTS.Chart.TASKKernel — Display task time chart with kernel markers (ORTI)

Format: **CTS.Chart.TASKKernel** [<*trace_area*>] [**/**<*option*>]

Similar command to **<trace>.Chart.TASKKernel**. The analysis is however based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature is only available if TRACE32 has been set for OS-aware debugging.

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

See also

■ <trace>.Chart.TASKKernel

CTS.Chart.TASKORINTERRUPT — Task and interrupt activity chart

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

Displays the time spent in different tasks and interrupts based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to **<trace>.Chart.TASKORINTERRUPT** for a description of the parameters and options.

See also

■ <trace>.Chart.TASKORINTERRUPT

CTS.Chart.TASKSRV — Service routine run-time analysis

Format: **CTS.Chart.TASKSRV** [*<trace_area>*] [**/***<option>*]

The time spent in OS service routines and different tasks is displayed. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual NORTi"** (rtos_norti.pdf) for more information.

Refer to **<Trace>.Chart.TASKSRV** for a description of the parameters and options.

CTS.Chart.TASKVSINTERRUPT — Time chart of interrupted tasks

Format: **CTS.Chart.TASKVSINTERRUPT** [*<trace_area>*] [**/***<option>*]

Shows a graphical representation of tasks that were interrupted by interrupt service routines based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to **<trace>.Chart.TASKVSINTERRUPT** for a description of the parameters and options.

See also

■ <trace>.Chart.TASKVSINTERRUPT

CTS.Chart.TASKVSINTR   Time chart of task-related interrupts

Format: **CTS.Chart.TASKVSINTR** [*<trace_area>*] [**/***<options>* …]

Displays a time-chart for task-related interrupt service routines based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**). This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual NORTi"** (rtos_norti.pdf) for more information.

Refer to **<trace>.Chart.TASKVSINTR** for a description of the parameters and options.

See also

■ <trace>.Chart.TASKVSINTR

CTS.Chart.TREE   Display function chart as tree view

Format: **CTS.Chart.TREE** [*<trace_area>*] [**/***<option>*]

The result of this command shows a graphical chart tree of the function nesting based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

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

See also

■ <trace>.Chart.TREE

CTS.EXPORT — Export trace data

Format:	**CTS.EXPORT** *<file>* [*<trace_area>*] [**/***<option>*]
<option>	**FILE** | **CORE** | **CACHE** | **BUS**

Exports the trace contents with CTS information for postprocessing by an external analysis tool. The command is similar to **<trace>.EXPORT**.

FILE	Exports the trace contents loaded with **<trace>.FILE**.
CORE	Exports core accesses.
CACHE	Exports cache accesses. This option is only available if **CTS.Mode CACHE** has been selected.
BUS	Exports bus accesses. This option is only available if **CTS.Mode CACHE** has been selected.

See also

■ CTS

CTS.FixedControl — Execution time at different symbols as chart

Format:	**CTS.FixedControl** [**ON** | **OFF**]

Fixes control register values to current value. Only supported for PowerPC E200ZX.

See also

■ CTS

CTS.GOTO — Select the specified record for CTS (absolute)

Format:	**CTS.GOTO** *<record>* [**/FILE**]

Selects the specified record for the trace based debugging. If CTS is OFF, CTS is switched to ON by this command.

This command can be used to set the starting point for trace-based debugging.

FILE	Takes trace memory contents loaded by **<trace>.LOAD**.

```
CTS.GOTO -123.                              ; Select record -123. for CTS
```

See also

■ CTS ■ CTS.state

CTS.INCremental — CTS displays intermediate results while processing

Format:	**CTS.INCremental** [**ON** | **OFF**]

ON	**CTS.List** displays intermediate results while TRACE32 is processing the trace contents.
OFF	**CTS.List** displays the result after TRACE32 has completely processed the trace contents.

See also

■ CTS ■ CTS.state

CTS.Init — Restart CTS processing

Format:	**CTS.Init**

Restarting the CTS processing has effects:

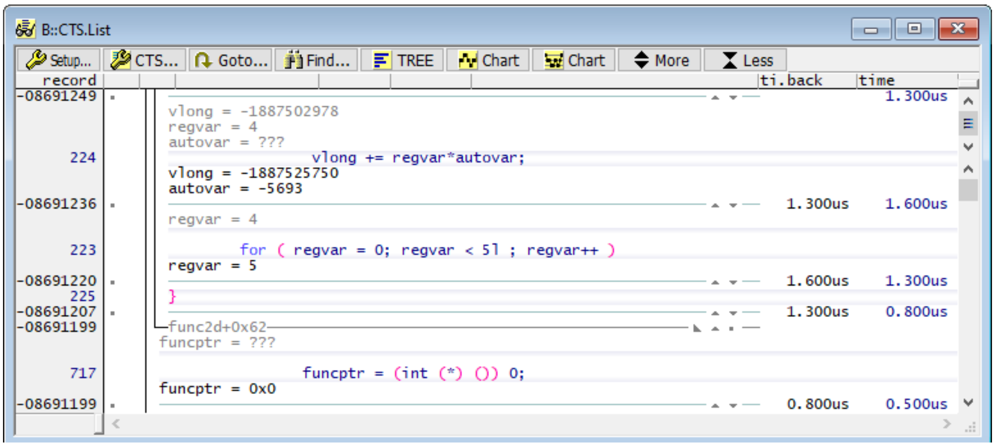
- **CTS.List** is reprocessed.
- The target context for trace-based debugging is re-processed.
- The new settings of the CTS window take effect.

See also

■ CTS ■ CTS.state

Format:	**CTS.List** [*<record>* | *<record_ range>*] [*<items>* …] [**/***<options>*]
<options>:	**FILE** **Track** **Mark** *<item>* **TASK** *<task_magic>* | *<task_id>* | *<task_name>* *<other_generic_options>*
<items>:	**%***<format>* **DEFault** | **ALL** | **CPU** | **LINE** | **PORTS** **Run** **CYcle** | **Data**[**.***<subitem>* | **BDATA** | **List**[**.***<subitem>*] **Address** | **BAddress** | **FAddress** | **sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** | **Var** **TIme**[**.***<subitem>*] **FUNC** | **FUNCR** | **FUNCVar** | **IGNORE** **LeVel** | **MARK**[**.***<marker>* | **FLAG**[**.***<flag_index>*] **Trigger** | **Trigger.A** | **Trigger.B** **SPARE** *<special_lines>*
<format>:	**Ascii** | **BINary** | **Decimal** | **Hex** | **Signed** | **Unsigned** **HighLow** | **Timing** **TimeAuto** | **TimeFixed** **LEN** *<size>*

<options>	For a detailed description of all other parameters and options, refer to the **<trace>.List** command.
TASK *<task_magic>*, etc.	Filters the **CTS.List** window by the specified task. See also **"What to know about the Task Parameters"** (general_ref_t.pdf).



```
   record                                                         ti.back    time
-08691249 ▪                                                                 1.300us
               vlong = -1887502978
               regvar = 4
               autovar = ???
      224                   vlong += regvar*autovar;
               vlong = -1887525750
               autovar = -5693
-08691236 ▪                                                      1.300us    1.600us
               regvar = 4
      223              for ( regvar = 0; regvar < 51 ; regvar++ )
               regvar = 5
-08691220 ▪                                                      1.600us    1.300us
      225      }
-08691207 ▪                                                      1.300us    0.800us
-08691199    └func2d+0x62
              funcptr = ???
      717                   funcptr = (int (*) ()) 0;
              funcptr = 0x0
-08691199 ▪                                                      0.800us    0.500us
```

Description of Buttons in the CTS.List Window

Setup …	Open a **<trace>.state** window to configure the trace.
CTS …	Open a **CTS.view** window to configure CTS.
Goto …	Open a **<trace>.GOTO** dialog box to move the cursor to a specific record.
Find …	Open a **<trace>.Find** dialog box to search for specific entries in the trace.
TREE	Open a **CTS.STATistic.TREE** window to display the call structure of the trace contents as a tree.
Chart	Opens a **CTS.Chart.sYmbol** window to display the program execution time at different symbols as a time chart.
Chart	Opens a **CTS.Chart.Func** window to display the time spent in different functions as chart.
More/Less	The **More** and **Less** button allow to switch between the following displays: • Interrupts and task levels • Function nesting • HLL lines • HLL lines and disassembled code • All CPU cycles

Cache analysis results (when enabled) are shown in the following formats:

- *<cache_mode> <cyclecount>***?**
  Information about a number of accesses is unknown.

- *<cache_mode> <hits>***/***<misses>*
  Regular cached cycles.

- *<cache_mode> <hits>***/***<misses>***/***<bypasses>*
  Bypasses are cycles that where not using the cache (non-allocated write cycles or trash cycles).

See also

■ CTS ■ CTS.state

▲ 'Release Information' in 'Legacy Release History'

CTS.ListNesting

Analyze function nesting

Format:	**CTS.ListNesting**[<*trace_area*>] [**/**<*option*>]

Investigates issues in the construction of the call tree for the nesting function run-time analysis based on the CTS data.

Refer to **<Trace>.ListNesting** for a description of the parameters and options.

See also

■ CTS

▲ 'Release Information' in 'Legacy Release History'

CTS.Mode

Operation mode

Format:	**CTS.Mode** [**Full** | **Memory** | **CACHE**]

Full (default)	The trace contains the full program and data flow information.
Memory	The trace contains only data flow information, a selective trace on specific data accesses was performed. CTS can reconstruct the memory contents only. CTS is used here e.g. to reconstruct the contents of several HLL variables or task control block information.
CACHE	Reconstruct the contents of caches and TBLs (only required if a cache analysis is performed).

See also

■ CTS ■ CTS.state

CTS.OFF — Switch off trace-based debugging

Format:	**CTS.OFF**

Trace-based debugging is switch to off. The current context of the target system is re-displayed on the TRACE32 screen.

See also

■ CTS ■ CTS.state

CTS.ON — Switch on trace-based debugging

Format:	**CTS.ON**

Switches trace-based debugging to ON. The starting point is either 0./1. or the last selected record.

Use **CTS.GOTO** to switch CTS to ON with at specific starting point.

See also

■ CTS ■ CTS.state

▲ 'Release Information' in 'Legacy Release History'

CTS.PROCESS — Process cache analysis

Format:	**CTS.PROCESS [/FILE]**

Switches CTS to ON and calculates the results for the cache analysis by processing the complete trace contents.

See also

■ CTS ■ CTS.state

CTS.PROfileChart

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

Displays distributions versus time graphically based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

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

See also

■ CTS ■ CTS.Chart.sYmbol ■ CTS.STATistic

▲ 'Release Information' in 'Legacy Release History'

CTS.PROfileChart.CACHE Display cache analysis results graphically

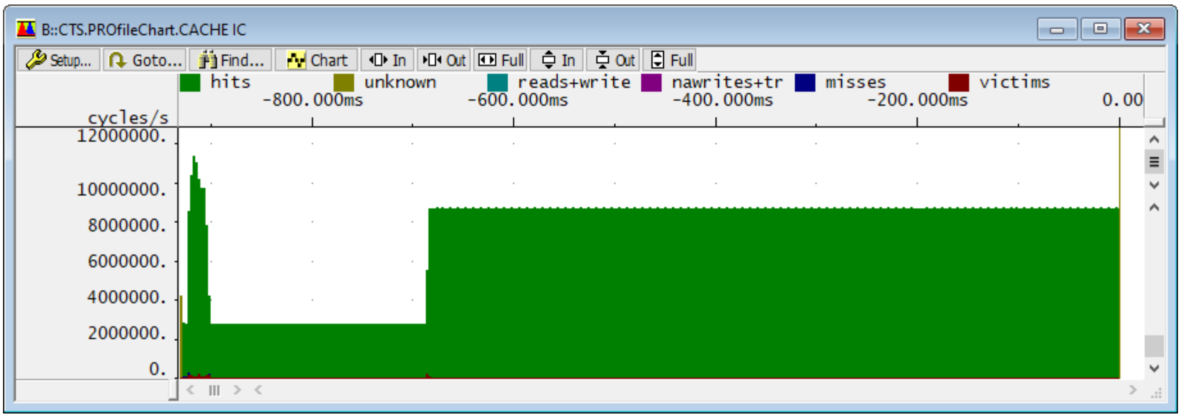
Format:	**CTS.PROfileChart.CACHE** *<cache>* [*<trace_area>*] [/*<option>*]
<cache>	**IC** | **DC** | **L2** | **L3** | **STALLS** | **BUS0** | **BUS1** | **BUS2** | **BUS3** | **MIPS** | **BTAC** | **STATIC** | **RSTACK**
<option>	**FILE** **FlowTrace** | **BusTrace** **ReScale** | **TimeScale** | **TimeZero** | **TimeREF** **Vector** | **Steps** **Track** | **ZoomTrack**

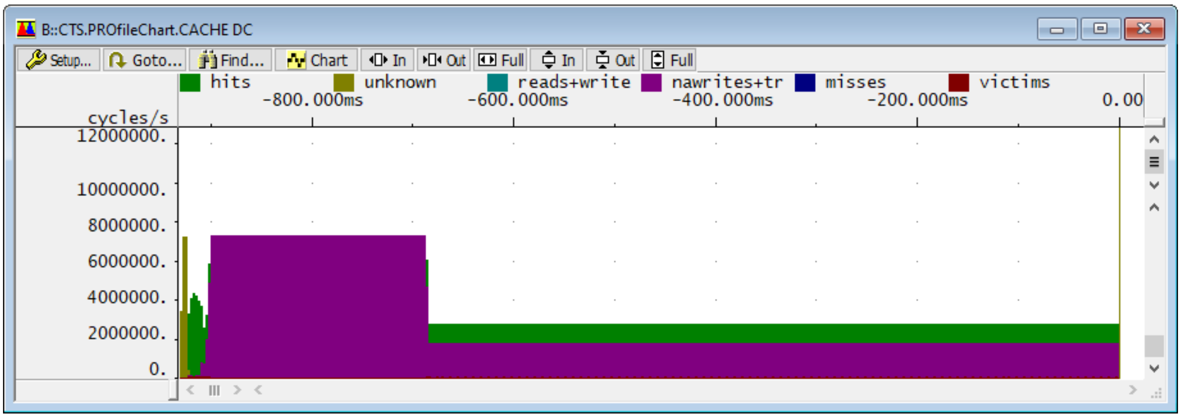
Displays the results of the CTS cache analysis as profile chart.

<trace_area> *<option>*	Refer to **<trace>.PROfileChart**

Example:

```
CTC.Mode CACHE
CTS.ON
CTS.PROFileChart DC
CTS.PROFileChart IC
```





See also

- CTS.CACHE

CTS.PROfileChart.sYmbol — Dynamic program behavior as profile chart

Format: **CTS.PROfileChart.sYmbol** [<*trace_area*>] [/<*option*>]

Displays the dynamic program behavior versus time graphically based on the CTS data.

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

See also

- <trace>.PROfileChart.sYmbol

CTS.PROfileChart.TASK — Task profile chart

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

Displays the dynamic task behavior versus time graphically based on the CTS data. This feature is only available if TRACE32 has been set for OS-aware debugging.

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

See also

- <trace>.PROfileChart.TASK

CTS.PROfileChart.TASKINFO — Profile chart for context ID special messages

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

Displays a graphical profile chart based on the CTS data for special messages written to the context ID register (ETM trace).

Refer to **<trace>.PROfileChart.TASKINFO** for a description of the parameters and options.

See also

- <trace>.PROfileChart.TASKINFO

CTS.PROfileChart.TASKINTR — ISR2 profile chart

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

Displays the dynamic behavior of ORTI based ISR2 versus time graphically based on the CTS data. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **“OS Awareness Manual NORTi”** (rtos_norti.pdf) for more information.

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

See also

- <trace>.PROfileChart.TASKINTR

CTS.PROfileChart.TASKKernel — Task profile chart with kernel markers

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

Similar command to **<trace>.PROfileChart.TASKKernel**. The analysis is however based on the CTS data. This feature is only available if TRACE32 has been set for OS-aware debugging.

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

See also

- <trace>.PROfileChart.TASKKernel

CTS.PROfileChart.TASKORINTERRUPT — Task and interrupt profile chart

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

Displays the dynamic behavior of tasks and interrupts versus time graphically based on the CTS data. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to **<trace>.PROfileChart.TASKORINTERRUPT** for a description of the parameters and options.

See also

- <trace>.PROfileChart.TASKORINTERRUPT

CTS.PROfileChart.TASKSRV — OS service routines profile chart

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

Displays the dynamic behavior of OS service routines versus time graphically based on the CTS data. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **“OS Awareness Manual NORTi”** (rtos_norti.pdf) for more information.

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

See also

- <trace>.PROfileChart.TASKSRV

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

Displays the dynamic behavior of task-related interrupts versus time graphically based on the CTS data. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual NORTi"** (rtos_norti.pdf) for more information.

Refer to **<trace>.PROfileChart.TASKVSINTR** for a description of the parameters and options.

See also

- <trace>.PROfileChart.TASKVSINTR

CTS.RESet — Reset the CTS settings

Format:	**CTS.RESet**

Resets the CTS setting and switch trace based debugging to off.

See also

- CTS
- CTS.state

CTS.SELectiveTrace — Trace contains selective trace information

Format:	**CTS.SELectiveTrace** [**ON** | **OFF**]

ON	A selective trace was performed, so the trace buffer does not contain the complete program and data flow. The sampling to the trace buffer is either controlled by the development tool or by the processor. In this case CTS clears the register and memory context after each discontinuance of the program/data flow. It is recommended to switch **CTS.UseMemory** to OFF (not supported for all CPUs).
OFF (default)	The trace contains the relevant program and data flow.

See also

- CTS

CTS.SKIP — Select the specified record for CTS (relative)

Format:	**CTS.SKIP** *<delta>* [**/FILE**]

Selects a specific record for CTS relative to the currently selected record.

```
CTS.SKIP 20.
```

See also

- CTS
- CTS.state

Format:	**CTS.SmartTrace** [**ON** | **OFF**]

Enables/disables CTS **SmartTrace**. When **SmartTrace** is enabled, all CTS commands as **CTS.List** and **CTS.Chart** will fill gaps in the trace caused by FIFO overflows.

Only supported for the following architectures:

- PowerPC MPC5xx Nexus
- PowerPC MPC5xxx Nexus
- ARM ETMv3
- MCORE Nexus
- StarCore Nexus

SmartTrace is an algorithm developed by LAUTERBACH. It allows to offset trace data loss caused by a FIFO OVERFLOW under certain circumstances. SmartTrace investigates whether there is a clear path from address A to address B via direct branches that can be reached in the calculated number of clock cycles with the instructions used. If a clear path exists the lost trace data can be reconstructed.

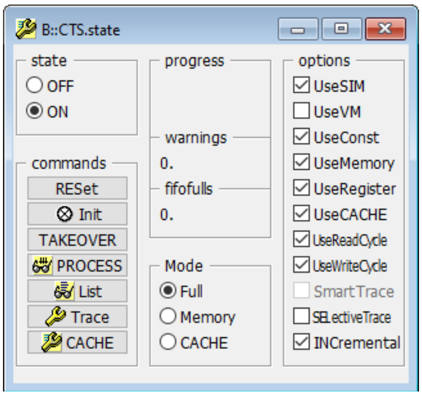
See also

■ CTS ■ CTS.state

▲ 'Release Information' in 'Legacy Release History'

Format:	**CTS.state** [*<address>* | *<range>*]

Displays the CTS settings.



state: OFF [✓ ON]

progress

warnings: 0.

fifofulls: 0.

Mode: [✓ Full] Memory CACHE

options:
- [✓] UseSIM
- [ ] UseVM
- [✓] UseConst
- [✓] UseMemory
- [✓] UseRegister
- [✓] UseCACHE
- [✓] UseReadCycle
- [✓] UseWriteCycle
- [ ] SmartTrace
- [ ] SELectiveTrace
- [✓] INCremental

The settings below are recommended in case:

- the program execution is still running while CTS is used
- or not all CPU cycles until the stop of the program execution are sampled to the trace buffer

In both cases the current state of the target can not be used by CTS.

```
CTS.UseMemory OFF                     ; don't use the current state of
                                      ; the target memory for CTS

CTS.UseRegister OFF                   ; don't use the current state of
                                      ; the CPU register for CTS

MAP.CONST                             ; attribute the constant section
sYmbol.SECRANGE(\.sdata2)

Data.COPY                             ; copy contents of constant section
sYmbol.SECRANGE(\.sdata2) VM:         ; to the virtual memory. This
                                      ; allows CTS to use this memory
                                      ; contents even when the program
                                      ; execution is running

CTS.UseConst ON                       ; read accesses to all memory
                                      ; locations with the attribute
                                      ; CONST are used by CTS
```

Recommended settings for selective trace on data:

```
CTS.Mode Memory                       ; CTS reconstructs only the memory
```

Recommended settings if a selective trace is used:

```
CTS.SELectiveTrace ON                 ; Clear memory and register context
                                      ; at each discontinuance of the
                                      ; program/data flow
```

The following settings are only necessary if the not sampled parts of the program/data flow change the memory or register contents.

```
CTS.UseMemory OFF                     ; CTS doesn't use the current
                                      ; memory

CTS.UseRegister OFF                   ; CTS doesn't use the current CPU
                                      ; registers
```

Recommended settings if only the program flow is sampled to the trace buffer:

```
CTS.UseMemory OFF                     ; CTS doesn't use the current
                                      ; memory
```

See also

■ CTS	■ CTS.CACHE	■ CTS.CACHE.state	■ CTS.CAPTURE
■ CTS.Chart.sYmbol	■ CTS.GOTO	■ CTS.INCremental	■ CTS.Init
■ CTS.List	■ CTS.Mode	■ CTS.OFF	■ CTS.ON
■ CTS.PROCESS	■ CTS.RESet	■ CTS.SKIP	■ CTS.SmartTrace
■ CTS.TAKEOVER	■ CTS.UseCACHE	■ CTS.UseConst	■ CTS.UseMemory
■ CTS.UseReadCycle	■ CTS.UseRegister	■ CTS.UseSIM	■ CTS.UseVM
■ CTS.UseWriteCycle	■ Go.BackTillWarning	■ Go.TillWarning	

CTS.STATistic — Nesting function runtime analysis

The **CTS.STATistic** command group displays a statistical analysis based on the CTS data. Gaps in the trace caused by FIFO overflows are filled by CTS when SmartTrace is enabled (**CTS.SmartTrace ON**).

See also

- CTS
- CTS.PROfileChart

CTS.STATistic.ChildTREE — Show callee context of a function

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

Show call tree and run-time of all functions called by the specified function based on the CTS data. The function is specified by its start *<address>*.

Refer to the description of **<trace>.STATistic.ChildTREE** for more information.

See also

- <trace>.STATistic.TREE

CTS.STATistic.Func — Nesting function runtime analysis

Format: **CTS.STATistic.Func** [*<trace_area>*] [/*<option>*]

Analyzes the function nesting and calculates the time spent in functions and the number of function calls based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace.

Refer to the description of **<trace>.STATistic.Func** for more information.

See also

- <trace>.STATistic.Func

CTS.STATistic.GROUP Group run-time analysis

Format:	**CTS.STATistic.GROUP** [*<trace_area>*] [/*<option>*]

The time spent in **groups** and the number of calls is calculated (flat statistic) based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. The results only include groups within the program range. Groups for data addresses are not included.

Refer to the description of **<trace>.STATistic.GROUP** for more information.

See also

- <trace>.STATistic.GROUP

CTS.STATistic.INTERRUPT Interrupt statistic

Format:	**CTS.STATistic.INTERRUPT** [*<trace_area>*] [/*<option>*]

Analyzes the function nesting and calculates the time spent in interrupts and the number of interrupt calls based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to the description of **<trace>.STATistic.INTERRUPT** for more information.

See also

- <trace>.STATistic.INTERRUPT

CTS.STATistic.INTERRUPTTREE Interrupt nesting

Format:	**CTS.STATistic.INTERRUPTTREE** [*<trace_area>*] [/*<option>*]

This command displays a graphical tree of the interrupt nesting based on the CTS data. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to the description of **<trace>.STATistic.INTERRUPTTREE** for more information.

See also

- <trace>.STATistic.INTERRUPTTREE

CTS.STATistic.LINKage — Per caller statistic of function

Format:	**CTS.STATistic.LINKage** [<*trace_area*>] [/<*option*>]

Performs a function run-time statistic for a single function itemized by its callers based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace.

Refer to the description of **<trace>.STATistic.LINKage** for more information.

See also

■ <trace>.STATistic.LINKage

CTS.STATistic.MODULE — Code execution broken down by module

Format:	**CTS.STATistic.MODULE** [<*trace_area*>] [/<*option*>]

Shows a statistical analysis of symbol modules based on the CTS data. The list of loaded modules can be displayed with **sYmbol.List.Module**. CTS tries to fill gaps in the trace using SmartTrace.

Refer to the description of **<trace>.STATistic.MODULE** for more information.

See also

■ <trace>.STATistic.MODULE

CTS.STATistic.ParentTREE — Show the call context of a function

Format:	**CTS.STATistic.ParentTREE** [<*trace_area*>] [/<*option*>]

Show call tree and run-time of all callers of the specified function based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace.

Refer to the description of **<trace>.STATistic.ParentTREE** for more information.

See also

■ <trace>.STATistic.ParentTREE

CTS.STATistic.PROGRAM — Code execution broken down by program

Format:	**CTS.STATistic.PROGRAM** [<*trace_area*>] [/<*option*>]

Shows a statistical analysis of loaded object file programs based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. The loaded programs can be displayed with the command **sYmbol.Browse \\***.

Refer to the description of **<trace>.STATistic.PROGRAM** for more information.

See also

- <trace>.STATistic.PROGRAM

CTS.STATistic.RUNNABLE — Runnable runtime analysis

Format:	**CTS.STATistic.RUNNABLE** [<*trace_area*>] [/<*option*>]

Analyzes the function nesting and calculates the time spent in AUTOSAR Runnables and the number of Runnable calls based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **“OS Awareness Manual NORTi”** (rtos_norti.pdf) for more information.

Refer to the description of **<trace>.STATistic.RUNNABLE** for more information.

See also

- <trace>.STATistic.RUNNABLE

CTS.STATistic.sYmbol — Flat run-time analysis

Format:	**CTS.STATistic.sYmbol** [<*trace_area*>] [/<*option*>]

Displays the execution time in different symbol regions based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace.

Refer to the description of **<trace>.STATistic.sYmbol** for more information.

See also

- <trace>.STATistic.sYmbol

CTS.STATistic.TASK Task statistic

Format: **CTS.STATistic.TASK** [<*trace_area*>] [/<*option*>]

Displays a task runtime statistic based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to the description of **<trace>.STATistic.TASK** for more information.

See also

■ <trace>.STATistic.TASK

CTS.STATistic.TASKINFO Statistic for context ID special messages

Format: **CTS.STATistic.TASKINFO** [<*trace_area*>] [/<*option*>]

Displays a run-time statistic based on the CTS data for special messages written to the context ID register (ETM trace).

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

See also

■ <trace>.STATistic.TASKINFO

CTS.STATistic.TASKINTR ISR2 statistic (ORTI)

Format: **CTS.STATistic.TASKINTR** [<*trace_area*>] [/<*option*>]

Displays an ORTI based ISR2 runtime statistic based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **"OS Awareness Manual NORTi"** (rtos_norti.pdf) for more information.

Refer to the description of **<trace>.STATistic.TASKINTR** for more information.

See also

■ <trace>.STATistic.TASKINTR

CTS.STATistic.TASKKernel Task statistic with kernel markers

Format:	**CTS.STATistic.TASK** [<*trace_area*>] [/<*option*>]

Similar command to **<trace>.STATistic.TASKKernel**. The analysis is however based on the CTS data. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to the description of **<trace>.STATistic.TASKKernel** for more information.

See also

- <trace>.STATistic.TASKKernel

CTS.STATistic.TASKORINTERRUPT Task and interrupt statistic

Format:	**CTS.STATistic.TASKORINTERRUPT** [<*trace_area*>] [/<*option*>]

Displays the execution time in different tasks and interrupts based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to the description of **<trace>.STATistic.TASKORINTERRUPT** for more information.

See also

- <trace>.STATistic.TASKORINTERRUPT

CTS.STATistic.TASKSRV OS service routines statistic

Format:	**CTS.STATistic.TASKSRV** [<*trace_area*>] [/<*option*>]

Displays the execution time in OS service routines based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **“OS Awareness Manual NORTi”** (rtos_norti.pdf) for more information.

Refer to the description of **<trace>.STATistic.TASKSRV** for more information.

See also

■ <trace>.STATistic.TASKSRV

CTS.STATistic.TASKVSINTERRUPT Statistic of interrupts, task-related

Format:	**CTS.STATistic.TASKVSINTERRUPT** [*<trace_area>*] [/*<option>*]

Displays the execution time in task-related interrupts based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace. This feature is only available if TRACE32 has been set for OS-aware debugging.

Refer to the description of **<trace>.STATistic.TASKVSINTERRUPT** for more information.

See also

■ <trace>.STATistic.TASKVSINTERRUPT

CTS.STATistic.TREE Tree display of nesting function run-time analysis

Format:	**CTS.STATistic.TREE** [*<trace_area>*] [/*<option>*]

The results of this command shows a graphical tree of the function nesting based on the CTS data. CTS tries to fill gaps in the trace using SmartTrace.

Refer to the description of **<trace>.STATistic.TREE** for more information.

See also

■ <trace>.STATistic.TREE

▲ 'Release Information' in 'Legacy Release History'

CTS.TAKEOVER Take memory/registers reconstructed by CTS over to target

Format:	**CTS.TAKEOVER**

If CTS is active, the TRACE32 screen displays the contents of the registers and memories as they have been when the currently active CTS record (see the yellow CTS field in the state line) was sampled to the state line. The command **CTS.TAKEOVER** takes the register and memory contents over to the target and deactivates CTS.

See also

■ CTS ■ CTS.state

CTS.UNDO Revert last CTS command

Format:	**CTS.UNDO**

Undoes last CTS run-control command (e.g CTS Step).

See also

■ CTS.UseConst ■ CTS.UseMemory ■ CTS.UseReadCycle ■ CTS.UseRegister
■ CTS.UseSIM ■ CTS.UseWriteCycle ■ CTS

Format:	**CTS.UseCache** [**ON** | **OFF**]

This command is typically used for short trace recordings to minimize the number of unknown cycles. It uses the current cache contents for cache analysis. Together with **CTS.CAPTURE** it will also use the initial cache contents.

NOTE:	Not all processors support debug access to the cache.

ON	The cache is included in the analysis.
OFF	The cache is excluded from the analysis. Information that is not analyzed is labeled “unknown” on the TRACE PowerView GUI. In addition, the percentage of unknown information is displayed.

Example:

```
...
; It is recommended to make this setting very early on in a script.
CTS.UseCache ON
...
; -------------------------------------------------------------------
; 1st analysis: Capture a snapshot of the system for the analysis.
CTS.Capture
Go
Break
...

; -------------------------------------------------------------------
; 2nd analysis
CTS.Capture
Go
Break
...
```

See also

- CTS.UseConst
- CTS.UseMemory
- CTS.UseReadCycle
- CTS.UseRegister
- CTS.UseSIM
- CTS.UseWriteCycle
- CTS
- CTS.CAPTURE
- CTS.state

Format:	**CTS.UseConst** [**ON** | **OFF**]

CTS.UseConst become effective after **CTS.UseMemory** is set to OFF.

ON	Read accesses to all memory locations that have the mapper attribute CONST are evaluated by CTS even if **CTS.UseMemory** is switched to **OFF**.
OFF	Memory locations with the attribute CONST are not used by CTS.

See also

■ CTS.UseCACHE	■ CTS.UNDO	■ CTS.UseMemory	■ CTS.UseReadCycle
■ CTS.UseRegister	■ CTS.UseSIM	■ CTS.UseVM	■ CTS.UseWriteCycle
■ CTS	■ CTS.state		

Format:	**CTS.UseMemory** [**ON**	**OFF**]

ON (default)	The memory contents is used by CTS. • When a memory location was not accessed by the program section sampled to the trace buffer, CTS assumes, that the memory location had the current contents during all program steps. • When there was no write access to a memory location by the program section sampled to the trace buffer, CTS assumes, that the current contents was read by read accesses to this memory location sampled to the trace buffer. To set **CTS.UseMemory** to ON requires, that all CPU cycles until the stop of the program execution were sampled to the trace buffer. Memory ranges that are changed not only by the CPU core e.g. peripherals or dual-ported memories can be excluded by using the **MAP.VOLATILE** command
OFF	**CTS.UseMemory OFF** is required: • if not all CPU cycles until the stop of the program execution were sampled to the trace buffer. • if the program execution is still running while CTS is used. • if no data flow is sampled to the trace buffer. **MAP.CONST** can be used to define memory ranges with constant contents that are used by CTS if **CTS.UseConst** is set to **ON**.

See also

■ CTS.UNDO	■ CTS.UseCACHE	■ CTS.UseConst	■ CTS.UseReadCycle
■ CTS.UseRegister	■ CTS.UseSIM	■ CTS.UseVM	■ CTS.UseWriteCycle
■ CTS	■ CTS.state	■ MAP.VOLATILE	

▲ 'Release Information' in 'Legacy Release History'

CTS.UseReadCycle — Use read cycles for CTS

Format:	**CTS.UseReadCycle** [**ON** | **OFF**]

ON (default)	CTS uses the read cycles sampled to the trace buffer.
OFF	CTS doesn't use the read cycles sampled to the trace buffer. This is required if a bus trace is used and the processor is using a write through cache.

See also

- ■ CTS.UseRegister
- ■ CTS.UseMemory
- ■ CTS
- ■ CTS.UNDO
- ■ CTS.UseSIM
- ■ CTS.state
- ■ CTS.UseCACHE
- ■ CTS.UseVM
- ■ CTS.UseConst
- ■ CTS.UseWriteCycle

CTS.UseRegister — Use the CPU registers for CTS

Format:	**CTS.UseRegister** [**ON** | **OFF**]

ON (default)	CTS uses the current contents of the CPU registers. When a CPU register was not accessed by the program section sampled to the trace buffer, CTS assumes, that the register had the current contents during all program steps.
OFF	CTS doesn't use the current contents of the CPU registers. This is required if the program execution is still running when CTS is used or if the program execution was still running after the sampling to the trace buffer was stopped.

See also

- ■ CTS.UseReadCycle
- ■ CTS.UseMemory
- ■ CTS
- ■ CTS.UNDO
- ■ CTS.UseSIM
- ■ CTS.state
- ■ CTS.UseCACHE
- ■ CTS.UseVM
- ■ CTS.UseConst
- ■ CTS.UseWriteCycle

▲ 'Release Information' in 'Legacy Release History'

CTS.UseSIM

Use instruction set simulator for CTS

Format:	**CTS.UseSIM** [**ON** | **OFF**]

ON (default)	CTS uses the instruction set simulator.
OFF	(For error diagnosis only.)

See also

■ CTS.UNDO	■ CTS.UseCACHE	■ CTS.UseConst	■ CTS.UseMemory
■ CTS.UseReadCycle	■ CTS.UseRegister	■ CTS.UseVM	■ CTS.UseWriteCycle
■ CTS	■ CTS.state		

Format:	**CTS.UseVM** [**ON** | **OFF**]

This command is typically used for short trace recordings to minimize the number of unknown cycles. It allows you to use the virtual memory contents as initial values for CTS. When you use the command, make sure that the trace recording contains the program start.

ON	The virtual memory contents (VM:) are used as initial values for CTS. This allows you to have valid memory contents even for the first record.
OFF	The virtual memory contents are *not* used.

```
...
; It is recommended to make this setting very early on in a script.
CTS.UseVM ON
...
; ------------------------------------------------------------------
; For the 1st analysis:

; Before the trace is started, data can be copied to the virtual memory
; (VM:) of TRACE32.
; Copy contents of specified address range to TRACE32 virtual memory.
Data.Copy 0x3fa000++0xfff VM:
; Start the trace recording and completely fill the trace buffer.
Go
Break
...
; ------------------------------------------------------------------
; For the 2nd analysis:

; Repeat the above Data.Copy command.
CTS.CAPTURE
; Start the trace recording and completely fill the trace buffer.
Go
Break
...
```

See also

- CTS.UseConst
- CTS.UseMemory
- CTS.UseReadCycle
- CTS.UseRegister
- CTS.UseSIM
- CTS.UseWriteCycle
- CTS
- CTS.CAPTURE
- CTS.state

Format:	**CTS.UseWriteCycle** [**ON** | **OFF**]

ON	CTS uses the write cycles sampled to the trace buffer.
OFF	CTS doesn't use the write cycles samples to the trace buffer. This is required if a bus trace is used and the processor is using a copy back cache.

See also

■ CTS.UNDO	■ CTS.UseCACHE	■ CTS.UseConst	■ CTS.UseMemory
■ CTS.UseReadCycle	■ CTS.UseRegister	■ CTS.UseSIM	■ CTS.UseVM
■ CTS	■ CTS.state		