LAUTERBACH
DEVELOPMENT TOOLS

System Trace User’s Guide

System Trace User’s Guide

System Trace User's Guide

Version 06-Jun-2024

History

06-Nov-2023	New command SYStem.CONFIG.CMN.DEViceName.
31-Oct-2022	New command CMN.CycleAccurate.
29-Sept-2022	New command CMN.TimeStampCLOCK.
31-Aug-2022	New commands CMN.MPAM and CMN.VERsion.
Sept-2021	Added possible values for SYStem.CONFIG.CMN.XYBits command to support CMN-700.

Introduction

Generally speaking a system trace is a hardware module on a SoC which enables the developer to output predefined hardware or software messages without affecting the run-time behavior of the system.

This manual covers the following system trace implementations:

1. The System Debug Trace Interface (SDTI) by Texas Instruments used in OMAP34xx devices
2. The System Trace Module (STM) by Texas Instruments used in OMAP44xx devices
3. The System Trace Macrocell (STM) by ARM as a CoreSight component

Due to the various implementations some commands and setup routines apply to a certain type of system trace only. While setup routines and implementation specific commands will be handled in separate sections (TI specific or CoreSight specific), some common commands differ in the number of available arguments or in the meaning of the arguments. These differences will be marked as follows:
• SDTI (TI) for Texas Instruments' SDTI implementation.
• STM (TI) for Texas Instruments' STM implementation.
• STM (CS) for the CoreSight implementation.
Arguments not available for a specific implementation will be marked as 'n.a.' - not applicable.

Another difference between those implementations is the trace protocol: SDTI (TI) outputs data in XTIv2 format, STM (TI) in STPv1, and STM (CS) in STPv2.

To simplify matters the term "STP = System Trace Protocol" will be used in the following.

STM (TI) and STM (CS) in turn offer the opportunity to route trace data to an Embedded Trace Buffer (ETB, also a CoreSight component), while SDTI (TI) does not. Reading from the ETB only requires an ARM debugger, no trace hardware modules like CombiProbe or PowerTrace. All sections/commands referring to that ETB will contain the word 'onchip' in any way.

The second way of exporting STP data is a dedicated trace port. For STM (CS) this trace port is called 'Trace Port Interface Unit, TPIU' (again a CoreSight component), for STM (TI) and SDTI (TI) this trace port is called 'Parallel Trace Interface, PTI'. In the following, the general term "trace port" will be used for both interfaces.

Terminology

This section describes the usage of the terms *component* and *module* in this manual.

Component	The term *component* is used as an umbrella term for anything you can configure using (a) the **SYStem.CONFIG.state /COmponents** window or (b) a command group specifically designed for a component. **Example**: The **STM** command group for the **STM** component. A component's actual function on a SoC can be characterized as: • Trace source or • Trace sink or • Funnel • etc. **Example**: The STM component you configure in TRACE32 is an STM trace source on the SoC.
Module	The term *module* is primarily reserved for the hardware modules of TRACE32, such as PowerTrace, PowerDebug, CombiProbe.

Preconditions

This manual assumes that the **In-Circuit Debugger is already installed**. You should be familiar with the features of the debugger. If you are not yet familiar with the debugger, refer to the **"Training Simulator and Demo Software"** (demo.pdf) and **"TRACE32 Installation Guide"** (installation.pdf).

Purpose of this Manual

The purpose of this manual is to get your trace running, to write a PRACTICE script (*.cmm) that does the necessary start-up procedure and to make you familiar with the main features of the trace. All list of all commands that are specific for the TRACE32-ICD trace for the C166 family can be found at the end of this manual.

Command Syntax

The TRACE32 commands are not case sensitive. In this tutorial, we use upper case letters for the characters that are necessary for the short form of the command entry. E.g. `Analyzer.List` can be shortened to `A.L.`

Where can I get more information?

TRACE32 provides a detailed online help offering the most current description of all debug features.

1. In TRACE32 choose **Help** menu > **Contents**.

2. See also **Online Help** for a brief overview of the online help.

Installation

Software Installation

The TRACE32-ICD software for the ARM debugger also includes the STM trace support. No extra software installation for the STM trace is required.

Hardware Installation

CombiProbe-ARM (LA-4502)

1. Simply attach the CombiProbe to your debugger.
2. Plug the header into the target's trace connector (or target adaption, if required).

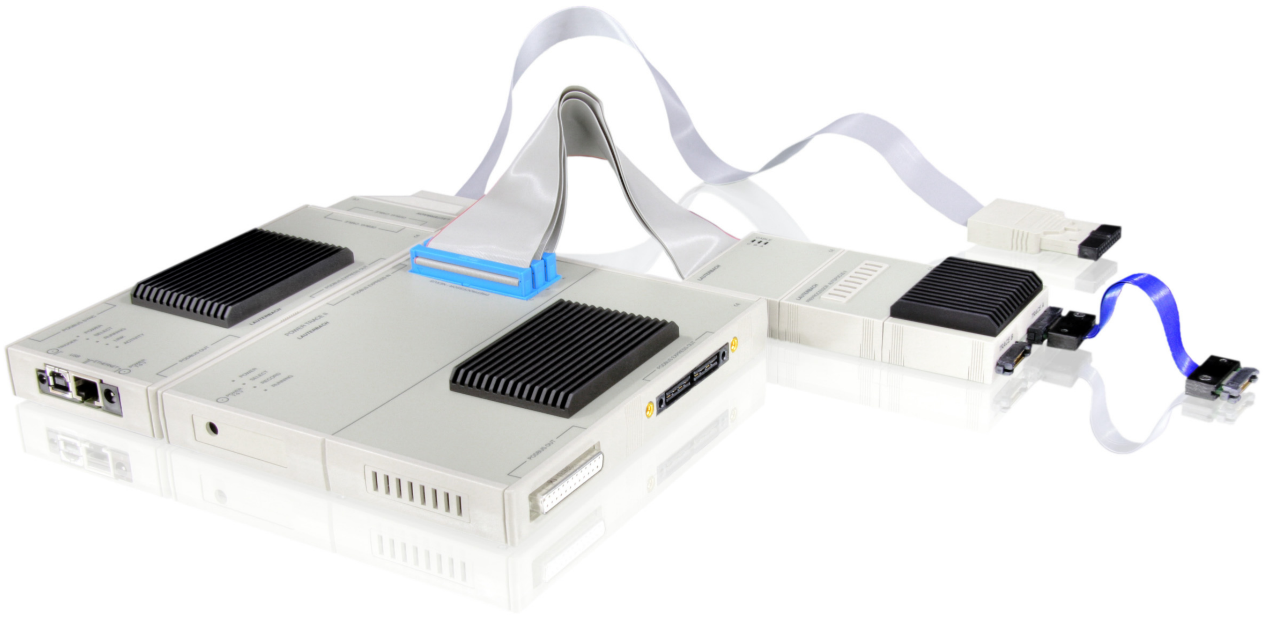
1. Attach the debug cable to the debugger.

2. Connect the ’PODBUS EXPRESS OUT’ port of the debugger to the “PODBUS EXPRESS IN“ port of PowerTrace II.

3. Plug the preprocessor’s flat cables into the according connectors of PowerTrace II: The shortest cable to the connector labelled ’A’, the middle to connector ’B’ and the longest to connector ’C’.

4. Connect the debug cable header to the target’s JTAG port (or target adaption, if required).

5. Connect the preprocessor’s MICTOR connector (labelled ’TRACE A’) to the target’s trace port (or target adaption, if required).



Start-up Script

Example STP Data to Onchip Buffer

Target: OMAP4430

NOTE:	This example applies to the STM by Texas Instruments only.

```
; Clock definition
Trace.CLOCK 100MHz                        ; Optional: If not defined,
                                          ; only raw timestamps will
                                          ; be displayed.

STM.RESet

; Route STP data to ETB
STM.PortRoute ONCHIP                      ; If ETM was on, it will be
                                          ; disabled here.

; !!!!!!!! Important !!!!!!!!             ; If AutoArm remain on, several
Onchip.AutoArm OFF                        ; final trace bytes will be
                                          ; missing.

; Turn on STM component
STM.ON

; Manually enable the ETB
Onchip.Arm

...

; Don’t forget to disable the ETB
afterwards!
Onchip.OFF
```

Example STP Data to Parallel Trace Interface

Target: OMAP4430

NOTE:	This example applies to the STM by Texas Instruments only.

```
; Define STP data format
SYStem.CONFIG.STM STP                ; This will unlock the STM
                                     ; commands.

STM.RESet

; Route STP data to PTI
STM.PortRoute CAnalyzer


; Pad configuration                  ; Multiplex emu[0:4] signals to
                                     ; dpm_emu[0:4] pads or multiplex
                                     ; emu[15:19] signals to dpm_emu[15:19]
                                     ; pads. You can even do both.

; dpm_emu[0:4]
Data.Set ahb:0x4a1001ac %long    Data.Long(ahb:0x4a1001ac)&0xffff
Data.Set ahb:0x4a1001b0 %long    0
Data.Set ahb:0x4a1001b4 %long    0


; dpm_emu[15:19]
Data.Set ahb:0x4a1001cc %long    0
Data.Set ahb:0x4a1001d0 %long    0
Data.Set ahb:0x4a1001d4 %long    0

; Configure & init CombiProbe
CAnalyzer.THreshold 0.9
CAnalyzer.Init

; Turn on STM component
STM.ON
```

Example Attach to Onchip Trace

NOTE:	This example applies to CoreSight compliant STMs only.

```
; Setup target
SYStem.CPU <cpu>
SYStem.CONFIG <config>
...                                 ; Do not attach to the target yet!

SystemTrace.Method Onchip

Onchip.TraceConnect <buffer>

SYStem.Attach

Onchip.Disable

Onchip.Attach

SystemTrace.List
```

Example Save/Load STP Data Embedded in CoreSight Trace Stream

NOTE:	This following examples assume that the target has been set up properly beforehand.

Save file:

```
Trace.Export.TracePort <file>    Save in binary file format
```

Load file:

```
TPIU.PortMode WRAPPED

CoreSightTrace.METHOD LA

SystemTrace.METHOD LA

LA.IMPORT.TracePort <file>

LA.IMPORT.StartValid

SystemTrace.List
```

SYStem.CONFIG.STM — Inform TRACE32 about STM component

Format:	**SYStem.CONFIG.STM**[*<instance>*] *<sub_cmd>*
<instance>:	**1** | **2**
<sub_cmd>:	*<generic>* | *<component_specific>*
<component_ specific>:	**ACCESS** [**Denied** | **ReadWrite**] **Type** *<type>* **Mode** *<mode>* **AutoSync** [**ON** | **OFF**]

Default: Device specific.

Provides essential information about the STM (manufacturer, protocol, etc.) to TRACE32. Usually this step is already included in the CPU selection.

<instance>	For a description of *<instance>*, refer to the introduction to the command group **STM**.
<generic>	For descriptions of the generic subcommands, click here.
STM	Single STM. If the chip contains more than one STM, the individual STM can be addressed by adding a number to the keyword **STM**, i.e. **STM1** or **STM2**.
STM1	Same as STM command. Used to differentiate between **STM1** and **STM2**.
STM2	Used to configure a 2nd STM, if present.
ACCESS [**Denied** | **ReadWrite**]	Only if **SYStem.CONFIG.STM.Type** is set to **ARM**. Set this property to **Denied** if TRACE32 is not supposed to write to the configuration registers of an STM. Default: ReadWrite.
AutoSync [**ON** | **OFF**]	If **ON**, TRACE32 tries to synchronize to the trace stream even if no synchronization packets can be found. This setting only has an effect for STP version >= 2. Default: ON.

Mode *<mode>*	For details, see **SYStem.CONFIG.STM.Mode**.
Type *<type>*	For details, see **SYStem.CONFIG.STM.Type**.

Example 1:

```
SYStem.CONFIG STM Mode STP64            ; chip contains a STM that uses
                                        ; MIPI STPv1 (D64) protocol

STM.state                               ; open STM configuration window
```

Example 2:

```
SYStem.CONFIG STM1 Mode STPv2           ; chip contains a STM that uses
                                        ; MIPI STPv2 protocol

STM1.state                              ; open STM1 configuration window

SYStem.CONFIG STM2 Mode STPv2           ; chip contains a second STM that
                                        ; uses MIPI STPv2 protocol

STM2.state                              ; open STM2 configuration window
```

See also

■ SYStem.CONFIG.STM.Mode ■ SYStem.CONFIG.STM.Type ■ SYStem.CONFIG

▲ 'Generic Subcommands, Parameters, and Options' in 'System Trace User's Guide'

Format:	**SYStem.CONFIG.STM.Mode** *<mode>*
<mode>:	**STP** | **STP64** | **STPv2** [**2** | **3** | **4**]

Default: Device specific.

Informs TRACE32 that the chip contains a System Trace Module. The TRACE32 command group **STM** will be enabled as a result.

STP	STP protocol (MIPI STPv1, D32 packets)
STP64	STP64 protocol (MIPI STPv1, D64 packets)
STPv2 [**2** | **3** | **4**]	STPv2 protocol (MIPI STPv2).

STP version 2 (STPv2) offers the possibility to output timestamps in different formats. Usually the device specific format will be set up by TRACE32 automatically during CPU selection.

The STPv2 mode allows you to set up the timestamp format manually afterwards, if necessary.

2	NATDELTA Natural binary delta timestamp; timestamp counter is reset after each timestamp packet.
3	NAT Natural binary absolute timestamp; free running timestamp counter.
4	GRAY Gray encoding of free running counter.

See also

■ SYStem.CONFIG.STM

Format:	**SYStem.CONFIG.STM.Type** *<type>*
<type>:	**None** | **Generic** | **ARM** | **SDTI** | **TI**

Configures the STM type in TRACE32.

None	No STM type is configured in TRACE32.
Generic (default)	STM component is generic.
ARM	System Trace Macrocell (STM) by ARM as a CoreSight component
SDTI	System Debug Trace Interface (SDTI) by Texas Instruments
TI	System Trace Module (STM) by Texas Instruments

See also

- SYStem.CONFIG.STM

STM

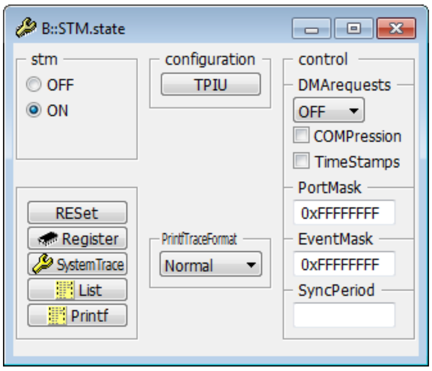
Configure STM component on target

Format:	**STM**[*<instance>*]**.***<sub_cmd>*
<instance>:	**1** | **2**

The **STM***<instance>* command group is used to configure the STM trace source.

1	Instance of the STM trace source 1. Most targets have only one STM trace source. For these targets, the commands **STM.***<sub_cmd>* and **STM1.***<sub_cmd>* are aliases. This means, if you are configuring a target with only one STM trace source, you may omit the instance number **1**.
2	Instance of the STM trace source 2. Some targets have two STM trace sources. For these targets, you must include the instance numbers **1** or **2**.
<sub_cmd>	For a description of the subcommands, refer to the command descriptions in this chapter.

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



To display and analyze the recorded trace data, use the **STM<trace>** command groups.

See also

■ STM.ChannelRepeat	■ STM.COMPression	■ STM.DMArequests	■ STM.EventMASK
■ STM.FilterChannels	■ STM.FilterMasters	■ STM.HWMasters	■ STM.IdleCount
■ STM.IgnoreHeader	■ STM.Init	■ STM.MasterRepeat	■ STM.OCPAutoIdle
■ STM.OFF	■ STM.ON	■ STM.PATTERN	■ STM.PortEndianness
■ STM.PortMASK	■ STM.PortMode	■ STM.PortRoute	■ STM.PortSize
■ STM.PrintfTraceFormat	■ STM.Register	■ STM.RESet	■ STM.SetMaster
■ STM.state	■ STM.SWMasters	■ STM.SyncPeriod	■ STM.SyncTime
■ STM.TimeStampCLOCK	■ STM.TimeStamps	■ STM.TraceID	■ STM.TracePriority
■ SystemTrace			

▲ 'Release Information' in 'Legacy Release History'
▲ 'STM Component - CoreSight specific Target Configuration' in 'System Trace User's Guide'
▲ 'STM Component - TI specific Target Configuration' in 'System Trace User's Guide'
▲ 'STM<trace> - Trace Data Analysis' in 'System Trace User's Guide'

STM.FilterMasters — Display specified masters only

Format:	**STM**[*<instance>*]**.FilterMasters** *<id1> <id2> <id3> <id4>*

Select up to four STM master IDs, which associated trace packets will be displayed in the trace results. All other STM packets will be masked out.

This command actually does not filter STM packets but only affects the display. After the filter has been reset, all STM packets will be shown. The filter is reset via **STM.FilterMasters** (without any ID specified).

See also

■ STM ■ STM.state

STM.FilterChannels — Display specified channels only

Format:	**STM**[*<instance>*]**.FilterChannels** *<id1> <id2> <id3> <id4>*

Selects up to four STM channels, which will be displayed in the trace results. All other channels will be masked out.

This command actually does not filter STM packets but only affects the display. After the filter has been reset, all STM packets will be shown. The filter is reset via **STM.FilterChannels** (without any ID specified).

See also

■ STM ■ STM.state

STM.Init — Initialize trace hardware

Format:	**STM**[<*instance*>]**.Init**

The trace hardware is initialized and set to its defaults.

See also

- ■ STM
- ■ STM.state

STM.OFF — Switch STM off

Format:	**STM**[<*instance*>]**.OFF**

Disables the STM functionality.

See also

- ■ STM
- ■ STM.state

STM.ON — Switch STM on

Format:	**STM**[<*instance*>]**.ON**

Enables the STM functionality.

See also

- ■ STM
- ■ STM.state

Format:	**STM**[*<instance>*]**.PortEndianness** [**Big** | **Little**]

Default: Big.

If **STM.PortSize** is > 8, this command determines the byte order of the trace port.

Big	MSB mapped to lower port bits.
Little	MSB mapped to upper port bits.

See also

■ STM ■ STM.state

Format:	**STM**[*<instance>*]**.PortMode** *<mode>*
<mode>:	**Continuous** **Gated** **AutoIdle** **Bypass** **Continuous** **HalfRate** **FullRate** **1/***<divisor>*

	SDTI (TI)	STM (TI)	STM (CS)
Continuous	n.a.	Port clock remains active even if no STP data are available (default).	n.a.
Gated	n.a.	Port clock is stopped if no STP data are available.	n.a.
AutoIdle	n.a.	Disables the Parallel Trace Interface (PTI) if no STP data are available (power saving).	n.a.
HalfRate	STP data are sampled on rising edge of port clock.	STP data are sampled on rising edge of port clock.	n.a.
FullRate	STP data are sampled on rising and falling edge of port clock.	STP data are sampled on rising and falling edge of port clock.	n.a.
1/*<divisor>*	The port clock rate is defined as ratio of OCP clock.	The port clock rate is defined as ratio of OCP clock.	n.a.
Bypass	n.a.	n.a.	n.a.
Wrapped	n.a.	n.a.	n.a.

For STM (CS): **ETM.PortMode**.

See also

■ STM ■ STM.state

Format: **STM**[*<instance>*]**.PortRoute** [**AUTO** | **Analyzer** | **CAnalyzer** | **Onchip**]

	SDTI (TI)	STM (TI)	STM (CS)
AUTO	Data are directed to the Parallel Trace Interface (PTI) and recorded by the attached trace hardware (default).	Data are directed to the Parallel Trace Interface (PTI) and recorded by the attached trace hardware (default).	n.a.
Analyzer	Data are directed to the Parallel Trace Interface (PTI) and recorded by the PowerTrace II / PowerTrace III.	Data are directed to the Parallel Trace Interface (PTI) and recorded by the PowerTrace II / PowerTrace III.	n.a.
CAnalyzer	Data are directed to the Parallel Trace Interface (PTI) and recorded by the CombiProbe.	Data are directed to the Parallel Trace Interface (PTI) and recorded by the CombiProbe.	n.a.
Onchip	n.a.	Data are directed to the Embedded Trace Buffer.	n.a.

For STM (CS): **ETM.PortRoute**.

See also

- STM
- STM.state

Format: **STM**[*<instance>*]**.PortSize** [**1** | **1E** | **1X** | **2** | **2E** | **2X** | **4** | **4E** | **4X** | **8** | **12** | **16**]

Default: 4

For SDTI (TI), STM (TI): Defines the number of parallel data pins of the trace port. Also the internal signal multiplexing of the Debug Resource Manager (DRM) is affected by this command. Please refer to the table below:

	no suffix (standard configuration)	**suffix 'X' (to be used with LA-xxxx)**	**suffix 'E' (to be used with LA-3812)**	**suffix 'Z'**	**suffix 'K' (to be used for Keystone devices)**
stm_clk	`emu19`	`emu2`	`emu2`	`emu0`	`emu10`
stm_data[0]	`emu18`	`emu3`	`emu0`	`emu1`	`emu0`
stm_data[1]	`emu17`	`emu4`	`emu1`	`emu2`	`emu1`
stm_data[2]	`emu16`	`emu5`	`emu3`	`emu3`	`emu2`
stm_data[3]	`emu15`	`emu6`	`emu4`	`emu4`	`emu3`

The trace signals are routed to emu signal lines only, not to the physical pads of the device! Refer to the example script of this manual of how to configure the pads!

For STM (CS): **ETM.PortSize**.

See also

- STM
- STM.state

STM.PrintfTraceFormat — Define format of printftrace style messages

Format:	**STM**[*<instance>*]**.PrintfTraceFormat** [**Normal** | **Kernel**]

Default: Normal.

Normal	String messages as described in section **Software Messages**
Kernel	Special string and FTRACE message format as described in **Software Messages**

See also

- STM
- STM.state

STM.Register — Display STM register

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

Displays the STM registers.

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

See also

- STM
- STM.state

STM.RESet

Reset STM settings

Format:	**STM**[*<instance>*].**RESet**

All STM settings are reset to their defaults.

See also

- STM
- STM.state

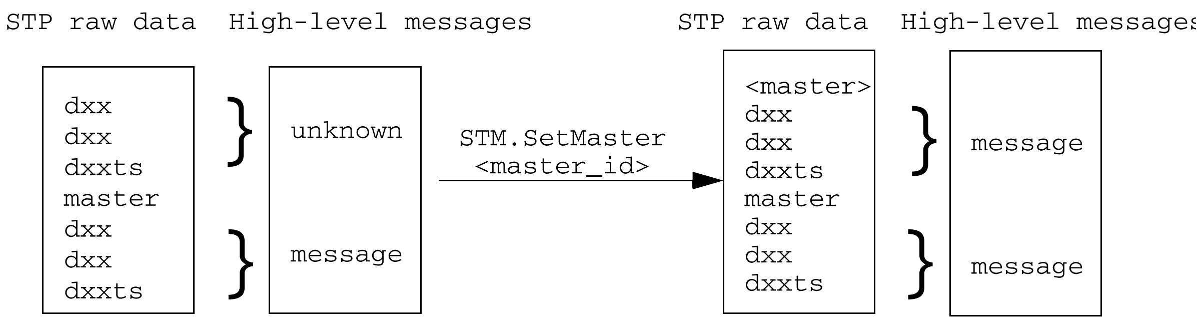
STM.SetMaster

Set master ID manually

Format:	**STM**[*<instance>*].**SetMaster** *<master_id>*

High-level STP messages from hardware modules (see **CMI**, **PMI**) or **Software Messages** must be preceded by an STP master packet in order to be decoded correctly in the according trace list window. If no master packet could be found the message will be marked as “unknown“.

However, by setting *<master_id>* manually the trace decoder assigns any unknown STP packets to the specified master until a valid STP master packet is found in the trace stream.



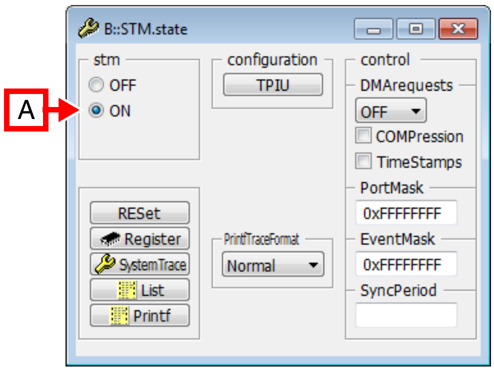
See also

- STM
- STM.state

Format:	**STM**[*<instance>*]**.state**
<instance>:	**1** | **2**

Opens the **STM.state** window, where you can configure the STM trace source.

The commands available in the window differ depending on the selected CPU. Commands that are not available for a certain CPU are hidden.



stm: OFF [✓ ON]
configuration: TPIU
control
DMArequests: OFF
☐ COMPression
☐ TimeStamps
PortMask: 0xFFFFFFFF
EventMask: 0xFFFFFFFF
SyncPeriod:
PrintfTraceFormat: Normal
RESet | Register | SystemTrace | List | Printf

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

Exceptions:

- The **SystemTrace** button opens the **SystemTrace.state** window, see **<trace>.state**. For more information, refer to the description of the **SystemTrace** command group.
- The **List** button opens the **SystemTrace.List** window, see **<trace>.List**.
- The **Printf** button opens the **PrintfTrace.List** window, see **<trace>.List**. For more information, refer to the description of the **PrintfTrace** command group.
- The **TPIU** button opens the **TPIU** window, see **TPIU.state**.

<instance>	For a description of *<instance>*, refer to the introduction to the command group **STM**.

See also

■ STM	■ STM.ChannelRepeat	■ STM.COMPression	■ STM.DMArequests
■ STM.EventMASK	■ STM.FilterChannels	■ STM.FilterMasters	■ STM.HWMasters
■ STM.IdleCount	■ STM.IgnoreHeader	■ STM.Init	■ STM.MasterRepeat
■ STM.OCPAutoIdle	■ STM.OFF	■ STM.ON	■ STM.PATTERN
■ STM.PortEndianness	■ STM.PortMASK	■ STM.PortMode	■ STM.PortRoute
■ STM.PortSize	■ STM.PrintfTraceFormat	■ STM.Register	■ STM.RESet
■ STM.SetMaster	■ STM.SWMasters	■ STM.SyncPeriod	■ STM.SyncTime
■ STM.TimeStampCLOCK	■ STM.TimeStamps	■ STM.TraceID	■ STM.TracePriority

STM.SyncTime

Trace decoder resync time

Format: **STM**[*<instance>*]**.SyncTime** *<time>*

Time after which a resync is forced in the trace decoder.

See also

■ STM ■ STM.state

STM.SyncPeriod

Add synchronization packets

Format: **STM**[*<instance>*]**.SyncPeriod** [*<value>*]

Default: 0

Inserts synchronization packets (ASYNC + VERSION) periodically into the trace stream approximately each *<value>* bytes. If *<value>* is zero, no synchronization packets will be generated.
This command is only applicable to STPv2 compliant System Trace implementations!

See also

■ STM ■ STM.state

STM.TimeStamps

Enables timestamps

Format: **STM**[*<instance>*]**.TimeStamps** [**ON** | **OFF**]

Default: OFF.

Enables or disables timestamp generation in the trace hardware.

See also

■ STM ■ STM.state

Format:	**STM**[*<instance>*]**.TimeStampCLOCK** *<frequency>*

Default: 0

Configures the debugger for the STM timestamp clock frequency of the target. The frequency is required to calculate timing information based on timestamp packets.

See also

■ STM ■ STM.state

STM.HWMasters — Enable hardware masters for tracing

Format:	**STM**[<*instance*>]**.HWMasters** <*name*> [**ON** | **OFF**]

Available <*names*> are device specific. If the corresponding hardware master is disabled, write accesses of the master to the STM will be ignored.

Default values	SDTI: N. a. STM: All off

See also

- STM
- STM.state

STM.IdleCount — Maximum idle packets

Format:	**STM**[<*instance*>]**.IdleCount** <*count*>

If there are no STP packets to be sent, <*count*> number of idle packets are emitted by the PTI. Depending on the port mode (STM.PortMode Continuous or STM.PortMode Gated), the PTI then stops or continues emitting idle packets. If the same HW master or the same SW master + channel resumes sending STP messages, a leading master packet is generated by the STM.

See also

- STM
- STM.state

STM.IgnoreHeader — Ignore leading dword in printftrace message

Format:	**STM**[*<instance>*]**.IgnoreHeader** [**ON** | **OFF**]

Default: OFF.

Newer versions of the TI CToolsLib generate a leading 32-bit word in front of the printftrace message. If not ignored, this header will produce some strange characters at the beginning of the message in the **PrintfTrace.List** window.

See also

■ STM ■ STM.state

STM.SWMasters — Enable software masters for tracing

Format:	**STM**[*<instance>*]**.SWMasters** *<name>* [**ON** | **OFF**]

Available *<names>* are device specific. If the corresponding software master is disabled, writes of that master to a stimulus port will have no effect.

Default values	SDTI: CU1 = ON, CPU2 = ON, Debugger = OFF
	STM: Device dependent

See also

■ STM ■ STM.state

STM.OCPAutoIdle — Set OCP idle clock behavior

Format:	**STM**[*<instance>*]**.OCPAutoIdle** [**ON** | **OFF**]

OFF (default)	OCP clock is free running.
ON	OCP clock may be gated if interface is in idle mode.

See also

■ STM ■ STM.state

STM.PATTERN — Enable test pattern generator

Format:	**STM**[*<instance>*]**.PATTERN** *<pattern>*

If *<pattern>* in nonzero, the selected test pattern is output instead of STP messages.

See also

■ STM ■ STM.state

STM.ChannelRepeat   Period of channel packet insertion

Format:	**STM**[*<instance>*]**.ChannelRepeat** [**OFF** | *<value>*]
<value>:	**8** | **16** | **24** | **32** | **40** | **48** | **56** | **64** | **72** | **80** | **104** | **112** | **120**

OFF (default)	No extra channel packets are inserted into the STP data stream.
<value>	If *<value>* subsequent STP messages are written to the same software channel, an extra STP channel packet is inserted into the data stream. Due to the working load of the STM component it may happen that extra channel packets are inserted only every 2 * *<value>* packets from the same channel.

This option is only available if STP data are routed to the onchip buffer: **STM.PortRoute.Onchip**.

See also

■ STM   ■ STM.state

STM.MasterRepeat   Period of master packet insertion

Format:	**STM**[*<instance>*]**.MasterRepeat** [**OFF** | *<value>*]
<value>:	**8** | **16** | **24** | **32** | **40** | **48** | **56** | **64** | **72** | **80** | **104** | **112** | **120**

OFF (default)	No extra master packets are inserted into the STP data stream.
<value>	If *<value>* subsequent STP packets are generated by the same master, an extra STP master packet is inserted into the data stream. Due to the working load of the STM component it may happen that extra master packets are inserted only every 2 * *<value>* packets from the same master.

This option is only available if STP data are routed to the onchip buffer: **STM.PortRoute.Onchip**.

See also

■ STM   ■ STM.state

STM.DMArequests DMA requests enable

Format:	**STM**[<*instance*>]**.DMArequests** [**OFF** | **25%** | **50%** | **75%** | **100%**]

Default: OFF.

The STM can request the DMA to write to the stimulus ports. Requests in turn are only issued if the internal STM FIFO contains less data than the stated filling level. This command does not set up the DMA.

See also

- STM
- STM.state

STM.COMPression Data compression enable

Format:	**STM**[<*instance*>]**.COMPression** [**ON** | **OFF**]

Default: OFF.

Enables or disables the automatic data compression of the STM. E.g. with compression enabled a 32-bit packet (D32) will be converted into an 8-bit packet (D8) if the value written to a stimulus port is less than 256.

See also

- STM
- STM.state

STM.EventMASK — Mask hardware event inputs

Format:	**STM**[*<instance>*]**.EventMASK** *<mask>*

Default: 0xFFFFFFFF

This 32-bit mask enables or disables hardware event inputs for packet generation. Thereby the LSB of the mask corresponds to hardware event input #0, the MSB corresponds to hardware event input #31.

See also

■ STM ■ STM.state

STM.PortMASK — Mask stimulus ports

Format:	**STM**[*<instance>*]**.PortMASK** *<mask>*

Default: 0xFFFFFFFF

This 32-bit mask enables or disables stimulus ports for instrumentation; that is if a bit of the mask is cleared, writes accesses to the corresponding stimulus port will not result in the generation of STP packets. Thereby the LSB of the mask corresponds to stimulus port #0, the MSB corresponds to stimulus port #31.

See also

■ STM ■ STM.state

STM.TraceID — Sets trace ID

Format:	**STM**[*<instance>*]**.TraceID** *<id>* | *<id_start>*--*<id_end>*

Default: 0x11

Sets the trace ID of the STM. To decode traces from multiple STM instances, define an ID range.

<id_start>	Must be an even number.

See also

- STM
- STM.state

STM.TracePriority — Set priority for STM manually

Format:	**STM**[*<instance>*]**.TracePriority** *<priority>*

TRACE32 automatically assigns an appropriate priority to the STM. This command allows the user to change the priority for the STM trace information.

See also

- STM
- STM.state

STM<trace> - Trace Data Analysis

STM<trace> Command groups for STM<trace> recording and analysis

See also

■ STMAnalyzer ■ STMCAnalyzer ■ STMHAnalyzer ■ STMLA
■ STMOnchip ■ STMTrace

▲ 'STM Component - General Target Configuration' in 'System Trace User's Guide'

Overview STM<trace>

[Example]

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

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

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

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

STMTrace.List              ;display a trace listing of the STM trace data
                           ;recorded with the trace method Analyzer.
                           ;STMTrace.List is the generic replacement
                           ;for the command used below:
STMAnalyzer.List           ;this is the equivalent and explicit command.
```

NOTE:	In the example above, the output of **STMTrace.List** is the same as the output of **STMAnalyzer.List**.

STMAnalyzer Analyze STM data recorded by TRACE32 PowerTrace

[Example]

Format:	**STMAnalyzer.**<*sub_cmd*> [<*stm_channel*>…] [<*channel*>…]

The **STMAnalyzer** command group allows to display and analyze the information emitted by the system trace implementations listed in the **"Introduction"**, page 8.

The STM information is emitted off-chip via:

- The Trace Port Interface Unit (TPIU), which is configured with the **TPIU** command group.
- Or the Parallel Trace Interface (PTI), which is configured with the **STM** command group.

The emitted STM information 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 **STMAnalyzer.List**, refer to **<trace>.List**
<*stm_channel*>, <*channel*>	For information about the channels, see **STMTrace**.

See also

■ STM<trace>

▲ 'STM Component - General Target Configuration' in 'System Trace User's Guide'

Format:	**STMCAnalyzer.***<sub_cmd>* [*<stm_channel>*…] [*<channel>*…]

The **STMCAnalyzer** command group allows to display and analyze the information emitted by the system trace implementations listed in the **"Introduction"**, page 8.

The STM information is emitted off-chip via:

- The Trace Port Interface Unit (TPIU), which is configured with the **TPIU** command group.
- Or the Parallel Trace Interface (PTI), which is configured with the **STM** command group.
- Or via Serial Wire Output (SWO), which is also configured with the **TPIU** command group.

The emitted STM information 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 **STMCAnalyzer.List**, refer to **<trace>.List**
<stm_channel>, *<channel>*	For information about the channels, see **STMTrace**.

See also

■ STM<trace>

▲ 'STM Component - General Target Configuration' in 'System Trace User's Guide'

STMHAnalyzer   Display and analyze STM data recorded by the host

Format:	**STMHAnalyzer.***<sub_cmd>* [*<stm_channel>*…] [*<channel>*…]

The **STMHAnalyzer** command group allows to display and analyze the information emitted by the system trace implementations listed in the **“Introduction”**, page 8.

Trace data is transferred off-chip via the USB port and 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 **STMHAnalyzer.List**, refer to **<trace>.List**
<stm_channel>, *<channel>*	For information about the channels, see **STMTrace**.

See also

■ STM<trace>

▲ ’STM Component - General Target Configuration’ in ’System Trace User’s Guide’

STMLA   Display and analyze STM data from binary file

Format:	**STMLA.***<sub_cmd>* [*<stm_channel>*…] [*<channel>*…]

<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 **STMLA.List**, refer to **<trace>.List**
<stm_channel>, *<channel>*	For information about the channels, see **STMTrace**.

See also

■ STM<trace>

▲ ’STM Component - General Target Configuration’ in ’System Trace User’s Guide’

Format:	**STMOnchip.***<sub_cmd>* [*<stm_channel>*…] [*<channel>*…] **STMOnchip2.***<sub_cmd>* [*<stm_channel>*…] [*<channel>*…]

The **STMOnchip** command group allows to display and analyze the information emitted by the system trace implementations listed in the **“Introduction”**, page 8.

The STM trace is sent to the device-specific onchip trace memory.

<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 **STMOnchip.List**, refer to **<trace>.List**
<stm_channel>, *<channel>*	For information about the channels, see **STMTrace**.

See also

■ STM<trace>

▲ ’STM Component - General Target Configuration’ in ’System Trace User’s Guide’

[Example]

Format:	**STMTrace.**<*sub_cmd*> [<*stm_channel*>…] [<*channel*>…]
<*stm_ channel*>:	**STMTITS** | **STMMASTER** | **STMCHANNEL**

The **STMTrace** command group can be used as a generic replacement for the above **STM<trace>** command groups.

<*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 **STMTrace.List**, refer to **<trace>.List**

<*stm_channel*>	The following, additional channels are available for the analysis of STP trace data:
STPTITS	Displays raw timestamp information of DxxTS messages. Only for TI Onchip traces.
STPV1TS	Display raw timestamp information of timestamped packets. Only for MIPI based STPv1 traces.
STPV2TS	Display raw timestamp information of timestamped packets. Only for MIPI based STPv2 traces.
STMMASTER	Displays the master ID of each message.
STMCHANNEL	Displays the channel ID of each message.

<*channel*>	For a description of the default channels, see **<trace>.List … <items>**.

STMMASTER and **STMCHANNEL** information can only be displayed if a master or channel message has been stored in the ETB prior to the current message. Otherwise the corresponding column will remain empty.

Example 1: The recommended way to display STP data generated by the STM:

```
STMTrace.List STMMASTER STMCHANNEL CYcle Data TIme.Back
```

Example 2: In case of TI, the recommended way to display STP onchip data is:

```
STMTrace.List STMMASTER STMCHANNEL CYcle Data STMTITS TIme.Back
```

See also

■ STM<trace>

▲ 'STM Component - General Target Configuration' in 'System Trace User's Guide'

PrintfTrace — Decoder for STP-based software messages

[Example]

Format:	**PrintfTrace.**<*sub_cmd*>

Applications running on a CPU may use the System Trace to output ’printf’-style software messages. The trace output can be displayed or analyzed with the **PrintfTrace** command group. Three different message types are available:

- String messages
- Kernel log messages
- Kernel FTRACE messages

<*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 **PrintfTrace.List**, refer to **<trace>.List**

String messages

String messages in general start with a data packet and are terminated by a time-stamped data packet or FLAG packet. Depending on the STP version being used, the PrintfTrace decoder decodes a STP software message as follows:

	STPv1	STPv2
Start of message	D8, D16, D32, D64	D4, D8, D16, D32, D64
Message body	D8, D16, D32, D64	D4, D8, D16, D32, D64
End of message	D8TS, D16TS, D32TS, D64TS	D4TS, D8TS, D16TS, D32TS, D64TS FLAG

Kernel log messages

Similar format as string messages, except that messages are initiated by a timestamped packet and terminated by a FLAG packet:

	STPv1	STPv2
Start of message		D4TS, D8TS, D16TS, D32TS, D64TS
Message body		D4, D8, D16, D32, D64
End of message		FLAG

In order to differentiate between regular string and kernel messages, **STM.PrintfTraceFormat Kernel** must be used.

Kernel FTRACE messages

These messages resemble a simple flow trace based on function calls with a source and target address. They always start with a D32TS packet which' lower 16bit data must be 0x0001. The message body consists of 3 D32 packets, followed by a FLAG packet:

	STPv1	STPv2
Start of message	n.a.	D32TS (0x????0001)
Process ID	n.a.	D32
Target address	n.a.	D32
Source address	n.a.	D32
End of message	n.a.	FLAG

In order to differentiate between regular string and kernel FTRACE messages, **STM.PrintfTraceFormat Kernel** must be used.

The following signals are relevant for all three types of software messages:

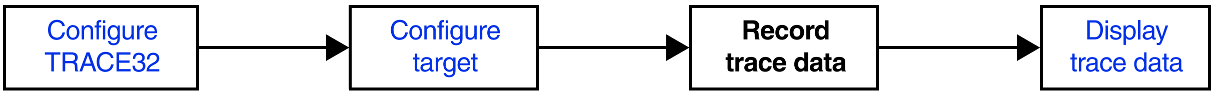
MESSAGE>	- Decoded normal string - FTRACE target function - Decoded kernel log
sYmbol	Fully translated FTRACE source function
sYmbolN	FTRACE source function
PID	FTRACE process ID
STMMASTER	Master ID of software message.
STMCHANNEL	Channel ID of software message.

Example

```
; Example of full trace listing for kernel messages

PrintfTrace.List STMMASTER STMCHANNEL pid sYmbolN MESSAGE
```

CMI Component



SYStem.CONFIG.CMI — Inform TRACE32 about CMI component

[Process Overview]

Format:	**SYStem.CONFIG.CMI**<*instance*>**.**<*sub_cmd*>
<*instance*>:	**1** | **2**
<*sub_cmd*>:	<*generic*> | <*component_specific*>
<*component_specific*>:	**TraceID** <*id*>

If the CMI is not enabled for your specific device, use the following commands for configuration. Both, the base address and the ID must be set in order to enable the CMI.

<*generic*>	For descriptions of the generic subcommands, click here.
1	Instance of the primary CMI component.
2	Instance of the secondary CMI component.
TraceID <*id*>	Sets the ATB trace ID of the corresponding CMI component.

Example:

```
SYStem.CONFIG.CMI1.Base       AHB:0x4A004F00
SYStem.CONFIG.CMI1.TraceID    0xf8
```

See also

■ CMI

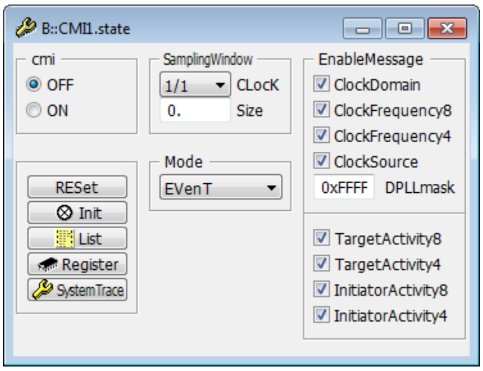
▲ 'Generic Subcommands, Parameters, and Options' in 'System Trace User's Guide'

Format:	**CMI***<instance>***.***<sub_cmd>*
<instance>:	**1** | **2**

The Clock Management component monitors clock activity and component activity of other components on the OMAP4. For more detailed information refer to the OMAP4 ETRM available from **https://www-a.ti.com/extranet/programs/emulation/OMAP4_ETRM_2.0-Setup.exe.**

1	Instance of the primary CMI component.
2	Instance of the secondary CMI component.
<sub_cmd>	For a description of the subcommands, refer to the command descriptions in this chapter.

For configuration of the primary or secondary CMI component, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **CMI1.state** or **CMI2.state** window.



cmi: [✓ OFF] ON

SamplingWindow: 1/1 CLocK; 0. Size

Mode: EVenT

EnableMessage:
- ✓ ClockDomain
- ✓ ClockFrequency8
- ✓ ClockFrequency4
- ✓ ClockSource
- 0xFFFF DPLLmask
- ✓ TargetActivity8
- ✓ TargetActivity4
- ✓ InitiatorActivity8
- ✓ InitiatorActivity4

To display and analyze the recorded trace data, use the **CMITrace** command group.

See also

■ CMI.EnableMessage	■ CMI.Mode	■ CMI.OFF	■ CMI.ON
■ CMI.Register	■ CMI.RESet	■ CMI.SamplingWindow	■ CMI.state
■ CMITrace	■ SYStem.CONFIG.CMI	❏ CMIBASE()	

Format:	**CMI**<*instance*>**.EnableMessage.**<*message*> [**ON** | **OFF**]	
<*message*>:	**ClockDomain**	(event message)
	ClockFrequency8	(event message)
	ClockFrequency4	(event message)
	ClockSource	(event message)
	DPLLmask	(event message)
	TargetActivity8	(activity message)
	TargetActivity4	(activity message)
	InitiatorActivity8	(activity message)
	InitiatorActivity4	(activity message)

Default: OFF.

Event messages are emitted for all clock domains derived from the same Digital Phase-Locked Loop (DPLL). They are only emitted on state changes and if **CMI<instance>.Mode EVenT** has been selected.

ClockDomain	Trace clock domain state changes (on / off).
ClockFrequency8	Trace clock frequency changes (8-bit divider ratio).
ClockFrequency4	Trace clock frequency changes (4-bit divider ratio).
ClockSource	Trace clock source selection changes (MUX input).
DPLLmask	Trace DPLL setting changes. Each of the 16 lower bits of DPLLmask represents one DPLL.

The following activity messages contain the active cycles count of the target or initiator. They are emitted on a periodically basis, even if the debugger in a halted state. Activity monitoring must be enabled via **CMI<instance>.Mode ACTivity** in addition.

TargetActivity8	Count target activity cycles. (If **CMI<instance>.SamplingWindow.Size** >= 16)
TargetActivity4	Count target activity cycles. (If **CMI<instance>.SamplingWindow.Size** < 16)
InitiatorActivity8	Count initiator activity cycles. (If **CMI<instance>.SamplingWindow.Size** >= 16)
InitiatorActivity4	Count initiator activity cycles. (If **CMI<instance>.SamplingWindow.Size** < 16)

See also

- ■ CMI
- ■ CMI.state

CMI.Mode — Set event or activity mode

Format:	**CMI**<*instance*>**.Mode** [**EVenT** | **ACTivity**]

EVenT (default)	Selects event mode monitoring.
ACTivity	Selects activity mode monitoring.

See also

- CMI
- CMI.state

CMI.OFF — Switch CMI off

Format:	**CMI**<*instance*>**.OFF**

Switches the CMI component off.

See also

- CMI
- CMI.state

CMI.ON — Switch CMI on

Format:	**CMI**<*instance*>**.ON**

Switches the CMI component on.

See also

- CMI
- CMI.state

CMI.Register — Display the CMI register

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

Displays the CMI registers.

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

See also

- CMI
- CMI.state

CMI.RESet — Resets CMI settings to their defaults

Format:	**CMI***<instance>***.RESet**

All CMI settings are reset to their defaults.

See also

- CMI
- CMI.state

See also

- CMI.SamplingWindow.CLocK
- CMI
- CMI.SamplingWindow.Size
- CMI.state

CMI.SamplingWindow.CLocK Set sampling window ratio

Format:	**CMI**<*instance*>**.SamplingWindow.CLocK** <*ratio*>

Default: 1/1

<*ratio*>	Divider ratio of the sampling window clock. It is derived from the CMI component's clock. Valid ratios range from 1/1 to 1/16.

See also

- CMI.SamplingWindow

CMI.SamplingWindow.Size Set sampling window size

Format:	**CMI**<*instance*>**.SamplingWindow.Size** <*cycles*>

Default: 1

<*cycles*>	Size of the sampling window. Smaller windows allow for more accurate activity or event reports while bigger sampling windows reduce trace traffic. Valid sizes range from 1 to 256.

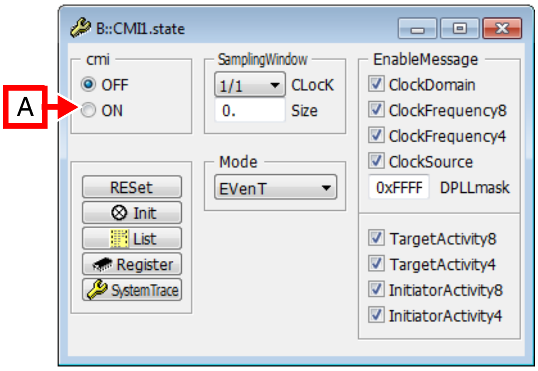
See also

- CMI.SamplingWindow

Format:	**CMI**<*instance*>**.state**
<*instance*>:	**1** | **2**

Opens the **CMI1.state** and **CMI2.state** window, where you can configure the CMI trace source 1 and 2.

Example of the **CMI1.state** window for the CMI trace source 1:



A For descriptions of the commands in the **CMI<instance>.state** window, please refer to the **CMI.*** commands in this chapter.
Example: For information about **ON**, click **CMI.ON**.

Exceptions:

- The **List** button opens the **CMITrace.List** window, see **<trace>.List**. For more information, refer to the description of the **CMITrace** command group.
- The **SystemTrace** button opens the **SystemTrace.state** window, see **<trace>.state**. For more information, refer to the description of the **SystemTrace** command group.

<*instance*>	For a description of <*instance*>, refer to the introduction to the command group **CMI**.

See also

■ CMI	■ CMI.EnableMessage	■ CMI.Mode	■ CMI.OFF
■ CMI.ON	■ CMI.Register	■ CMI.RESet	■ CMI.SamplingWindow

CMI Example

This example for an OMAP4430 which has got two CMI components. In this case the components are addressed as CMI1 or CMI2, respectively.

```
CAnalyzer.AutoArm OFF                  ; In activity mode CMI messages are
                                       ; output permanently, so we

CAnalyzer.ARM                          ; must arm the CAnalyzer before
                                       ; the CMI is activated.



CMI1.RESet

CMI1.SamplingWindow.Size 15

CMI1.Mode ACTivity

CMI1.ON

...

CMI1.OFF                               ; Turn off CMI ...

CAnalyzer.OFF                          ; ... before shutting down
                                       ; the CAnalyzer.



CMITrace.List CYcle CMITA.<name1>      ; Display cycle activity of target
CMITA.<name2>                          ; <name1> and <name2>.
```

[Process Overview] [Example]

Format:	**CMITrace.**<*sub_cmd*> [<*cmi_channel*>…] [<*channel*>…]
<*cmi_ channel*>:	**CMICD.**<*domain*> | **CMIDR.**<*clock*> | **CMICS**<*clock*> | **CMICR**<*clock*> | **CMIDPLL.**<*setting*> | **CMITA.**<*target*> | **CMIIA.**<*initiator*> | **CMISTAT** | **CMI-LAT** | **CYcle**

Using the **CMITrace** command group, you can analyze and display the recorded CMI trace data. The command group consists of the name of the trace source, here **CMI**, plus the keyword **Trace** of the <*trace*> command group.

<*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 **CMITrace.List**, refer to **<trace>.List**

<*cmi_channel*>	The following, additional channels are relevant for the analysis of CMI trace data:
CMICD.<*domain*>	Clock state of domain <*domain*>
CMIDR.<*clock*>	Divider ratio of clock <*clock*>
CMICR<*clock*>	Ratio of clock <*clock*>
CMICS<*clock*>	Source of clock <*clock*>
CMIDPLL.<*setting*>	DPLL setting <*setting*>
CMITA.<*target*>	Target <*target*> activity
CMIIA.<*initiator*>	Initiator <*initiator*> activity
CMISTAT	Only applies to event messages: Error flag indicating event message loss(es) caused by an undersized sampling window.
CMILAT	Event messages: Export latency in multiples of the sampling window. Activity messages: Export latency in multiples of target or initiator cycles.
CYcle	Domain name.

<channel>	For a description of the default channels, see **<trace>.List … <items>**.

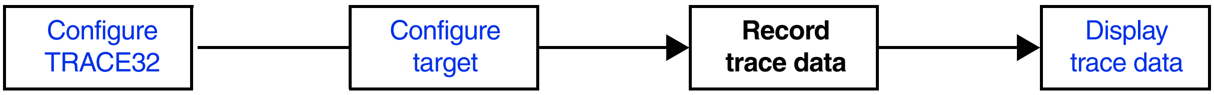
Example:

```
; Display cycle activity of target <name1> and <name2>.
CMITrace.List CYcle CMITA.<name1> CMITA.<name2>
```

See also

■ CMI

CMN Component



SYStem.CONFIG.CMN Inform TRACE32 about CMN component

[Process Overview]

Format:	**SYStem.CONFIG.CMN**<instance>**.**<*sub_cmd*>
<instance>:	**1 | 2 | 3 | 4**
<*sub_cmd*>:	<*generic*> | <*component_specific*>
<component_specific>:	**DEViceName** <string> <string> **RootNodeBase** <*address*> **XYBits 2bit | 3bit | 4bit | 2bitEXtra | 3bitEXtra | 4bitEXtra | 0bitEXtra**

If the Coherent Mesh Network (CMN) component is not enabled for your specific device, use the following commands for configuration. Please be aware, that a single CMN component represents a so-called Debug Trace Controller (DTC) and not a whole CMN. A DTC is used to set general Trace settings and works as a Trace output interface. For configuration, two addresses are needed. The Base address represents the Debug Trace Controller's configuration register block in your System's memory. On the other hand, the RootNodeBase address represents as starting point for crosspoint (XP) detection, so that you are able to set individual trace filters. If a CMN has more than one DTC, all CMN components must have the same RootNodeBase address.

Furthermore, TRACE32 needs to be informed about the number of bits used to identify the XPs inside the mesh. If this is not done correctly, the component won't work properly.

<instance>	Instance of the current CMN component.
<generic>	For descriptions of the generic subcommands, click here.
DEViceName	Set a device's name that is connected to the mesh via the XP port given in the first string argument. The device's name can be shown in the trace listing window.

RootNodeBase	Starting address for CMN crosspoint detection.
XYBits	Number of bits used to address the dimension of a single XP inside the mesh network.The “*bitEXtra” variants should be used with CMNs that support “extra device ports” on a crosspoint. E.g. In case the network has only one XP, the option “0bitEXtra” should be used.

Examples:

```
; Example configuration for a CMN with a single DTC
SYStem.CONFIG.CMN1.Base         AXI:0x50D30000
SYStem.CONFIG.CMN1.RootNodeBase AXI:0x50D00000
SYStem.CONFIG.CMN1.XYBits       2bit
SYStem.CONFIG.CMN1.DEViceName   “(2,1,0,0)” “Core0”

; Example configuration for a CMN with two DTCs
SYStem.CONFIG.CMN1.Base         AXI:0x31034000
SYStem.CONFIG.CMN1.RootNodeBase AXI:0x31004000
SYStem.CONFIG.CMN1.XYBits       3bit
SYStem.CONFIG.CMN2.Base         AXI:0x31A34000
SYStem.CONFIG.CMN2.RootNodeBase AXI:0x31004000
SYStem.CONFIG.CMN2.XYBits       3bit
```

NOTE:	Please be aware, that the RootNodeBase address of multiple CMN<instance> components must be the same if they are part of the same network.

If you are unsure about the necessary address values, Lauterbach provides a detection script in the directory: “~~/demo/arm/etc/cmn/cmn_detect.cmm”

Example:

```
;The detection script searches for the component IDs of the CMN DTC and
CMN Root Node

DO ~~/demo/arm/etc/cmn/cmn_detect.cmm START=AXI:0x50C00000
END=AXI:0x50E00000
```

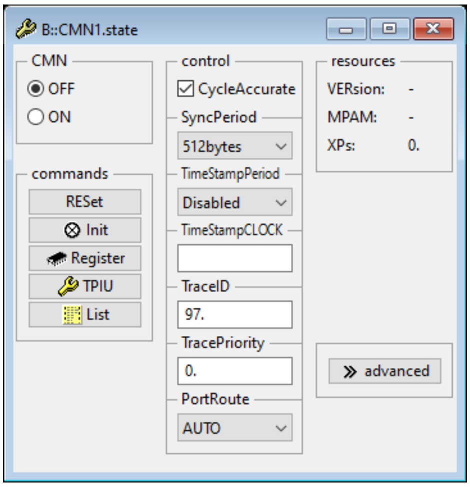
Format:	**CMN**<instance>**.***<sub_cmd>*
<instance>:	**1 | 2 | 3 | 4**

CMN stands for Coherent Mesh Network. It is a scalable configurable coherent interconnect designed by ARM and used in high-end networking and enterprise compute applications. The CMN's Debug and Trace functionality allows for non-intrusive tracing of the messages sent or received at each CMN crosspoint (XP).TRACE32 supports up to 4 independent Debug Trace Controllers inside a single CMN. Though the trace outputs of the independent Debug Trace Controllers can be configured individually, the filters set on the XPs apply to all of them.

1	Instance of the first CMN component.
2	Instance of the second CMN component.
3	Instance of the third CMN component.
4	Instance of the fourth CMN component.
<sub_cmd>	For a description of the subcommands, refer to the command descriptions in this chapter.

Each XP supports following channels: Request (REQ), Response (RSP), Snoop (SNP), and Data (DAT). Furthermore, each XP provides means to filter the traced messages individually. As the CMN is connected to the System's Advanced Trace Bus (ATB), the trace output can be configured similarly to other ARM CoreSight components. For example, the trace can be set up as on-chip trace by using an Embedded Trace Buffer or as off-chip trace by routing the data to a trace port.

For configuration, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **CMN**<instance>**.state** window.


B::CMN1.state

CMN: [✓ OFF] ON

commands: RESet, Init, Register, TPIU, List

control:
[✓] CycleAccurate
SyncPeriod: 512bytes
TimeStampPeriod: Disabled
TimeStampCLOCK:
TraceID: 97.
TracePriority: 0.
PortRoute: AUTO

resources:
VERsion: -
MPAM: -
XPs: 0.

advanced

To display and analyze the recorded trace data, use the **CMNTrace** command group.

CMN.CycleAccurate Enable cycle counter information

[build 153418 - DVD 02/2023]

Format:	**CMN**<instance>**.CycleAccurate** [**ON** | **OFF**]

Default: ON.

If this flag is set, the recorded trace data will contain timing information in form of 16 bit cycle counter values. These values are then used to precisely determine the timings of the received trace messages. Cycle counter values allow a better precision than global timestamps.

ON	Enable MPAM format.
OFF	Disable MPAM format.

NOTE:	Currently, TRACE32 does not allow the simultaneous use of timestamps and cycle counter values in CMN trace. So either the generation of timestamps or cycle counter information is turned off.

CMN.EnhancedFilter — Set an individual filter on a CMN XP

Format: **CMN**<instance>**.EnhancedFilter** *<x_coord> <y_coord>* <wp_num> <config> <value> <mask>

Configures the filter for a single watchpoint on a single CMN XP. Only the filtered messages are visible to the ATB. For further details, refer to the *Technical Reference Manual* of your system's CMN.

<x_coord>	X coordinate of the XP which is configured The used value can range from 0. to 7. for CMN-600 and from 0. to 15. for CMN-700 variants.
<y_coord>	Y coordinate of the XP which is configured The used value can range from 0. to 7. for CMN-600 and from 0. to 15. for CMN-700 variants.
<wp_num>	Number of the XP's watchpoint which is configured Each XP provides 4 watchpoints, therefore the used value can range from 0. to 3.
<config>	The value which is written to the selected watchpoint's config register. For detailed information refer to the register por_dtm_wp<n>_config in the *Technical Reference Manual* of your system's CMN.

<value>	The comparison value which is written to the selected watchpoint's value register. For detailed information refer to the register por_dtm_wp<n>_val in the *Technical Reference Manual* of your system's CMN.
<mask>	The comparison mask which is set to the selected watchpoint's mask register. For detailed information refer to the register por_dtm_wp<n>_mask in the *Technical Reference Manual* of your system's CMN.

Example:

```
;The following lines show how to filter REQ flits corresponding to a
"ReadShared" transaction to address=0x12345 uploaded from port 1 at XP
(2,1)
;This can be done by configuring the Watchpoints 0. and 1.

CMN1.EnhancedFilter 2. 1. 0. 0x1941 0x80000000 0xFFFFFFE07FFFFFFF
CMN1.EnhancedFilter 2. 1. 1. 0x1911 0x123450 0xFFF000000000000F


;The following line shows how to filter for all RSP flits downloaded to
port 0 at XP (0,1)

CMN1.EnhancedFilter 0. 1. 2. 0x1902 0x0 0xFFFFFFFFFFFFFFFF
```

NOTE:	This command resets all global filters which have been set by **CMN.NodeID**, **CMN.Opcode**, and **CMN.TraceChannel**.

CMN.Init — Initialize CMN on target

Format:	**CMN**<instance>**.Init**

Initializes the CMN registers on the target.

Format:	**CMN.MPAM** [**ON** | **OFF**]

Default: OFF.

Defines the MPAM format of CMN tracking messages.

When using the simulator, the *mpam_en* bit cannot be used for detection as it would always be 0.

ON	Enable MPAM format.
OFF	Disable MPAM format.

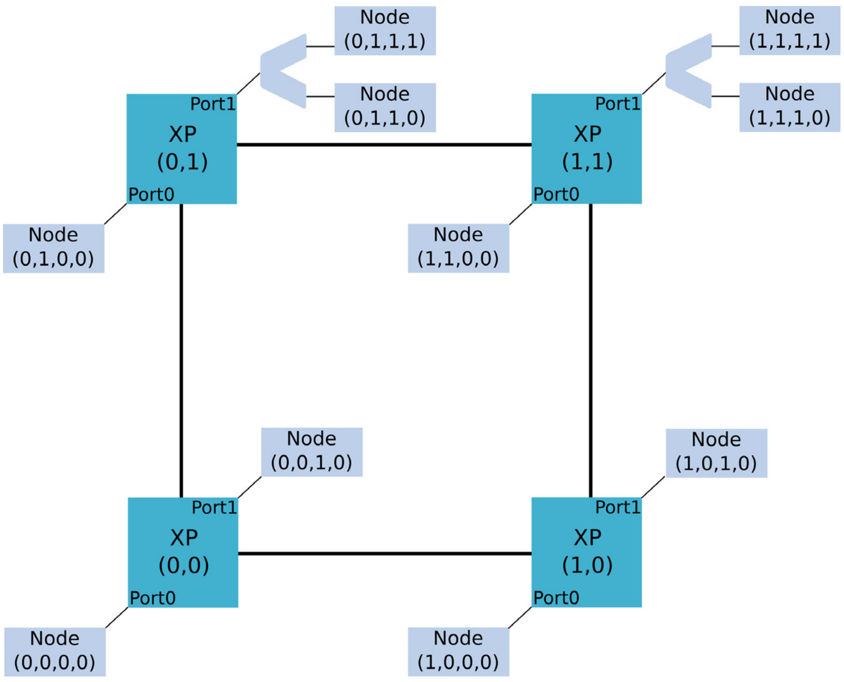
CMN.NodeID

Format:	**CMN**<instance>**.NodeID** *<xp_nodeid>*
	CMN.SourceID (deprecated)

Default: None

This command sets a global filter for flits sent to the specified Node. In case the selected CMN channel is set to SNP, then this value is used to set a global filter for flits, which are sent by the specified Node. The picture below shows the structure of an example CMN with XPs, Nodes and their respective IDs.

<xp_nodeid>	XP node id in the form **"(X,Y,Port,DevID)"**



NOTE: This command resets the configurations done by **CMN.EnhancedFilter**

Example:

```
;Trace all CMN REQ flits sent to Subdevice at port 1 of XP (2,3)

CMN1.NodeID "(2,3,1,0)"
```

CMN.OFF — Switch CMN trace off

Format:	**CMN**<instance>**.OFF**

Performs a global disable of traces from the selected CMN component.

CMN.ON — Switch CMN trace on

Format:	**CMN**<instance>**.ON**

Performs a global enable of traces from the selected CMN component.

CMN.Opcode — Set global filter for opcode

Format:	**CMN**<instance>**.Opcode** *<value>*

Default: None

Sets a global filter for the specified opcode uploaded at each XP.

NOTE:	Depending on the selected CMN **channel**, the same opcode value can describe different trace packets. This command resets the configurations done by **CMN.EnhancedFilter**

CMN.PortRoute — Select output of CMN data

Format:	**CMN**<instance>**.PortRoute** <port_route>
<port_route>	**AUTO** | **Analyzer** | **CAnalyzer** | **Onchip** | **Onchip2** | **Onchip3** | **Onchip4** | **Onchip5**

Default: AUTO

Prepares the selected trace hardware for CMN trace capture.

AUTO	Automatic detection
Analyzer	PowerTrace (via TPIU)
CAnalyzer	Compact-Analyzer: CombiProbe or µTrace (MicroTrace)
Onchip **Onchip2** **Onchip3** **Onchip4** **Onchip5**	Onchip trace buffer (ETB, ETF or ETR)

CMN.Register — Display CMN register

Format:	**CMN**<instance>**.Register** [*<file>*] [/*<option>*]
<option>:	**SpotLight** | **DualPort** | **Track** | **CORE** *<core_number>*

Displays the CMN registers.

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

CMN.RESet

Format: **CMN**<instance>**.RESet**

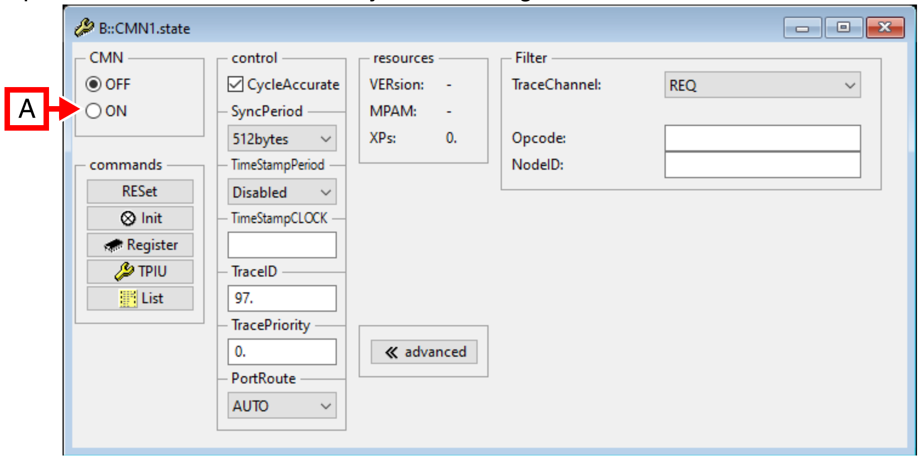
Resets all CMN settings to default.

CMN.state

Display CMN settings

Format: **CMN**<instance>**.state**

Opens the **state** window, where you can configure the trace source of the selected CMN instance.



CMN: (● OFF) ○ ON
control: ☑ CycleAccurate
SyncPeriod: 512bytes
TimeStampPeriod: Disabled
TimeStampCLOCK:
TraceID: 97.
TracePriority: 0.
PortRoute: AUTO
resources:
VERsion: -
MPAM: -
XPs: 0.
Filter:
TraceChannel: REQ
Opcode:
NodeID:

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

Exceptions:

- The **TPIU** button opens the **TPIU** window, see **TPIU.state**.
- The **List** button opens the **SystemTrace.List** window, see **<trace>.List**.

<instance>	For a description of <instance>, refer to the introduction to the command group **CMN**.

CMN.SyncPeriod Set period of synchronisation packet

Format:	**CMN**<instance>**.SyncPeriod** *<period>*
<period>	**256bytes** | **512bytes** | **1Mbytes**

Default: 512bytes

Configures the amount of trace packet data sent between two synchronization packets.

CMN.TimeStampCLOCK Configure debugger for CMN timestamp clock

[build 152301 - DVD 09/2022]

Format:	**CMN<instance>.TimeStampCLOCK** *<frequency>*

Default: 0

Configures the debugger to use the given frequency for the CMN timestamp calculation. The frequency is required to calculate timing information based on timestamp values.

CMN.TimeStampPeriod Set period of timestamp packet

Format:	**CMN**<instance>**.TimeStampPeriod** *<period>*
<period>:	**Disabled | 8kCycles | 16kCycles | 32kCycles | 64kCycles**

Default: Disabled

Configures the timestamp packet insertion period in clock cycles. Global timestamp values can be used to correlate CMN trace with trace data from other sources. However, they are not as precise as cycle counter values.

NOTE:	Currently, TRACE32 does not allow the simultaneous use of timestamps and cycle counter values in CMN trace. So either the generation of timestamps or cycle counter information is turned off.

CMN.TraceChannel Set global filter for CMN channel

Format:	**CMN**<instance>**.TraceChannel** *<channel>*
<channel>	**REQ | RSP | SNP | DAT**

Default: REQ

Sets a global filter for the specified CMN channel at each XP. As the communication on the CMN is channel based, the selection of this command determines which trace messages are part of the trace. The CMN implements a **REQ**uest channel, a **R**e**SP**onse channel, a **SN**oo**P** channel, and a **DAT**a channel.

NOTE:	This command resets the configurations done by **CMN.EnhancedFilter**

CMN.TraceID Sets trace ID

Format:	**CMN**<instance>**.TraceID** *<id>* | *<id_start>*--*<id_end>*

Default: 0x61

Sets the trace ID of the CMN. To decode traces from multiple CMN instances, define an ID range.

<id_start>	Must be an even number.

CMN.TracePriority Set priority for CMN manually

Format:	**CMN**<instance>**.TracePriority** *<priority>*

TRACE32 automatically assigns an appropriate priority to the CMN. This command allows the user to change the priority for the CMN trace information.

Format:	**CMN.VERsion** *<version>*
<option>:	**AUTO** | **CMN-600** | **CMN-650** | **CMN-700**

Default: AUTO.

Allows to specify manually the CMN type.

CMN Example

This example for a typical onchip setup with a NeoverseN1.

```
SYStem.CONFIG.CMN1.Base AXI:0x50d00000            ; Informs TRACE32 about
SYStem.CONFIG.CMN1.RootNodeBase AXI:0x50d30000    ; the CMN component at
SYStem.CONFIG.CMN1.XYBits 2bit                    ; CoreSight ETF1
SYStem.CONFIG.ETF1.Base DAP:0x80900000
SYStem.CONFIG.ETF1.ATBSource CMN

CMN1.Reset

CMN1.TraceID 3.

CMN1.TraceChannel RSP                             ; Configure CMN to
                                                  trace

CMN1.Opcode 0x3                                   ; only RetryAck packets

CMN1Trace.METHOD Onchip

CMN1Trace.ARM

...

CMN1Trace.OFF

CMN1Trace.List CYcle AAddress SRCNODE TGTNODE     ; Display trace packets
                                                  ; with CMN operation,
                                                  ; phys. address,
                                                  ; source node,
                                                  ; and target node
```

Format:	**CMN**<instance>**Trace.**<*sub_cmd*> [<*cmn_channel*>…] [<*channel*>…]
<*cmn_ channel*>:	**CMNDETAILS** | **CYcle** | **SRCNAME** | **SRCNODE** | **TAG** | **TGTNAME** | **TGTNODE** | **TXNID** | **WP**

Using the **CMNTrace** command group, you can analyze and display the recorded CMN trace data. The command group consists of the name of the trace source, here **CMN**<instance>, plus the keyword **Trace** of the <*trace*> command group.

<instance>	For a description of <instance>, refer to the introduction to the command group **CMN**.
<*sub_cmd*> **PMILV**	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 **CMNTrace.List**, refer to **<trace>.List**

<*cmn_channel*>	The following, additional channels are relevant for the analysis of CMN trace data:
CMNDETAILS	Shows flags, attributes and error codes from the trace packet. For a description of the shown values, please refer to the **next chapter**.
CYcle	Name of the trace packet type according to the opcode and channel. To see which types are possible, please refer to the *AMBA® 5 CHI Architecture Specification Appendix A*
SRCNAME	CMN subcomponent's name of the trace packet source which was set with **SYStem.CONFIG.CMNx.DEViceName**
SRCNODE	CMN subcomponent identifier of the trace packet source in the following format: (X,Y,Port,DevID)
TAG	Shows the trace packet's TRACETAG flag
TGTNAME	CMN subcomponent's name of the trace packet destination which was set with **SYStem.CONFIG.CMNx.DEViceName**
TGTNODE	CMN subcomponent identifier of the trace packet destination in the following format: (X,Y,Port,DevID)

TXNID	Trace packet's Transaction ID
WP	The DTM's watchpoint that generated the trace packet

<channel>	For a description of the default channels, see **<trace>.List … <items>**.

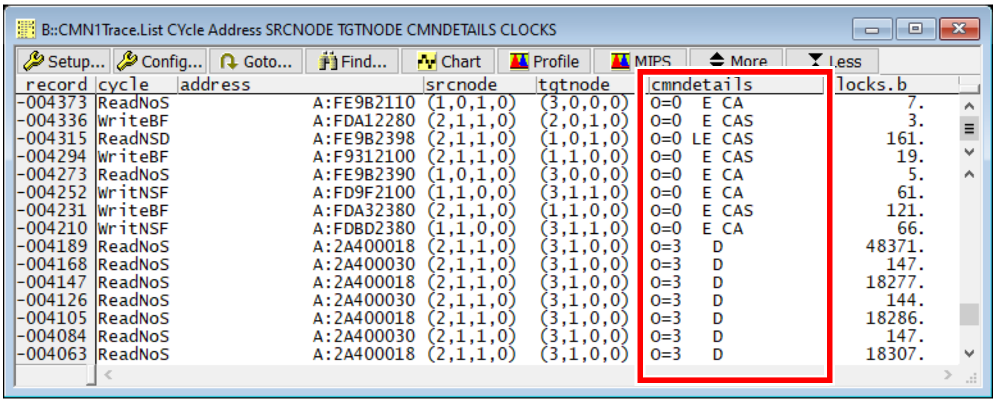
Example:

```
; Display cycle activity between SRCNODE and TGTNODE
CMN1Trace.List CYcle SRCNODE TGTNODE
```

NOTE:	Please be aware, that the CMN trace does not contain information about the "real" memory content. The trace provides meta information on the coherency protocol, only.

Description of the CMNDETAILS values

The CMNDETAILS trace channel displays flags, attributes and error codes contained in the CMN trace packets. Please be aware that not all information is contained in each CMN trace packet. The following picture shows an example CMN trace listing with a highlighted CMNDETAIL column.


B::CMN1Trace.List CYcle Address SRCNODE TGTNODE CMNDETAILS CLOCKS
Setup...
Config...
Goto...
Find...
Chart
Profile
MIPS
More
Less

```
 record |cycle   |address                  |srcnode   |tgtnode   |cmndetails     |locks.b
-004373 ReadNoS              A:FE9B2110 (1,0,1,0)  (3,0,0,0)  O=0  E CA            7.
-004336 WriteBF              A:FDA12280 (2,1,1,0)  (2,0,1,0)  O=0  E CAS           3.
-004315 ReadNSD              A:FE9B2398 (2,1,1,0)  (1,0,1,0)  O=0 LE CAS         161.
-004294 WriteBF              A:F9312100 (2,1,1,0)  (1,1,0,0)  O=0  E CAS          19.
-004273 ReadNoS              A:FE9B2390 (1,0,1,0)  (3,0,0,0)  O=0  E CA            5.
-004252 WritNSF              A:FD9F2100 (1,1,0,0)  (3,1,1,0)  O=0  E CA           61.
-004231 WriteBF              A:FDA32380 (2,1,1,0)  (1,1,0,0)  O=0  E CAS         121.
-004210 WritNSF              A:FDBD2380 (1,1,0,0)  (3,1,1,0)  O=0  E CA           66.
-004189 ReadNoS              A:2A400018 (2,1,1,0)  (3,1,0,0)  O=3   D          48371.
-004168 ReadNoS              A:2A400030 (2,1,1,0)  (3,1,0,0)  O=3   D            147.
-004147 ReadNoS              A:2A400018 (2,1,1,0)  (3,1,0,0)  O=3   D          18277.
-004126 ReadNoS              A:2A400030 (2,1,1,0)  (3,1,0,0)  O=3   D            144.
-004105 ReadNoS              A:2A400018 (2,1,1,0)  (3,1,0,0)  O=3   D          18286.
-004084 ReadNoS              A:2A400030 (2,1,1,0)  (3,1,0,0)  O=3   D            147.
-004063 ReadNoS              A:2A400018 (2,1,1,0)  (3,1,0,0)  O=3   D          18307.
```

If the trace packet originated from the REQ CMN channel, the CMNDETAILS display the CMN transaction attributes. This information is shown by following values:

O=<value>	This label displays the <value> given in the Order field.
L	If this letter is visible, the packet's LikelyShared flag is set.
E	If this letter is visible, the EWA bit is set in the packet's MemAttr field.
D	If this letter is visible, the Device bit is set in the packet's MemAttr field.

C	If this letter is visible, the Cacheable bit is set in the packet’s MemAttr field.
A	If this letter is visible, the Allocate bit is set in the packet’s MemAttr field.
S	If this letter is visible, the packet’s SnpAttr flag is set.

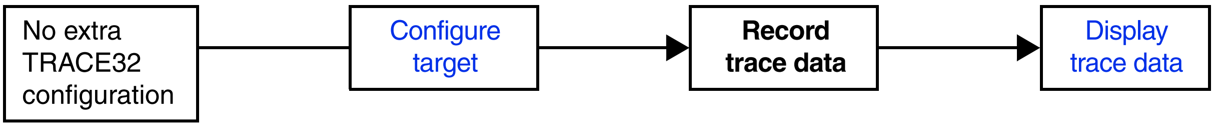
If the trace packet originated from either the RSP or DAT CMN channel, the CMNDETAILS display the error status of the CMN response. This information is given as shown here:

OK	Either the normal access was successful or the exclusive access failed.
EXOK	Either the read or write portion of an exclusive access was successful.
DERR	A data error occurred.
NDERR	A non-data error occurred.

If the trace packet originated from the SNP CMN channel, the CMNDETAILS display following flags relevant to CMN snoop packets:

DNGTSD	If this is visible, the packet’s DoNotGoToSD flag is set.
RTSRC	If this is visible, the packet’s RetToSrc flag is set.

CPTracer Component

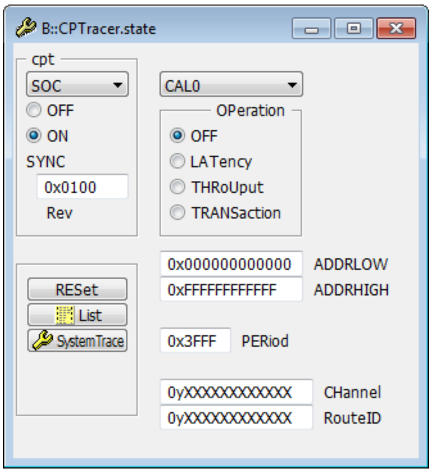


CPTracer — Configure CPTracer component on target

[Process Overview]

CPTracer stands for Common Platform Tracer (CPT). The CPTracer component allows you to collect statistics from different bus probes, such as latency, throughput and other transactional metrics.

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



B::CPTracer.state

cpt: SOC
OFF
[✓ ON]
SYNC
0x0100
Rev

CAL0
OPeration: [✓ OFF] LATency THRoUput TRANSaction

0x000000000000 ADDRLOW
0xFFFFFFFFFFFF ADDRHIGH
0x3FFF PERiod
0yXXXXXXXXXXXX CHannel
0yXXXXXXXXXXXX RouteID

To display and analyze the recorded trace data, use the **CPTracerTrace** command group.

See also

- CPTracer.<aggregator>.<probe>.ADDRessHIGH
- CPTracer.<aggregator>.<probe>.ADDRessLOW
- CPTracer.<aggregator>.<probe>.CHannel
- CPTracer.<aggregator>.<probe>.DIRection
- CPTracer.<aggregator>.<probe>.OPeration
- CPTracer.<aggregator>.<probe>.PERiod
- CPTracer.<aggregator>.<probe>.RouteID
- CPTracer.<aggregator>.OFF
- CPTracer.<aggregator>.ON
- CPTracer.<aggregator>.SYNC
- CPTracer.RESet
- CPTracer.state
- CPTracer.TraceID
- CPTracerTrace

CPTracer.RESet

Resets CPT settings to their defaults

Format: **CPTracer.RESet**

All CPT settings are reset to their defaults.

See also

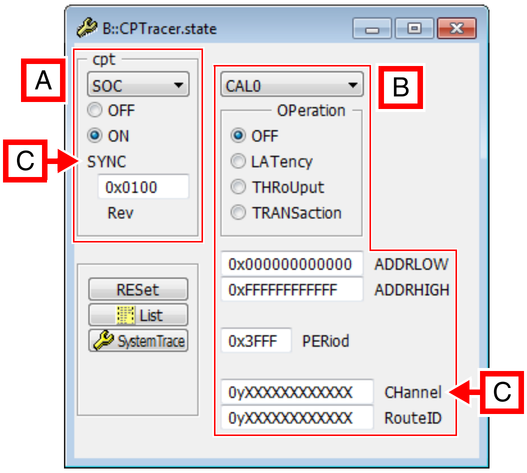
- CPTracer
- CPTracer.state

CPTracer.state

Display CPT settings

Format: **CPTracer.state**

Shows the CPT setup window.



A *<aggregator>*. Here, it is set to **SOC**. The fields below refer to the selected *<aggregator>*.

B *<probe>*. Here, it is set to **CAL0**. The fields below refer to the selected *<probe>*.

C For descriptions of the commands in the **CPTracer.state** window, please refer to the **CPTracer.*** commands in this chapter.
Example 1: For information about **SYNC**, see **CPTracer.<aggregator>.SYNC**.
Example 2: For information about **CHannel**, see **CPTracer.<aggregator>.<probe>.CHannel**.

See also

- CPTracer
- CPTracer.<aggregator>.<probe>.ADDRessHIGH
- CPTracer.<aggregator>.<probe>.ADDRessLOW
- CPTracer.<aggregator>.<probe>.CHannel
- CPTracer.<aggregator>.<probe>.DIRection
- CPTracer.<aggregator>.<probe>.OPeration
- CPTracer.<aggregator>.<probe>.PERiod
- CPTracer.<aggregator>.<probe>.RouteID
- CPTracer.<aggregator>.OFF
- CPTracer.<aggregator>.ON

- CPTracer.<aggregator>.SYNC
- CPTracer.TraceID
- CPTracer.RESet

CPTracer.TraceID Set ATB ID

Format:	**CPTracer.TraceID** *<id>*

Default: 0x50

Sets the CoreSight ATB ID of the first aggregator.

<id>	Will increase automatically with each subsequent aggregator.

See also

- CPTracer
- CPTracer.state

CPTracer.<aggregator>.ON Switch aggregator on

Format:	**CPTracer.***<aggregator>***.ON**

Performs a global enable of traces from all probes of the aggregator.

See also

- CPTracer
- CPTracer.state

CPTracer.<aggregator>.OFF Switch aggregator off

Format:	**CPTracer.***<aggregator>***.OFF**

Performs a global disable of traces from all probes of the aggregator.

See also

- CPTracer
- CPTracer.state

CPTracer.<aggregator>.SYNC Sync period of aggregator

Format: **CPTracer.SYNC** *<bytes>*

Default: 0x100

Sets the number of regular trace *<bytes>* between two synchronization packets.

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

In this example, the number of regular trace *<bytes>* is 0x100.

$\underbrace{\text{B0 … B255}}_{\text{0x100}}$ **SP** $\underbrace{\text{B0 … B255}}_{\text{0x100}}$ **SP** $\underbrace{\text{B0 … B255}}_{\text{0x100}}$ **SP** ...

B = regular trace *<bytes>*
SP = synchronization packet

See also

- CPTracer
- CPTracer.state

CPTracer.<aggregator>.<probe>.ADDRessLOW Lower filter address

Format: **CPTracer.***<aggregator>***.***<probe>***.ADDRessLOW** *<value>*

Default: 0

Only transactions with address >= *<value>* will generate trace packets. This command must be used together with **CPTracer.<aggregator>.ADDRessHIGH**.

See also

- CPTracer
- CPTracer.state

CPTracer.<aggregator>.<probe>.ADDRessHIGH   Upper filter address

Format:	**CPTracer.***<aggregator>***.***<probe>***.ADDRessLOW** *<value>*

Default: 0xFFFFFFFFFFFF

Only transactions with address <= *<value>* will generate trace packets. This command must be used together with **CPTracer.<aggregator>.ADDRessLOW**.

See also

■ CPTracer ■ CPTracer.state

CPTracer.<aggregator>.<probe>.CHannel   Filter by channel ID

Format:	**CPTracer.***<aggregator>***.***<probe>***.CHannel** *<bitmask>*

Default: 0yXXXXXXXXXXXX

Only transactions with a channel ID within *<bitmask>* will generate trace packets.

See also

■ CPTracer ■ CPTracer.state

CPTracer.<aggregator>.<probe>.DIRection   Filter by transfer direction

[build 140174 - DVD 02/2022]

Format:	**CPTracer.***<aggregator>***.***<probe>***.DIRection** *<direction>*
<direction>:	**Read** **Write** **ReadWrite**

Default: ReadWrite

Only transactions with selected *<direction>* will generate trace packets.

See also

- CPTracer
- CPTracer.state

CPTracer.<aggregator>.<probe>.OPeration Mode of operation

Format:	**CPTracer.***<aggregator>***.***<probe>***.OPeration** *<mode>*
<mode>:	**OFF** **LATency** **THRoUput** **TRANSaction**

Default: OFF.

Defines which type of trace packet is to be generated.

OFF	No trace packets.
LATency	Trace packets carrying latency information.
THRoUput	Trace packets carrying throughput information.
TRANSaction	Trace packets carrying transaction information.

See also

■ CPTracer ■ CPTracer.state

CPTracer.<aggregator>.<probe>.PERiod Set period of sample window

Format:	**CPTracer.***<aggregator>***.***<probe>***.PERiod** *<value>*

Default: 0x3FFF

Sets the period of the sample window which triggers trace packet generation.

See also

■ CPTracer ■ CPTracer.state

Format:	**CPTracer.***<aggregator>***.***<probe>***.RouteID** *<bitmask>*

Default: 0yXXXXXXXXXXXX

Only transactions with a route ID within *<bitmask>* will generate trace packets.

See also

■ CPTracer ■ CPTracer.state

CPTracer Example

This example refers to AM65xx devices. Please note that aggregator and probe names are device specific and may be different on other SOCs.

```
; The CPTracer aggregators are mapped to STM2 on AM65xx devices.
SystemTrace.Method Analyzer          ; Select trace port
STM2.TimeStamps ON                   ; We want to inspect STP timestamps
STM2.TraceID 0x50--0x53              ; Do not confuse with
                                     ; CPTracer.TraceID!
                                     ; We have to assign the same
                                     ; TraceID(s)twice.
                                     ; You can also use this command to
                                     ; filter an already captured trace.
SystemTrace.Init

; Now configure the CPTracer component:
CPTracer.RESet
CPTracer.SOC.CAL0.OPeration.LATency
CPTracer.SOC.MCU.EXPORT_SLV.OPeration.LATency

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

; Display the recorded trace data
CPTracerTrace.List PRobe CYcle Address
```

[Process Overview] [Example]

Format:	**CPTracerTrace.***<sub_cmd>* [*<cpt_channel>*...] [*<channel>*...]
<cpt_ channel>:	**PRobe** | **CYcle** | **Address** | **LAT.***<xxx>* | **TRANS.***<xxx>* | **THRU.***<xxx>* | **TIme.***<xxx>*

Using the **CPTTracerTrace** command group, you can analyze and display the recorded CPT trace data. The command group consists of the name of the trace source, here **CPTracer**, plus the keyword **Trace** of the *<trace>* command group.

<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 **CPTracerTrace.List**, refer to **<trace>.List**

<cpt_channel>	The following channels are relevant for the analysis of CPT trace data:
<xxx>	A list of all valid replacements for the placeholder *<xxx>* will be displayed as softkeys in TRACE32 as soon as the dot '.' is entered in the TRACE32 command line.
PRobe	Name of the probe. See example.
CYcle	Type of trace packet.
Address	Traced address of transaction packet. Also see note below.
LAT.*<xxx>*	Latency packet specific information *<xxx>*. See example.
TRANS.*<xxx>*	Transaction packet specific information *<xxx>*.
THRU.*<xxx>*	Throughput packet specific information *<xxx>*.
TIme.*<xxx>*	Timing information.

<channel>	For a description of the default channels, see **<trace>.List … <items>**.

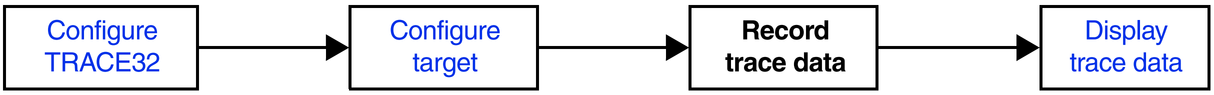
Example:

```
; Display maximum latency:
;                     <cpt_channel>   <cpt_channel>
CPTracerTrace.List      PRobe         LAT.maxwait
```

See also

■ CPTracer

OCP Component



SYStem.CONFIG.OCP Inform TRACE32 about OCP component

[Process Overview]

Format:	**SYStem.CONFIG.OCP.***<sub_cmd>*
<sub_cmd>:	*<generic>* | *<component_specific>*
<component_ specific>:	**TraceID** **Type 4**

If the OCP component is not enabled for your specific device, use the following commands for configuration. Both, the base address and the ID must be set in order to enable the OCP.

<generic>	For descriptions of the generic subcommands, click here.
Type 4	Currently only supported for OMAP4.
TraceID	Sets the STM master ID of the OCP component.

Deprecated vs. New Commands:

SYStem.CONFIG.TIOCPBASE (deprecated)	**SYStem.CONFIG.OCP.Base**
SYStem.CONFIG.TIOCPID (deprecated)	**SYStem.CONFIG.OCP.TraceID**

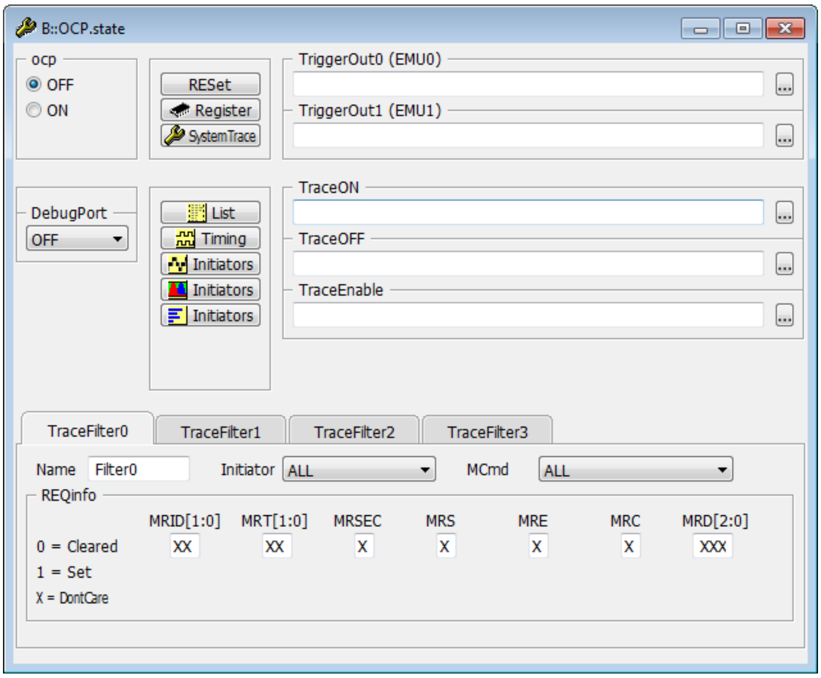
See also

■ OCP

▲ 'Generic Subcommands, Parameters, and Options' in 'System Trace User's Guide'

OCP stands for OpenCoreProtocol WatchPoint (OCP-WP). The OCP-WP monitors OCP requests directed to a selected target attached to the L3 interconnect of the OMAP4. Tracing the bus traffic is non-intrusive and enables the developer to capture all requests addressed to a target or only a subset of it defined by up to four different filters (see **OCP.TraceFilter<x>** commands).

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


B::OCP.state

ocp: [✓ OFF] ON

DebugPort: OFF

TriggerOut0 (EMU0):
TriggerOut1 (EMU1):
TraceON:
TraceOFF:
TraceEnable:

TraceFilter0 | TraceFilter1 | TraceFilter2 | TraceFilter3

Name: Filter0 Initiator: ALL MCmd: ALL

REQinfo

	MRID[1:0]	MRT[1:0]	MRSEC	MRS	MRE	MRC	MRD[2:0]
0 = Cleared	XX	XX	X	X	X	X	XXX
1 = Set							
X = DontCare							

To display and analyze the recorded trace data, use the **OCPTrace** command group.

See also

- OCP.AutoIDLE
- OCP.DebugPort
- OCP.OFF
- OCP.ON
- OCP.Register
- OCP.RESet
- OCP.state
- OCP.TraceEnable
- OCP.TraceFilter
- OCP.TraceOFF
- OCP.TraceON
- OCP.TriggerOut<x>
- OCPTrace
- SYStem.CONFIG.OCP

OCP.AutoIDLE OCP-WP clocking strategy

Format:	**OCP.AutoIDLE** [**ON** | **OFF**]

Default: OFF.

If **ON**, the OCP-WP is gated whenever no activity can be observed on the OCP interface. If **OFF**, the OCP-WP is clocked permanently.

See also

- OCP
- OCP.state

OCP.DebugPort Select target to be traced

Format:	**OCP.DebugPort** [*<debug_port>* | *<number>*]

Default: OFF.

<debug_port>	Trace transactions to the predefined L3 target *<debug_port>*.
<number>	Trace transactions to the user-defined L3 target with ID *<number>*.

See also

- OCP
- OCP.state

OCP.OFF Switch OCP off

Format:	**OCP.OFF**

Switches the OCP component off.

See also

- OCP
- OCP.state

OCP.ON   Switch OCP on

Format:	**OCP.ON**

Switches the OCP component on.

See also

■ OCP   ■ OCP.state

OCP.Register   Display OCP registers

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

Displays the OCP registers.

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

See also

■ OCP   ■ OCP.state

OCP.RESet   Reset OCP settings to their defaults

Format:	**OCP.RESet**

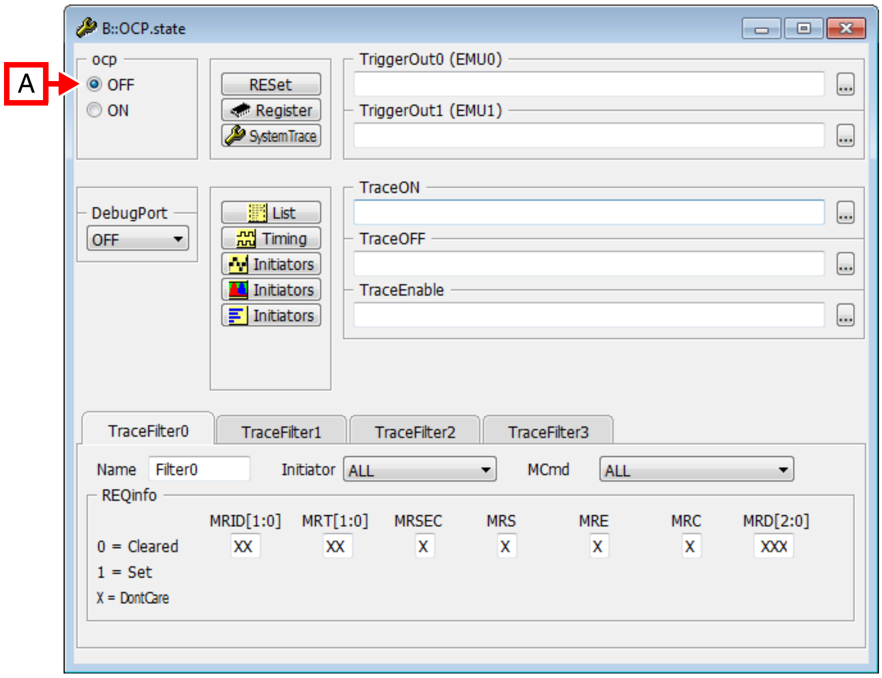
All OCP settings are reset to their defaults.

See also

■ OCP   ■ OCP.state

Format:	**OCP.state**

Shows the OCP setup window.



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

See also

- OCP
- OCP.ON
- OCP.TraceFilter
- OCP.AutoIDLE
- OCP.Register
- OCP.TraceOFF
- OCP.DebugPort
- OCP.RESet
- OCP.TraceON
- OCP.OFF
- OCP.TraceEnable
- OCP.TriggerOut<x>

See also

- OCP.TraceFilter<x>.Initiator
- OCP.TraceFilter<x>.MCmd
- OCP.TraceFilter<x>.NAME
- OCP.TraceFilter<x>.REQinfo
- OCP
- OCP.state

OCP.TraceFilter<x>.NAME Name a filter

Format: **OCP.TraceFilter**<*x*>**.NAME** <*name*>

Filters can be named in order to identify the filter a traced transaction has passed. The name of the filter can be displayed in the trace list window via **TraceOCP.List FilterName**.

Example:

```
OCP.TraceFilter0.NAME "Filter0"
```

See also

- OCP.TraceFilter

OCP.TraceFilter<x>.MCmd Filters traffic by transaction type

Format: **OCP.TraceFilter**<*x*>**.MCmd** <*command*>

Default: ALL.

Only transactions of type <*command*> will pass filter <*x*>.

See also

- OCP.TraceFilter

OCP.TraceFilter<x>.Initiator

Filters traffic by transaction initiator

Format: **OCP.TraceFilter**<*x*>**.Initiator** [**ALL** | <*initiator*>]

ALL (default)	Transactions from all initiators is traced.
<*initiator*>	Only transactions from <*initiator*> will pass filter <*x*>.

See also

■ OCP.TraceFilter

OCP.TraceFilter<x>.REQinfo

Filters traffic by transaction qualifier

Format: **OCP.TraceFilter**<*x*>**.REQinfo** <*qualifier*>.[<*value*> | <*mask*>]

Default: 0yXXX (Trace all)

<*value*>	Only trace transactions if the <*qualifier*> equals <*value*>.
<*mask*>	Alternative way to define the REQinfo filter criteria as bitmask; <*mask*> must be of format ’0ybbb’, whereas b = [0, Cleared 1, Set X]. Don’t Care

Example:

```
OCP.TraceFilter0.REQinfo MReqDomain.0y11X       ;Trace transactions which
                                                ;have the two upper bits
                                                ;set ignore the state of
                                                ;the lowest bit.
```

See also

■ OCP.TraceFilter

OCP.TraceEnable Filter OCP traffic by address range

Format: **OCP.TraceEnable** *<range>*

Default: 0x00000000-0xffffffff

OCP traffic is only captured if the address is within the specified *<range>*. The range must be specified as the offset from the base address of the selected debug port (**OCP.DebugPort**), not to the global address!
OCP.TraceEnable and **OCP.TraceON** / **OCP.TraceOFF** cannot be applied at the same time!

Example:

```
;Debug port base address = 0xa0001000
;Range to be monitored    = 0xa0001000 to 0xa0001020

OCP.TraceEnable 0x00000000--0x00000020
```

See also

- OCP
- OCP.state

OCP.TraceOFF Stop tracing

Format: **OCP.TraceOFF** [**EMU1** | *<address>*]

Stops tracing if the trigger condition or address match occurs. Tracing will continue on an **OCP.TraceON** condition.

OCP.TraceEnable and **OCP.TraceOFF** cannot be applied at the same time!
OCP.TriggerOut<x> and **OCP.TraceOFF EMU1** cannot be used at the same time!

Default: **OCP.TraceEnable**

EMU1	Stops tracing upon a HIGH-TO-LOW transition of the EMU1 trigger input.
<address>	Stops tracing upon an address match.

See also

- OCP
- OCP.state

OCP.TraceON — Start tracing

Format:	**OCP.TraceON** [**EMU0** | *<address>*]

Starts tracing if the trigger condition or address match occurs. Tracing continues even if the trigger condition or address match no longer holds.

OCP.TraceEnable and **OCP.TraceON** cannot be applied at the same time!
OCP.TriggerOut<x> and **OCP.TraceON EMU0** cannot be used at the same time!

Default: **OCP.TraceEnable**

EMU0	Starts tracing upon a HIGH-TO-LOW transition of the EMU0 trigger input.
<address>	Starts tracing upon an address match.

See also

- OCP
- OCP.state

OCP.TriggerOut<x> — Generate trigger event

Format:	**OCP.TriggerOut** [*<address* | *<range>*]

<address>	Asserts trigger EMU*<x>* if the monitored address matches *<address>*.
<range>	Asserts trigger EMU*<x>* if the monitored address is within *<range>*.

OCP.TriggerOut<x> and **OCP.TraceON EMU0** / **OCP.TraceOFF EMU1** cannot be used at the same time!

See also

- OCP
- OCP.state

Format:	**OCPTrace.**<*sub_cmd*> [<*ocp_channel*>] [<*channel*>...]

Using the **OCPTrace** command group, you can analyze and display the recorded OCP trace data. The command group consists of the name of the trace source, here **OCP**, plus the keyword **Trace** of the <*trace*> command group.

<*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 **OCPTrace.List**, refer to **<trace>.List**

<*ocp_channel*>	The following channel is relevant for the analysis of OCP trace data:
OCPFN	FilterName: Name of the filter the OCP message has passed.

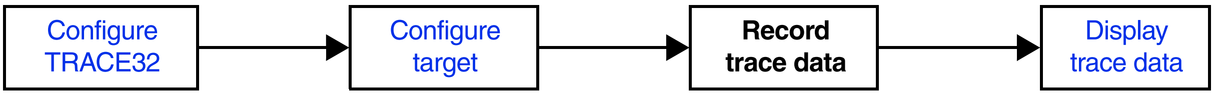
<*channel*>	For a description of the default channels, see **<trace>.List … <items>**.

Example:

```
; Display trace data
OCPTrace.List DEFault OCPFN List.NoDummy
```

See also

■ OCP



SYStem.CONFIG.PMI Inform TRACE32 about PMI component

[Process Overview]

Format:	**SYStem.CONFIG.PMI.***<sub_cmd>*
<sub_cmd>:	*<generic>* | *<component_specific>*
<component_ specific>:	**TraceID** *<id>*

If the PMI component is not enabled for your specific device, use the following commands for configuration. Both, the base address and the ID must be set in order to enable the PMI.

<generic>	For descriptions of the generic subcommands, click here.
TraceID	Sets the STM master ID of the PMI component.

Deprecated vs. New Commands:

SYStem.CONFIG.TIPMIBASE (deprecated)	**SYStem.CONFIG.PMI.Base**
SYStem.CONFIG.TIPMIID (deprecated)	**SYStem.CONFIG.PMI.TraceID**

See also

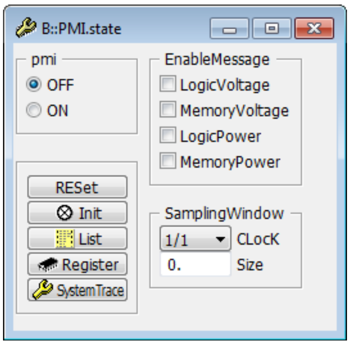
■ PMI

▲ 'Generic Subcommands, Parameters, and Options' in 'System Trace User's Guide'

[Process Overview]

The Power Management component monitors power domain state changes of other components on the OMAP4. For more detailed information refer to the OMAP4 ETRM available from **https://www-a.ti.com/extranet/programs/emulation/OMAP4_ETRM_2.0-Setup.exe**.

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


B::PMI.state

pmi: [✓ OFF] ON

EnableMessage: ☐ LogicVoltage ☐ MemoryVoltage ☐ LogicPower ☐ MemoryPower

SamplingWindow: 1/1 CLocK, 0. Size

To display and analyze the recorded trace data, use the **PMITrace** command group.

See also

- PMI.EnableMessage
- PMI.OFF
- PMI.ON
- PMI.Register
- PMI.RESet
- PMI.SamplingWindow
- PMI.state
- PMITrace
- SYStem.CONFIG.PMI

PMI.EnableMessage — Enables event message generation

Format:	**PMI.EnableMessage.***<event_msg>* [**ON** | **OFF**]
<event_msg>:	**LogicVoltage** **MemoryVoltage** **LogicPower** **MemoryPower**

Default: OFF.

Event messages are emitted in case a memory or logic block changes its voltage or power state.

LogicVoltage	Voltage levels of logic blocks.
MemoryVoltage	Voltage levels of memory blocks.
LogicPower	Power FSM states of logic blocks.
MemoryPower	Power state of memory blocks.

See also

■ PMI

PMI.OFF — Switch PMI off

Format:	**PMI.OFF**

Switches the PMI component off.

See also

■ PMI

PMI.ON — Switch PMI on

Format:	**PMI.ON**

Switches the PMI component on.

See also

■ PMI

PMI.Register — Display the PMI registers

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

Displays the PMI registers.

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

See also

■ PMI

PMI.RESet — Resets PMI settings to their defaults

Format:	**PMI.Reset**

All PMI settings are reset to their defaults.

See also

■ PMI

See also

- PMI.SamplingWindow.CLocK
- PMI.SamplingWindow.Size
- PMI

PMI.SamplingWindow.CLocK — Set sampling window clock

Format:	**PMI.SamplingWindow.CLocK** *<ratio>*

<ratio>	Divider ratio of the sampling window clock. It is derived from the PMI component’s clock. Valid ratios range from 1/1 to 1/16. Default: 1/1

See also

- PMI.SamplingWindow

PMI.SamplingWindow.Size — Set sampling window size

Format:	**PMI.SamplingWindow.Size** *<cycles>*

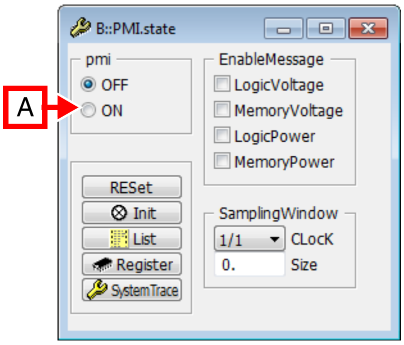
<cycles>	Size of the sampling window. Smaller windows allow for more accurate event reports while bigger sampling windows reduce trace traffic. Valid sizes range from 1 to 256. Default: 1

See also

- PMI.SamplingWindow

Format:	**PMI.state**

Shows the PMI setup window.



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

Exceptions:

- The **List** button opens the **PMITrace.List** window, see **<trace>.List**. For more information, refer to the description of the **PMITrace** command group.
- The **SystemTrace** button opens the **SystemTrace.state** window, see **<trace>.state**. For more information, refer to the description of the **SystemTrace** command group.

See also

■ PMI

PMI Example

```
PMI.RESet

PMI.SamplingWindow.Size 15

PMI.EnableMessage.LogicVoltage ON

PMI.ON

...

PMI.OFF



PMITrace.List CYcle PMILV.<domain>         ; Display logic voltage domain
                                           ; <domain> voltage level.
```

Format:	**PMITrace.**<*sub_cmd*> [<*pmi_channel*>…] [<*channel*>…]
<*pmi_ channel*>:	**PMILV.**<*domain*> | **PMILVOFF** | **PMIMV.**<*domain*> | **PMILP.**<*domain*> | **PMIMP.**<*domain*> | **PMISTAT** | **PMILAT** | **CYcle**

Using the **PMITrace** command group, you can analyze and display the recorded CMI trace data. The command group consists of the name of the trace source, here **PMI**, plus the keyword **Trace** of the <*trace*> command group.

<*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 **PMITrace.List**, refer to **<trace>.List**

<*pmi_channel*>	The following channels are relevant for the analysis of PMI trace data:
<*domain*>	A list of all valid replacements for the placeholder <*domain*> will be displayed as softkeys in TRACE32 as soon as the dot '.' is entered in the TRACE32 command line.
PMILV.<*domain*>	Voltage level of logic voltage domain <*domain*>.
PMILVOFF	OFF mode voltage domain.
PMIMV.<*domain*>	FSM state of memory voltage domain <*domain*>.
PMILP.<*domain*>	Power state of logic power domain <*domain*>.
PMIMP.<*domain*>	Power state of memory power domain <*domain*>.
PMISTAT	Error flag indicating event message loss(es) caused by an undersized sampling window.
PMILAT	Event messages: Export latency in multiples of the sampling window.
CYcle	Domain name.

<*channel*>	For a description of the default channels, see **<trace>.List … <items>**.

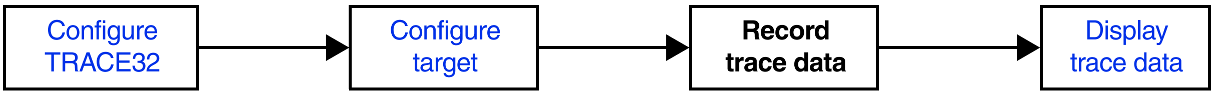
Example:

```
; Display trace data
PMITrace.List CYcle PMILP.IVAHD PMIMP.IVAHD-TCM1
```

See also

■ PMI

StatCol Component (Statistics Collector)



SYStem.CONFIG.SC Inform TRACE32 about StatCol component

[Process Overview]

Format:	**SYStem.CONFIG.SC.***<sub_cmd>*
<sub_cmd>:	*<generic>* | *<component_specific>*
<component_ specific>:	**TraceID** *<id>*

If the statistics collector is not enabled for your specific device, use the following commands allow for configuration. Both, the base address and the ID must be set in order to enable the statistics collector.

<generic>	For descriptions of the generic subcommands, click here.
TraceID	Set the STM master ID of the statistics collector.

Deprecated vs. New Commands:

SYStem.CONFIG.TISCID (deprecated)	**SYStem.CONFIG.SC.TraceID**

See also

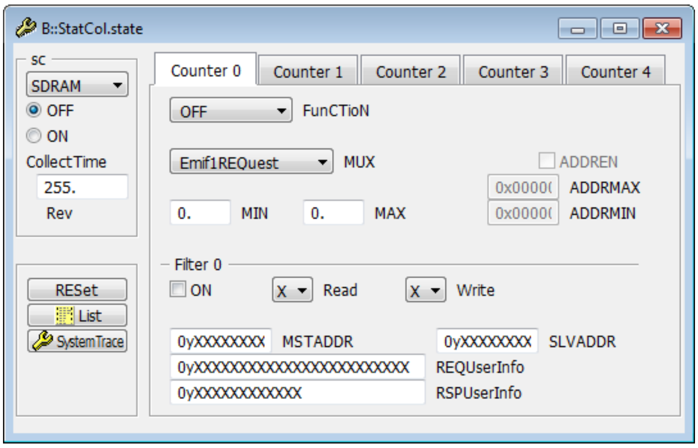
■ StatCol

▲ 'Generic Subcommands, Parameters, and Options' in 'System Trace User's Guide'

The NoC statistics collector provides information about the workload of an onchip bus system like throughput, latency, etc. For each bus system there is a separate implementation of the statistics collector (called 'probe'), hence all the commands listed in the following will affect the selected probe only, except for the **StatCol.RESet** command.

Configuring the statistics collector requires an in-depth knowledge of its structure and modes of operations. For those who do not have that knowledge or don't need to make use of the full extent of the statistics collector's features there are macro functions available. These set up most of the required configurations and are explained in chapter StatCol macro functions.

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



SC: SDRAM
[✓ OFF] ON
CollectTime: 255.
Rev
OFF FunCTioN
Emif1REQuest MUX
ADDREN
0x0000(ADDRMAX
0x0000(ADDRMIN
0. MIN 0. MAX
ON X Read X Write
0yXXXXXXXX MSTADDR 0yXXXXXXXX SLVADDR
0yXXXXXXXXXXXXXXXXXXXXXXXX REQUserInfo
0yXXXXXXXXXXXX RSPUserInfo

To display and analyze the recorded trace data, use the **StatColTrace** command group.

See also

- StatCol.<probe>.CollectTime
- StatCol.<probe>.Counter
- StatCol.<probe>.OFF
- StatCol.<probe>.ON
- StatCol.<probe>.REQuestEVenT
- StatCol.<probe>.ReSPonseEVenT
- StatCol.RESet
- StatCol.state
- StatColTrace
- SYStem.CONFIG.SC

StatCol.RESet   Resets all statistics collector settings to their default

Format: **StatCol.RESet**

All statistics collector settings are reset to their defaults.

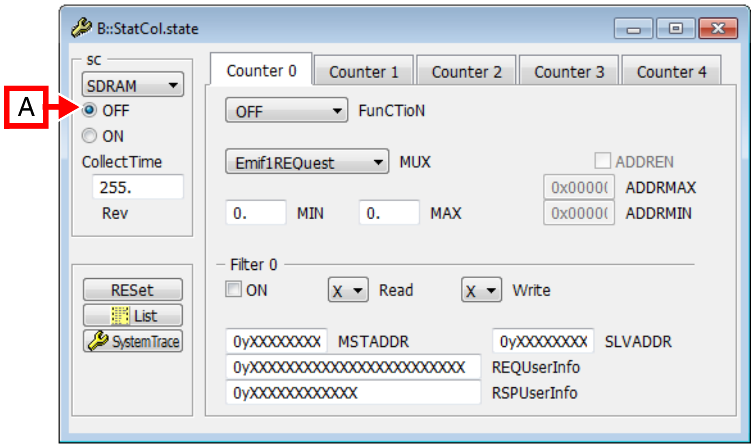
See also

- StatCol
- StatCol.state

StatCol.state   Display statistics collector settings

Format: **StatCol.state**

Shows the statistics collector setup window.



SDRAM
OFF [✓]
ON
CollectTime 255.
Rev

OFF FunCTioN
Emif1REQuest MUX
ADDREN
0x0000(ADDRMAX
0x0000(ADDRMIN
0. MIN
0. MAX

ON
X Read
X Write
0yXXXXXXXX MSTADDR
0yXXXXXXXX SLVADDR
0yXXXXXXXXXXXXXXXXXXXXXXXXX REQUserInfo
0yXXXXXXXXXXXXX RSPUserInfo

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

Exceptions:

- The **List** button opens the **StatColTrace.List** window, see **<trace>.List**. For more information, refer to the description of the **StatColTrace** command group.
- The **SystemTrace** button opens the **SystemTrace.state** window, see **<trace>.state**. For more information, refer to the description of the **SystemTrace** command group.

See also

- StatCol
- StatCol.<probe>.Counter
- StatCol.<probe>.ON
- StatCol.<probe>.ReSPonseEVenT
- StatCol.<probe>.CollectTime
- StatCol.<probe>.OFF
- StatCol.<probe>.REQuestEVenT
- StatCol.RESet

StatCol.<probe>.OFF

Switch probe off

Format:	**StatCol.***<probe>***.OFF**

Switches the probe off.

See also

- StatCol
- StatCol.state

StatCol.<probe>.ON

Switch probe on

Format:	**StatCol.***<probe>***.ON**

Switches the probe on.

See also

- StatCol
- StatCol.state

Format:	**StatCol.***<probe>***.REQuestEVenT.***<event>*
<event>:	**NONE** **ANY** **TRANSFER** **WAIT** **BUSY** **PAKET** **DATA** **IDLES** **LATENCY**

Default: NONE.

Selects the event detector for the probe's request link.

NONE	Do not detect any events.
ANY	Detect all events.
TRANSFER	Detect NTTP cell or OCP data/command transfers.
WAIT	Detect WAIT cycles (NTTP only).
BUSY	Detect BUSY cycles.
PAKET	Detect packet headers or OCP commands.
DATA	Detect payload transfers.
IDLES	Detect idle cycles.
LATENCY	Apply latency measurement.

See also

■ StatCol ■ StatCol.state

StatCol.<probe>.ReSPonseEVenT

Select event detector

Format: **StatCol.**<*probe*>**.REQuestEVenT.**<*event*>

Default: NONE.

Selects the event detector for the probe's response link. See **StatCol.<probe>.REQuestEVenT**.

See also

■ StatCol	■ StatCol.state

StatCol.<probe>.CollectTime

Set up collection period

Format: **StatCol.**<*probe*>**.CollectTime** <*cycles*>

Default: 255.

Sets up the time interval in cycles after which the internal counters are reset and the result is sent to the STM.

See also

■ StatCol	■ StatCol.state

See also

- StatCol.<probe>.Counter <counter> ADDREN
- StatCol.<probe>.Counter <counter> ADDRMAX
- StatCol.<probe>.Counter <counter> ADDRMIN
- StatCol.<probe>.Counter <counter> EventInfo
- StatCol.<probe>.Counter <counter> Filter
- StatCol.<probe>.Counter <counter> FunCTioN
- StatCol.<probe>.Counter <counter> MAX
- StatCol.<probe>.Counter <counter> MIN
- StatCol.<probe>.Counter <counter> SELect
- StatCol
- StatCol.state

StatCol.<probe>.Counter <counter> ADDRMAX Filter max address

Format: **StatCol.***<probe>***.Counter** *<counter>* **ADDRessMAX.***<value>*

Default: 0.

Sets the upper bound for address filtering. See **StatCol<probe>.Counter <counter> ADDRessENable**. This command is available for certain CPUs only.

See also

- StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> ADDRMIN Filter min address

Format: **StatCol.***<probe>***.Counter** *<counter>* **ADDRessMIN.***<value>*

Default: 0.

Sets the lower bound for address filtering. See **StatCol<probe>.Counter <counter> ADDRessENable**. This command is available for certain CPUs only.

See also

- StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> ADDREN Enable address filtering

Format:	**StatCol.***<probe>***.Counter** *<counter>* **ADDRessENable** [**ON** | **OFF**]

Default: OFF.

Only generates statistic data if address on bus is smaller than **ADDRessMAX** and greater than **ADDRessMIN**. This command is available for certain CPUs only.

See also

■ StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> EventInfo Select 'EventInfo' to count

Format:	**StatCol.***<probe>***.Counter** *<counter>* **EventInfo.***<eventinfo>*
<eventinfo>:	**LENgth** **PRESsure** **LATency**

Default: LENgth.

Detects additional event information:

LENgth	Payload length.
PRESsure	Link pressure.
LATency	Transfer latency.

See also

■ StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> MAX Set max threshold for events

Format: **StatCol.***<probe>***.Counter** *<counter>* **MAX.***<value>*

Default: 0.

Increments *<counter>* if **StatCol.<probe>.Counter <counter> SELect.MINMAX** is selected and the defined **EventInfo** is within **Max.***<value>* and **Min.<value>**.

See also

- StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> MIN Set min threshold for events

Format: **StatCol.***<probe>***.Counter** *<counter>* **MIN.***<value>*

Default: 0.

Increments *<counter>* if **StatCol.<probe>.Counter <counter> SELect.MINMAX** is selected and the defined **EventInfo** is within **Min.***<value>* and **Max.<value>**.

See also

- StatCol.<probe>.Counter

Format:	**StatCol.**<*probe*>**.Counter** <*counter*> **SELect.**<*input*>
<*input*>:	**HIT** **MINMAX** **ADD** **AND** **OR** **REQ** **RSP** **ALL** **EXT**

Default: HIT.

Defines what kind of statistics the counter will count:

HIT	Increment the counter by one each time an event has passed the counter's filter. (See **StatCol.<probe>.Counter <counter> Filter <filter>** commands).
MINMAX	Increment the counter by one each time the selected EventInfo is within the range **Min.<value>** and **Max.<value>**.
ADD	Add the selected EventInfo value to the counter if an event has passed the counter's filter. (See **StatCol.<probe>.Counter <counter> Filter <filter>** commands)
AND	Increment the counter by one if an event has passed all filters of <*probe*>.
OR	Increment the counter by one if an event has passed at least one of all filters of <*probe*>.
REQ	Increment the counter by one each time a request message is detected on any port of <*probe*>.
RSP	Increment the counter by one each time a response message is detected on any port of <*probe*>.
ALL	Increment the counter by one each time a response or request message is detected on any port of <*probe*>.
EXT	Increment the counter by one each time the external event input is sampled high.

See also

■ StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> Filter

Format:	**StatCol.***<probe>***.Counter** *<counter>* **Filter** *<filter> <item>*.[*<value>* | *<mask>*]
<item>:	**MaSTerADDRess** **ReaD** **WRite** **ERRor** **REQuestUserInfo** **ReSPonseUserInfo** **SLaVeADDRess**

Filters out packets which do not comply with the defined item bitmask or value.

MaSTerADDRess	Master address (NTTP) or MConnId (OCP).
ReaD	Read bit.
WRite	Write bit.
ERRor	Error bit (NTTP only).
REQuestUserInfo	RequestUserInfo bits (NTTP only).
ReSPonseUserInfo	ResponseUserInfo bits (NTTP only).
SLaVeADDRess	Slave address (NTTP only).

<value>	(Hexa)decimal, octal or binary value that defines the required packet item.	
<mask> (default: Don't care)	Bitmask of format '0y......':	x = don't care 1 = set 0 = cleared

Example:

```
MaSTerADDRess.0yxxxx11        ;Only packets with the lower two bits set of
                              ;the master address will pass the filter
                              ;element.
```

See also

- StatCol.<probe>.Counter <counter> Filter <filter> MUX
- StatCol.<probe>.Counter <counter> Filter <filter> ON
- StatCol.<probe>.Counter <counter> Filter <filter> OFF
- StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> Filter <filter> MUX Input port

Format:	**StatCol.***<probe>***.Counter** *<counter>* **Filter** *<filter>* **MUX** *<input>*

Selects one of the probe's inputs as the input for the specified filter. Available inputs are depended on the probe.

See also

■ StatCol.<probe>.Counter <counter> Filter

StatCol.<probe>.Counter <counter> Filter <filter> OFF Switch filter off

Format:	**StatCol.***<probe>***.Counter** *<counter>* **Filter** *<filter>* **OFF**

Switches the filter off.

See also

■ StatCol.<probe>.Counter <counter> Filter

StatCol.<probe>.Counter <counter> Filter <filter> ON Switch filter on

Format:	**StatCol.***<probe>***.Counter** *<counter>* **Filter** *<filter>* **ON**

Switches the filter on.

See also

■ StatCol.<probe>.Counter <counter> Filter

Format:	**StatCol.***<probe>***.Counter** *<counter>* **FunCTioN** *<macro>*
<macro>:	**OFF** **AvgPayloadLength** **THRoughput** **LnkOcc** **ArbConf** **TransUflow** **IBusy** **HistPayloadLen** **HistPresDist** **HistLatDist** **AvgLatDist**

Macro functions set up the selected probe for common statistics and allow for only few (optional) additional configuration. Therefore they are best suited for users with only little knowledge of the statistics collector or for non-complex statistics tracing scenarios.

Of course macro functions do not make use of the entire feature set of the statistics collector probes. The following limitations apply when using macro functions only:

Every counter can be assigned to exactly ONE macro function. A counter can not be used for multiple macro functions. That means that the number of available macro functions depends on the number of available counters of the selected probe. A counter is assigned to a macro function by the StatCol.*<probe>*.Counter *<counter>* FunCTioN *<macro>* command.

Only the first available filter element of a filter will be used for filtering. Any second (or third, ...) filter elements of a filter are disabled by default. Advanced users may enable and configure those additional filter elements to set up more complex filtering criteria. This may involve overwriting some configurations made by the macro functions, hence the recommended sequence is to first select the macro function and then to set up additional filtering criteria via the **StatCol<probe>.Counter <counter> Filter <element> <item>** commands.

OFF

Clears all filters..

Mandatory additional configuration	-
Optional additional configuration	-

AvgPayloadLength

Average payload length: Outputs the average payload length in bytes of request transfers.

Mandatory additional configuration	**StatCol.<probe>.Counter <counter> MUX**
Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDRess**
	StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess
	StatCol.<probe>.Counter <counter> Filter <element> ReaD
	StatCol.<probe>.Counter <counter> Filter <element> WRite
	StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo

THRoughput

Payload per cycle: Outputs the payload in bytes per cycle.

Mandatory additional configuration	**StatCol.<probe>.Counter <counter> MUX**
	StatCol.<probe>.CollectTime
Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDRess**
	StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess
	StatCol.<probe>.Counter <counter> Filter <element> ReaD
	StatCol.<probe>.Counter <counter> Filter <element> WRite
	StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo

LnkOcc

Link occupancy: Percentage of non-idle cycles.

Mandatory additional configuration	**StatCol.<probe>.CollectTime**
Optional additional configuration	-

ArbConf

Arbitration conflicts: Percentage of busy cycles caused by a target which cannot accept further write transactions from the initiator.

Mandatory additional configuration	**StatCol.<probe>.CollectTime**
Optional additional configuration	-

TransUflow

Transaction underflow: Percentage of wait cycles (The initiator is not able to send as much data as requested by the target).

Mandatory additional configuration	**StatCol.<probe>.CollectTime**
Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDRess**
	StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess
	StatCol.<probe>.Counter <counter> Filter <element> ReaD
	StatCol.<probe>.Counter <counter> Filter <element> WRite
	StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo

IBusy

Initiator busy: Percentage of busy cycles caused by an initiator which cannot accept further read data from the target.

Mandatory additional configuration	**StatCol.<probe>.CollectTime**
Optional additional configuration	-

HistPayloadLen

Histogram of payload length: Filter packets by means of payload length. A histogram can be obtained by assigning the HPL macro to different counters with different min / max values.

Mandatory additional configuration	**StatCol.<probe>.Counter <counter> MIN** (Minimum payload length in bytes)
	StatCol.<probe>.Counter <counter> MAX (Maximum payload length in bytes)
Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDRess**
	StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess
	StatCol.<probe>.Counter <counter> Filter <element> ReaD
	StatCol.<probe>.Counter <counter> Filter <element> WRite
	StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo

HistPresDist

Histogram of pressure distribution: Filter packets by priority. A histogram can be obtained by assigning the HPD macro to different counters with different min / max values.

Mandatory additional configuration	**StatCol.<probe>.Counter <counter> MIN** (Minimum pressure)
	StatCol.<probe>.Counter <counter> MAX (Maximum pressure)
Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess**
	StatCol.<probe>.Counter <counter> Filter <element> ReaD
	StatCol.<probe>.Counter <counter> Filter <element> WRite
	StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo

HistLatDist

Histogram of latency distribution: Filter read packets by latency. A histogram can be obtained by assigning the HistLatDist macro to different counters with different min / max values.

Mandatory additional configuration	**StatCol.<probe>.Counter <counter> MIN** (Minimum latency)
	StatCol.<probe>.Counter <counter> MAX (Maximum latency)

Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess** **StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDRess** **StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo**

AvgLatDist

Average latency distribution: Output average latency of read transactions in latency / cycle.

Mandatory additional configuration	-
Optional additional configuration	**StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDRess** **StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDRess** **StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo**

See also

■ StatCol.<probe>.Counter

StatCol Example

This example shows how to gather throughput statistics of the EMIF1 request port on an OMAP4. EMIF1 is monitored by the SDRAM probe on OMAP4.

```
StatCol.RESet

CAnalyzer.AutoArm OFF                          ; The statistics collector
                                               ; outputs data periodically
                                               ; so we must arm the

CAnalyzer.ARM                                  ; CAnalyzer before the
                                               ; statistics collector is
                                               ; activated.


StatCol.SDRAM.Counter 0 FunCTion THRoughput

StatCol.SDRAM.Counter 0 Filter MUX.Emif1REQuest

StatCol.SDRAM.CollectTime 255.

StatCol.SDRAM.ON

...

StatCol.SDRAM.OFF                              ; Turn off the statistics
                                               ; collector ...

CAnalyzer.OFF                                  ; ... before shutting down
                                               ; the CAnalyzer.


StatColTrace.List CYcle SCC0                   ; Display value of counter 0
```

Format:	**StatColTrace.**<*sub_cmd*> [<*statcol_channel*>…] [<*channel*>…]
<*statcol_ channel*>:	**SCC0.**<*probe*> | **SCC1.**<*probe*> | **SCC2.**<*probe*> | **SCC3.**<*probe*> | **SCC4.**<*probe*>

Using the **StatColTrace** command group, you can analyze and display the recorded CMI trace data. The command group consists of the name of the trace source, here **StatCol**, plus the keyword **Trace** of the <*trace*> command group.

<*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 **StatColTrace.List**, refer to **<trace>.List**

<*statcol_channel*>	The following channels are relevant for the analysis of StatCol trace data:
<*probe*>	A list of all valid replacements for the placeholder <*probe*> will be displayed as softkeys in TRACE32 as soon as the dot '.' is entered in the TRACE32 command line.
SCC0.<*probe*>	Value of statistics collector counter 0.
SCC1.<*probe*>	Value of statistics collector counter 1.
SCC2.<*probe*>	Value of statistics collector counter 2.
SCC3.<*probe*>	Value of statistics collector counter 3.
SCC4.<*probe*>	Value of statistics collector counter 4.

<*channel*>	For a description of default channels, see **<trace>.List … <items>**.

Example:

```
; Display trace data
StatColTrace.List CYCle SCC0 SCC1 SCC2 SCC3
```

See also

■ StatCol

Generic Subcommands, Parameters, and Options

This section describes the *<generic>* subcommands, parameters, and options that are common to the **SYStem.CONFIG.<component>** commands.

SYStem.CONFIG.<component>.<generic>

Format:	**SYStem.CONFIG.***<component>.<generic>*
<comp.t>:	**CMI** | **OCP** | **PMI** | **SC** | **STM** | **CMN**
<generic>:	**Base** *<parameter>* **Name** [**/***<option>*] **RESet** **view** [*<parameter>*]

<component>	Click a blue *<component>* name to jump to the respective **SYStem.CONFIG.<component>** command: CMI, OCP, PMI, SC, STM, CMN.
<generic>	Generic subcommands of the **SYStem.CONFIG.<component>** commands. For descriptions of the generic subcommands, see: • **SYStem.CONFIG.<component>.Base** • **SYStem.CONFIG.<component>.Name** • **SYStem.CONFIG.<component>.RESet** • **SYStem.CONFIG.<component>.view**

SYStem.CONFIG.<component>.Base

Base address of a component

Format:	**SYStem.CONFIG.***<component>***.Base** *<parameter>*
<parameter>:	**NONE** | *<address>*

Sets the base *<address>* of the *<component>*.

NONE	Removes the base address of the *<component>*.

SYStem.CONFIG.<component>.Name

Name of a component

Format:	**SYStem.CONFIG.***<component>***.Name** *<name>* [**/***<option>*]
<option>:	**CORE** *<number>* | **CONTinue**

Assigns a user-defined name to a component.

<name>	**Parameter Type**: String. User-defined names for *<components>* allow you to distinguish between different instances having the same parameters, such as the same addresses on different buses.
CORE	Sets partial addresses.
CONTinue	Collects calls without triggering any action until the next call without the **CONTinue** parameter.

SYStem.CONFIG.<component>.RESet

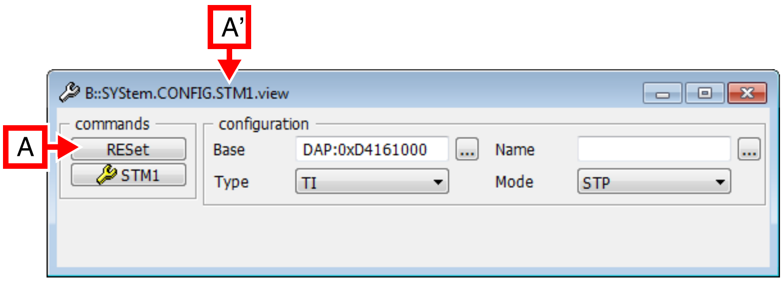
Reset of a component

Format:	**SYStem.CONFIG.***<component>***.RESet**

Resets the settings of the *<component>*.

Format:	**SYStem.CONFIG.***<component>***.view**

Opens the **SYStem.CONFIG.<component>.view** window, displaying the settings of the *<component>*.



Base: DAP:0xD4161000
Type: TI
Name:
Mode: STP

A For description of the commands in a **SYStem.CONFIG.<component>.view** window, refer to the subcommands of the respective component.

Example: For information about **RESet** for the component **STM**, see **SYStem.CONFIG.<component>.RESet**.

FAQ

Please refer to https://support.lauterbach.com/kb.