System Trace User’s Guide

TRACE32 Online Help
TRACE32 Directory
TRACE32 Index

TRACE32 Documents .............................................................................................................
System Trace .........................................................................................................................
System Trace User’s Guide .................................................................................................
History .................................................................................................................................
Introduction ...........................................................................................................................
Installation ............................................................................................................................... Software Installation
 Hardware Installation
 CombiProbe-ARM (LA-4502)
 PowerTrace II (LA-769x) + AutoFocus II Preprocessor for ARM (LA-7992)
Utilization of the STM ...........................................................................................................
Start-up Script
 Example STP Data to Onchip Buffer
 Example STP Data to Parallel Trace Interface
 Example Attach to Onchip Trace
 Example Save/Load STP Data Embedded in CoreSight Trace Stream
STM Component - Configuration of TRACE32 .................................................................
SYStem.CONFIG.STM Inform TRACE32 about STM component
SYStem.CONFIG.STM.Mode STPv2 timestamp format
SYStem.CONFIG.STM.Type Configure STM type in TRACE32
STM Component - General Target Configuration ............................................................
STM Configure STM component on target
 STM.FilterMasters Display specified masters only
 STM.FilterChannels Display specified channels only
 STM.Init Initialize trace hardware
 STM.OFF Switch STM off
 STM.ON Switch STM on
 STM.PortEndianness Select port endianness
 STM.PortMode Select STM port modes
 STM.PortRoute Select output of STP data
 STM.PortSize Port size in bits
 STMPrintfTraceFormat Define format of printftrace style messages
 STM.Register Display STM register

©1989-2019 Lauterbach GmbH
STM.RESet
Reset STM settings 25
STM.SetMaster
Set master ID manually 25
STM.state
Display STM settings 26
STM.SyncTime
Trace decoder resync time 27
STM.SyncPeriod
Add synchronization packets 27
STM.TimeStamps
Enable timestamps 27
STM.TimeStampCLOCK
Configure debugger for STM timestamp clock 28

**STM Component - TI specific Target Configuration** .............................................. 29
STM.HWMasters
Enable hardware masters for tracing 29
STM.IdleCount
Maximum idle packets 29
STM.IgnoreHeader
Ignore leading dword in printftrace message 30
STM.SWMasters
Enable software masters for tracing 30
STM.OCPAutoldle
Set OCP idle clock behavior 31
STM.PATTERN
Enable test pattern generator 31
STM.ChannelRepeat
Period of channel packet insertion 32
STM.MasterRepeat
Period of master packet insertion 32

**STM Component - CoreSight specific Target Configuration** ................................ 33
STM.DMArequests
DMA requests enable 33
STM.COMpression
Data compression enable 33
STM.EventMASK
Mask hardware event inputs 34
STM.PortMASK
Mask stimulus ports 34
STM.TraceID
Sets trace ID 35
STM.TracePriority
Set priority for STM manually 35

**STM<trace> - Trace Data Analysis** ........................................................................ 36
STM<trace>
Command groups for STM<trace> recording and analysis 36
Overview STM<trace>
36
STMAnalyzer
Analyze STM data recorded by TRACE32 PowerTrace 37
STMCAnalyzer
Analyze STM data recorded by TRACE32 CombiProbe 38
STMHAnalyzer
Display and analyze STM data recorded by the host 39
STMLA
Display and analyze STM data from binary file 39
STMMonchip
Display and analyze STM data stored on target memory 40
STMTTrace
Method-independent display and analysis of STM trace data 41

**PrintfTrace** .......................................................................................................... 43
PrintfTrace
Decoder for STP-based software messages 43

**CMI Component** .................................................................................................. 46
SYStem.CONFIG.CMI
Inform TRACE32 about CMI component 46
CMI
Configure CMI component on target 47
CMI.EnableMessage
Enables event or activity message generation 48
CMI.Mode
Set event or activity mode 49
CMI.OFF
Switch CMI off 49
CMI.ON
Switch CMI on 49
CMI.Register
Display the CMI register 50
CMI.RESet Resets CMI settings to their defaults 50
CMI.SamplingWindow Sampling window 51
CMI.SamplingWindow.CLock Set sampling window ratio 51
CMI.SamplingWindow.Size Set sampling window size 51
CMI.state Display CMI settings 52
CMI Example 53
CMITrace Display and analyze CMI trace data 54

CPTTracer Component

CPTTracer Configure CPTTracer component on target 56
CPTTracer.RESet Resets CPT settings to their defaults 57
CPTTracer.state Display CPT settings 57
CPTTracer.TraceID Set ATB ID 58
CPTTracer.<aggregator>.ON Switch aggregator on 58
CPTTracer.<aggregator>.OFF Switch aggregator off 58
CPTTracer.<aggregator>.SYNC Sync period of aggregator 59
CPTTracer.<aggregator>.<probe>.ADDressLOW Lower filter address 59
CPTTracer.<aggregator>.<probe>.ADDressHIGH Upper filter address 60
CPTTracer.<aggregator>.<probe>.CChannel Filter by channel ID 60
CPTTracer.<aggregator>.<probe>.OPeration Mode of operation 61
CPTTracer.<aggregator>.<probe>.PERiod Set period of sample window 61
CPTTracer.<aggregator>.<probe>.RouteID Filter by route ID 62
CPTTracer Example 63
CPTTracerTrace Display and analyze CPT trace data 64

OCP Component

SYStem.CONFIG.OCP Inform TRACE32 about OCP component 66
OCP Configure OCP component on target 67
OCP.AutoIDLE OCP-WP clocking strategy 68
OCP.DebugPort Select target to be traced 68
OCP.OFF Switch OCP off 68
OCP.ON Switch OCP on 69
OCP.Register Display OCP registers 69
OCP.RESet Reset OCP settings to their defaults 69
OCP.state Display OCP settings 70
OCP.TraceFilter Set filter criteria 71
OCP.TraceFilter<x>.NAME Name a filter 71
OCP.TraceFilter<x>.MCmd Filters traffic by transaction type 71
OCP.TraceFilter<x>.Initiator Filters traffic by transaction initiator 72
OCP.TraceFilter<x>.REQinfo Filters traffic by transaction qualifier 72
OCP.TraceEnable Filter OCP traffic by address range 73
OCP.TraceOFF Stop tracing 73
OCP.TraceON Start tracing 74
OCP.TriggerOut<x> Generate trigger event 74
OCPTrace Display and analyze OCP trace data 75
PMI Component ....................................................................................................... 76
SYStem.CONFIG.PMI Inform TRACE32 about PMI component 76
PMI Configure PMI component on target 77
PMI.EnableMessage Enables event message generation 78
PMI.OFF Switch PMI off 78
PMI.ON Switch PMI on 79
PMI.Register Display the PMI registers 79
PMI.RESet Resets PMI settings to their defaults 79
PMI.SamplingWindow Sampling window 80
PMI.SamplingWindow.CLocK Set sampling window clock 80
PMI.SamplingWindow.Size Set sampling window size 80
PMI.state Display PMI settings 81
PMI Example 82
PMITrace Display and analyze PMI trace data 83

StatCol Component (Statistics Collector) ............................................................................ 85
SYStem.CONFIG.SC Inform TRACE32 about StatCol component 85
StatCol Configure StatCol component on target 86
StatCol.RESet Resets all statistics collector settings to their default 87
StatCol.state Display statistics collector settings 87
StatCol.<probe>.OFF Switch probe off 88
StatCol.<probe>.ON Switch probe on 88
StatCol.<probe>.REQuestEVenT Select event detector 89
StatCol.<probe>.ReSPonseEVenT Select event detector 90
StatCol.<probe>.CollectTime Set up collection period 90
StatCol.<probe>.Counter Counter configuration 91
StatCol.<probe>.Counter <counter> ADDRMAX Filter max address 91
StatCol.<probe>.Counter <counter> ADDRMIN Filter min address 91
StatCol.<probe>.Counter <counter> ADDRREN Enable address filtering 92
StatCol.<probe>.Counter <counter> EventInfo Select ‘EventInfo’ to count 92
StatCol.<probe>.Counter <counter> MAX Set max threshold for events 93
StatCol.<probe>.Counter <counter> MIN Set min threshold for events 93
StatCol.<probe>.Counter <counter> SELect Set counter input 94
StatCol.<probe>.Counter <counter> Filter Set filter criteria 95
StatCol.<probe>.Counter <counter> Filter <filter> MUX Input port 96
StatCol.<probe>.Counter <counter> Filter <filter> OFF Switch filter off 96
StatCol.<probe>.Counter <counter> Filter <filter> ON Switch filter on 96
StatCol Macro Functions 97
AvgPayloadLength 97
TRHoughput 97
LnkOcc 98
ArbConf 98
TransUflow 98
IBusy 99
HistPayloadLen 99
HistPresDist 99
HistLatDist 100
AvgLatDist 100
StatCol Example 101
StatColTrace Display and analyze StatCol trace data 102

Generic Subcommands, Parameters, and Options .............................................................. 103

System.CONFIG.<component>.<generic> 103
System.CONFIG.<component>.Base Base address of a component 104
System.CONFIG.<component>.Name Name of a component 104
System.CONFIG.<component>.RESet Reset of a component 104
System.CONFIG.<component>.view Display component settings 105

FAQ ................................................................................................................. .......................... 106
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. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>The term <em>module</em> is primarily reserved for the hardware modules of TRACE32, such as PowerTrace, PowerDebug, CombiProbe.</td>
</tr>
</tbody>
</table>

**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).
Utilization of the STM

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 !!!!!!!!
Onchip.AutoArm OFF ; If AutoArm remain on, several 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
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
; dpmemu[0:4] pads or multiplex
; emu[15:19] signals to dpmemu[15:19]
; pads. You can even do both.

; dpmemu[0:4]
Data.Set ahb:0x4a1001ac %long Data.Set ahb:0x4a1001ac %long
Data.Set ahb:0x4a1001b0 %long 0
Data.Set ahb:0x4a1001b4 %long 0

; dpmemu[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

```verbatim
; 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

```verbatim
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
```
### STM Component - Configuration of TRACE32

**SYStem.CONFIG.STM**

Inform TRACE32 about STM component

<table>
<thead>
<tr>
<th>Format:</th>
<th>SYStem.CONFIG.STM[&lt;instance&gt;] &lt;sub_cmd&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;instance&gt;:</td>
<td>1</td>
</tr>
<tr>
<td>&lt;sub_cmd&gt;:</td>
<td>&lt;generic&gt;</td>
</tr>
<tr>
<td>&lt;component_specific&gt;:</td>
<td>ACCESS [Denied</td>
</tr>
<tr>
<td></td>
<td>Type &lt;type&gt;</td>
</tr>
<tr>
<td></td>
<td>Mode &lt;mode&gt;</td>
</tr>
<tr>
<td></td>
<td>AutoSync [ON</td>
</tr>
</tbody>
</table>

Default: Device specific.

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

<table>
<thead>
<tr>
<th>&lt;instance&gt;</th>
<th>For a description of &lt;instance&gt;, refer to the introduction to the command group STM.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;generic&gt;</td>
<td>For descriptions of the generic subcommands, click here.</td>
</tr>
<tr>
<td>STM</td>
<td>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.</td>
</tr>
<tr>
<td>STM1</td>
<td>Same as STM command. Used to differentiate between STM1 and STM2.</td>
</tr>
<tr>
<td>STM2</td>
<td>Used to configure a 2nd STM, if present.</td>
</tr>
<tr>
<td>ACCESS [Denied</td>
<td>ReadWrite]</td>
</tr>
<tr>
<td>AutoSync [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>Mode &lt;mode&gt;</td>
<td>For details, see SYStem.CONFIG.STM.Mode.</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Type &lt;type&gt;</td>
<td>For details, see SYStem.CONFIG.STM.Type.</td>
</tr>
</tbody>
</table>

**Example 1:**

<table>
<thead>
<tr>
<th>System.CONFIG STM Mode STP64</th>
<th>; chip contains a STM that uses MIPI STPv1 (D64) protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM.state</td>
<td>; open STM configuration window</td>
</tr>
</tbody>
</table>

**Example 2:**

<table>
<thead>
<tr>
<th>System.CONFIG STM1 Mode STPv2</th>
<th>; chip contains a STM that uses MIPI STPv2 protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM1.state</td>
<td>; open STM1 configuration window</td>
</tr>
<tr>
<td>System.CONFIG STM2 Mode STPv2</td>
<td>; chip contains a second STM that uses MIPI STPv2 protocol</td>
</tr>
<tr>
<td>STM2.state</td>
<td>; open STM2 configuration window</td>
</tr>
</tbody>
</table>

**See also**

- SYStem.CONFIG.STM.Mode
- SYStem.CONFIG.STM.Type
- SYStem.CONFIG
- ‘COMPonent Functions’ in ‘General Function Reference’
- ‘Generic Subcommands, Parameters, and Options’ in ‘System Trace User’s Guide’
SYStem.CONFIG.STM.Mode

STPv2 timestamp format

<table>
<thead>
<tr>
<th>Format:</th>
<th>SYStem.CONFIG.STM.Mode &lt;mode&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;:</td>
<td>STP</td>
</tr>
</tbody>
</table>

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
SYstem.CONFIG.STM.Type

Configure STM type in TRACE32

<table>
<thead>
<tr>
<th>Format:</th>
<th>SYstem.CONFIG.STM.Type &lt;type&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;type&gt;:</td>
<td>None</td>
</tr>
</tbody>
</table>

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 Component - General Target Configuration

STM

Configure STM component on target

Format: \texttt{STM[<instance>].<sub_cmd>}

\texttt{<instance>:} \hspace{1em} 1 \mid 2

The \texttt{STM<instance>} command group is used to configure the STM trace source.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instance of the STM trace source 1. Most targets have only one STM trace source. For these targets, the commands \texttt{STM.&lt;sub_cmd&gt;} and \texttt{STM1.&lt;sub_cmd&gt;} are aliases. This means, if you are configuring a target with only one STM trace source, you may omit the instance number 1.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
</tbody>
</table>

\texttt{<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 \texttt{STM.state} window.

![STM.state window](image)

To display and analyze the recorded trace data, use the \texttt{STM<trace>} command groups.

See also

- STM.ChannelRepeat
- STM.FilterChannels
- STM.IgnoreHeader
- STM.OFF
- STM.PortMASK
- STM.PrintTraceFormat
- STM.state
- STM.TimeStampCLOCK
- SystemTrace

- STM.COMPression
- STM.FilterMasters
- STM.Init
- STM.ON
- STM.PortMode
- STM.Register
- STM.SWMasters
- STM.TimeStamps

- STM.DMArequests
- STM.HWmasters
- STM.MasterRepeat
- STM.PATTERN
- STM.PortRoute
- STM.RESet
- STM.SyncPeriod
- STM.TraceID

- STM.EventMASK
- STM.IdleCount
- STM.OCPAutoIdle
- STM.PortEndianness
- STM.PortSize
- STM.SetMaster
- STM.SyncTime
- STM.TracePriority

©1989-1999 Lauterbach GmbH
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
STM.PortEndianness

Select port endianness

| 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
### STM.PortMode

Select STM port modes

<table>
<thead>
<tr>
<th>Format:</th>
<th>STM[&lt;instance&gt;].PortMode &lt;mode&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;:</td>
<td>Continuous</td>
</tr>
</tbody>
</table>

### STM (TI)

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

For STM (CS): ETM.PortMode.

### See also

- STM
- STM.state
### STM.PortRoute

Select output of STP data

| 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. | Data are directed to the Parallel Trace Interface (PTI) and recorded by the PowerTrace II. | 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
STM.PortSize

Port size in bits

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:

<table>
<thead>
<tr>
<th></th>
<th>no suffix (standard configuration)</th>
<th>suffix ‘X’ (to be used with LA-xxxx)</th>
<th>suffix ‘E’ (to be used with LA-3812)</th>
<th>suffix ‘Z’</th>
<th>suffix ‘K’ (to be used for Keystone devices)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stm_clk</td>
<td>emu19</td>
<td>emu2</td>
<td>emu2</td>
<td>emu0</td>
<td>emu10</td>
</tr>
<tr>
<td>stm_data[0]</td>
<td>emu18</td>
<td>emu3</td>
<td>emu0</td>
<td>emu1</td>
<td>emu0</td>
</tr>
<tr>
<td>stm_data[1]</td>
<td>emu17</td>
<td>emu4</td>
<td>emu1</td>
<td>emu2</td>
<td>emu1</td>
</tr>
<tr>
<td>stm_data[2]</td>
<td>emu16</td>
<td>emu5</td>
<td>emu3</td>
<td>emu3</td>
<td>emu2</td>
</tr>
<tr>
<td>stm_data[3]</td>
<td>emu15</td>
<td>emu6</td>
<td>emu4</td>
<td>emu4</td>
<td>emu3</td>
</tr>
</tbody>
</table>

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 | CORE <core_number>

Displays the STM registers.

- 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
STM.state

Display STM settings

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.

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.

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

See also
- STM
- STM.EventMASK
- STM.IdleCount
- STM.OCPAutoIdle
- STM.PortEndianness
- STM.SetMaster
- STM.TimeStampCLOCK
- STM.ChannelRepeat
- STM.FilterChannels
- STM.IgnoreHeader
- STM.OFF
- STM.PortMASK
- STM.PortTraceFormat
- STM.SWMasters
- STM.COMPression
- STM.FilterMasters
- STM.Init
- STM.ON
- STM.PortMode
- STM.PortRoute
- STM.SyncPeriod
- STM.SyncTime
- STM.TraceID
- STM.TracePriority
- STM.DMArequests
- STM.HWMasters
- STM.MasterRepeat
- STM.PATTERN
- STM.RESet
- STM.Synch
- STM.TracePriority

©1989-2019 Lauterbach GmbH
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
STM.TimeStampCLOCK  Configure debugger for STM timestamp clock

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 Component - TI specific Target Configuration

STM.HWMasters
Enable hardware masters for tracing

Format: $\text{STM}[\text{<instance>}]$.HWMasters $\text{<name>}$ [ON | OFF]

Available $\text{<names>}$ 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: $\text{STM}[\text{<instance>}]$.IdleCount $\text{<count>}$

If there are no STP packets to be sent, $\text{<count>}$ 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: \texttt{STM[<instance>].OCPAutoIdle [ON | OFF]}

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

See also
\begin{itemize}
  \item STM
  \item STM.state
\end{itemize}

STM.PATTERN

Enable test pattern generator

Format: \texttt{STM[<instance>].PATTERN <pattern>}

If \(<\text{pattern}>\) in nonzero, the selected test pattern is output instead of STP messages.

See also
\begin{itemize}
  \item STM
  \item STM.state
\end{itemize}
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

©1989-2019 Lauterbach GmbH
STM Component - CoreSight specific Target Configuration

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: \texttt{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: \texttt{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.

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
Overview STM<trace>

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 “List of <trace> Command Groups consisting of <trace_source><trace_method>” (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.
The STMAnalyzer command group allows to display and analyze the information emitted by the system trace implementations listed in the “Introduction”, page 6.

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.

<table>
<thead>
<tr>
<th>&lt;sub_cmd&gt;</th>
<th>For descriptions of the subcommands, please refer to the general &lt;trace&gt; command descriptions in “General Commands Reference Guide T” (general_ref_t.pdf).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
<td>For a description of STMAnalyzer.List, refer to &lt;trace&gt;.List</td>
</tr>
<tr>
<td>&lt;stm_channel&gt;,&lt;channel&gt;</td>
<td>For information about the channels, see STMTrace.</td>
</tr>
</tbody>
</table>

See also

- STM<trace>
- ‘STM Component - General Target Configuration’ in ‘System Trace User’s Guide’
STMCAnalyzer

Analyze STM data recorded by TRACE32 CombiProbe

Format: \texttt{STMCAnalyzer.<sub\_cmd>[<stm\_channel>…]<channel>…]}

The \texttt{STMCAnalyzer} command group allows to display and analyze the information emitted by the system trace implementations listed in the \textit{Introduction}, page 6.

The STM information is emitted off-chip via:

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

The emitted STM information is recorded by the TRACE32 CombiProbe.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\texttt{<sub\_cmd>} & For descriptions of the subcommands, please refer to the general \texttt{<trace>} command descriptions in \textit{"General Commands Reference Guide T"} (general_ref_t.pdf). \\
\hline
\texttt{Example:} & \texttt{For a description of STMCAnalyzer.List, refer to <trace>.List}\\
\hline
\texttt{<stm\_channel>,<channel>} & For information about the channels, see \texttt{STMTrace}.\\
\hline
\end{tabular}
\end{center}

\textbf{See also}

- \texttt{STM<trace>}
- ‘STM Component - General Target Configuration’ in ‘System Trace User’s Guide’
The **STMHAnalyzer** command group allows to display and analyze the information emitted by the system trace implementations listed in the “Introduction”, page 6.

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** |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;stm_channel&gt;, &lt;channel&gt;</strong></td>
<td>For information about the channels, see <strong>STMTrace</strong>.</td>
</tr>
</tbody>
</table>

See also
- STM<trace>
- ‘STM Component - General Target Configuration’ in ‘System Trace User’s Guide’

---

**STMLA**

Display and analyze STM data from binary file

<table>
<thead>
<tr>
<th><strong>Format:</strong></th>
<th><strong>STMLA.&lt;sub_cmd&gt; [stm_channel&gt;…] [channel&gt;…]</strong></th>
</tr>
</thead>
</table>
| **<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’
STMOnchip
Display and analyze STM data stored on target memory

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 6.

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’
STMTrace
Method-independent display and analysis of STM trace data

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

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.

Example: For a description of STMTrace.List, refer to <trace>.List

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

STMTrace.List STMMASTER STMCHANNEL CYcle Data TIme Back

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:
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'
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**

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:

<table>
<thead>
<tr>
<th>Table</th>
<th>STPv1</th>
<th>STPv2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start of message</strong></td>
<td>D8, D16, D32, D64</td>
<td>D4, D8, D16, D32, D64</td>
</tr>
<tr>
<td><strong>Message body</strong></td>
<td>D8, D16, D32, D64</td>
<td>D4, D8, D16, D32, D64</td>
</tr>
<tr>
<td><strong>End of message</strong></td>
<td>D8TS, D16TS, D32TS, D64TS</td>
<td>D4TS, D8TS, D16TS, D32TS, D64TS, FLAG</td>
</tr>
</tbody>
</table>
Kernel log messages

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

<table>
<thead>
<tr>
<th></th>
<th>STPv1</th>
<th>STPv2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of message</td>
<td></td>
<td>D4TS, D8TS, D16TS, D32TS, D64TS</td>
</tr>
<tr>
<td>Message body</td>
<td></td>
<td>D4, D8, D16, D32, D64</td>
</tr>
<tr>
<td>End of message</td>
<td></td>
<td>FLAG</td>
</tr>
</tbody>
</table>

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 whose lower 16bit data must be 0x0001. The message body consists of 3 D32 packets, followed by a FLAG packet:

<table>
<thead>
<tr>
<th></th>
<th>STPv1</th>
<th>STPv2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of message</td>
<td>n.a.</td>
<td>D32TS (0x????0001)</td>
</tr>
<tr>
<td>Process ID</td>
<td>n.a.</td>
<td>D32</td>
</tr>
<tr>
<td>Target address</td>
<td>n.a.</td>
<td>D32</td>
</tr>
<tr>
<td>Source address</td>
<td>n.a.</td>
<td>D32</td>
</tr>
<tr>
<td>End of message</td>
<td>n.a.</td>
<td>FLAG</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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

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

PrintfTrace.List STMMASTER STMCHANNEL pid sYmbolN MESSAGE
```
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.

<table>
<thead>
<tr>
<th>&lt;generic&gt;</th>
<th>For descriptions of the generic subcommands, click here.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instance of the primary CMI component.</td>
</tr>
<tr>
<td>2</td>
<td>Instance of the secondary CMI component.</td>
</tr>
<tr>
<td>TraceID &lt;id&gt;</td>
<td>Sets the ATB trace ID of the corresponding CMI component.</td>
</tr>
</tbody>
</table>

Example:

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

See also

- SYStem.CONFIG
- CMI
- 'COMPonent Functions' in 'General Function Reference'
- 'Generic Subcommands, Parameters, and Options' in 'System Trace User's Guide'

<table>
<thead>
<tr>
<th>1</th>
<th>Instance of the primary CMI component.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Instance of the secondary CMI component.</td>
</tr>
<tr>
<td>&lt;sub_cmd&gt;</td>
<td>For a description of the subcommands, refer to the command descriptions in this chapter.</td>
</tr>
</tbody>
</table>

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.

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

See also
- CMI.EnableMessage
- CMI.Register
- CMI.Trace
- CMI.Mode
- CMI.RESET
- SYStem.CONFIG.CMI
- CMI.OFF
- CMI.SamplingWindow
- CMIBASE()
**CMI.EnableMessage**

Enables event or activity message generation

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.

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

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 is in a halted state. Activity monitoring must be enabled via `CMI<instance>.Mode ACTIVITY` in addition.

<table>
<thead>
<tr>
<th>TargetActivity8</th>
<th>Count target activity cycles. (If <code>CMI&lt;instance&gt;.SamplingWindow.Size &gt;= 16</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetActivity4</td>
<td>Count target activity cycles. (If <code>CMI&lt;instance&gt;.SamplingWindow.Size &lt; 16</code>)</td>
</tr>
<tr>
<td>InitiatorActivity8</td>
<td>Count initiator activity cycles. (If <code>CMI&lt;instance&gt;.SamplingWindow.Size &gt;= 16</code>)</td>
</tr>
<tr>
<td>InitiatorActivity4</td>
<td>Count initiator activity cycles. (If <code>CMI&lt;instance&gt;.SamplingWindow.Size &lt; 16</code>)</td>
</tr>
</tbody>
</table>

See also

- CMI
- CMI.state
CMI.Mode  
Set event or activity mode

Format:  \[\text{CMI}<\text{instance}>.\text{Mode}[\text{EVENT} | \text{ACTIVITY}]\]

| EVENT (default) | Selects event mode monitoring. |
| ACTivity       | Selects activity mode monitoring. |

See also  
- CMI  
- CMI.state

CMI.OFF  
Switch CMI off

Format:  \[\text{CMI}<\text{instance}>.\text{OFF}\]

Switches the CMI component off.

See also  
- CMI  
- CMI.state

CMI.ON  
Switch CMI on

Format:  \[\text{CMI}<\text{instance}>.\text{ON}\]

Switches the CMI component on.

See also  
- CMI  
- CMI.state
CMI.Register
Display the CMI register

Format:  \texttt{CMI<instance>.Register \lbrack <option>\rbrack}

\texttt{<option>}: \texttt{SpotLight | DualPort | Track | CORE \ <core_number>}

Displays the CMI registers.

\texttt{<option>}: For a description of the options, see \texttt{PER.view}.

See also
- CMI
- CMI.state

CMI.RESet
Resets CMI settings to their defaults

Format:  \texttt{CMI<instance>.RESet}

All CMI settings are reset to their defaults.

See also
- CMI
- 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
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:

![Example of the CMI1.state window](image)

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.

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

---

**See also**
- CMI
- CMI.ON
- CMI.EnableMessage
- CMI.Register
- CMI.Mode
- CMI.OFF
- CMI.RESet
- CMI.SamplingWindow
This example for an OMAP4430 which has got two CMI components. In this case the components are addressed as CMI1 or CMI2, respectively.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAnalyzer.AutoArm OFF</td>
<td>In activity mode CMI messages are output permanently, so we must arm the CAnalyzer before the CMI is activated.</td>
</tr>
<tr>
<td>CAnalyzer.ARM</td>
<td></td>
</tr>
<tr>
<td>CMI1.RESet</td>
<td></td>
</tr>
<tr>
<td>CMI1.SamplingWindow.Size 15</td>
<td></td>
</tr>
<tr>
<td>CMI1.Mode ACTivity</td>
<td></td>
</tr>
<tr>
<td>CMI1.ON</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>CMI1.OFF</td>
<td>Turn off CMI ...</td>
</tr>
<tr>
<td>CAnalyzer.OFF</td>
<td>... before shutting down the CAnalyzer.</td>
</tr>
<tr>
<td>CMITrace.List CYcle CMITA.&lt;name1&gt;</td>
<td>Display cycle activity of target &lt;name1&gt; and &lt;name2&gt;.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th><code>&lt;sub_cmd&gt;</code></th>
<th>Format: <strong>CMITrace.&lt;sub_cmd&gt;</strong> [ <code>&lt;cmi_channel&gt;</code> ] [ <code>&lt;channel&gt;</code> ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cmi_channel&gt;</code></td>
<td><strong>CMICD.&lt;domain&gt;</strong></td>
</tr>
</tbody>
</table>

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`.

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

For a description of the default channels, see `<trace>.List ... <items>`. 
Example:

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

See also

- CMI
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.

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

tbd.

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

- CPTracer.<aggregator>.<probe>.ADDRessHIGH
- CPTracer.<aggregator>.<probe>.CHannel
- CPTracer.<aggregator>.<probe>.PERiod
- CPTracer.<aggregator>.OFF
- CPTracer.<aggregator>.SYNC
- CPTracer.TraceID
**CPTracer.TraceID**

**Set ATB ID**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>CPTracer.TraceID &lt;id&gt;</code></th>
</tr>
</thead>
</table>

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**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>CPTracer.&lt;aggregator&gt;.ON</code></th>
</tr>
</thead>
</table>

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

**See also**

- CPTracer
- CPTracer.state

**CPTracer.<aggregator>.OFF**

**Switch aggregator off**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>CPTracer.&lt;aggregator&gt;.OFF</code></th>
</tr>
</thead>
</table>

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

**See also**

- CPTracer
- CPTracer.state
**CPTracer.<aggregator>.SYNC**

**Sync period of aggregator**

<table>
<thead>
<tr>
<th>Format:</th>
<th>CPTracer.SYNC &lt;bytes&gt;</th>
</tr>
</thead>
</table>

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 CPTracerTrace.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.

- **B0 … B255** SP **B0 … B255** SP **B0 … B255** SP ...
- 0x100 0x100 0x100

**B** = regular trace <bytes>

**SP** = synchronization packet

---

**See also**

- CPTracer
- CPTracer.state

---

**CPTracer.<aggregator>.<probe>.ADDRessLOW**

**Lower filter address**

<table>
<thead>
<tr>
<th>Format:</th>
<th>CPTracer.&lt;aggregator&gt;.&lt;probe&gt;.ADDRessLOW &lt;value&gt;</th>
</tr>
</thead>
</table>

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:  
\[
\text{CPTracer.<aggregator>.<probe>.ADDRessLOW <value>}
\]

Default: 0xFFFFFFFFFFFFF

Only transactions with address \( \leq <value> \) will generate trace packets. This command must be used together with \text{CPTracer.<aggregator>.ADDRessLOW}.

---

**See also**

- CPTracer
- CPTracer.state

---

**CPTracer.<aggregator>.<probe>.CHANNEL**  
Filter by channel ID

Format:  
\[
\text{CPTracer.<aggregator>.<probe>.CHANNEL <bitmask>}
\]

Default: 0xXXXXXXXXXXXX

Only transactions with a channel ID within \(<bitmask>\) will generate trace packets.

---

**See also**

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

Format:  

\[
\text{CPTracer.}<\text{aggregator}>.<\text{probe}>.\text{OPeration} \ <\text{mode}>
\]

\(<\text{mode}>:\)

- 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:  

\[
\text{CPTracer.}<\text{aggregator}>.<\text{probe}>.\text{PERiod} \ <\text{value}>
\]

Default: 0x3FFF

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

See also
- CPTracer
- CPTracer.state
Format: \texttt{CPTracer.<aggregator>.<probe>.RouteID <bitmask>}

Default: 0yXXXXXXXXXXXX

Only transactions with a route ID within \texttt{<bitmask>} will generate trace packets.

\textbf{See also}

- CPTracer
- CPTracer.state
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.CAL0.0Peration.LATency
CPTracer.SOC.MCU.EXPORT_SLV.0Peration.LATency

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

; Display the recorded trace data
CPTracerTrace.List PRobe CYcle Address
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 **CPTTracer**, plus the keyword **Trace** of the `<trace>` command group.

<table>
<thead>
<tr>
<th><code>&lt;cpt_channel&gt;</code></th>
<th>The following channels are relevant for the analysis of CPT trace data:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xxx&gt;</code></td>
<td>A list of all valid replacements for the placeholder <code>&lt;xxx&gt;</code> will be displayed as softkeys in TRACE32 as soon as the dot <code>.</code> is entered in the TRACE32 command line.</td>
</tr>
<tr>
<td><strong>PRobe</strong></td>
<td>Name of the probe. See example.</td>
</tr>
<tr>
<td><strong>CYcle</strong></td>
<td>Type of trace packet</td>
</tr>
<tr>
<td><strong>Address</strong></td>
<td>Traced address of transaction packet</td>
</tr>
<tr>
<td><strong>LAT.&lt;xxx&gt;</strong></td>
<td>Latency packet specific information <code>&lt;xxx&gt;</code>. See example.</td>
</tr>
<tr>
<td><strong>TRANS.&lt;xxx&gt;</strong></td>
<td>Transaction packet specific information <code>&lt;xxx&gt;</code>.</td>
</tr>
<tr>
<td><strong>THRU.&lt;xxx&gt;</strong></td>
<td>Throughput packet specific information <code>&lt;xxx&gt;</code>.</td>
</tr>
<tr>
<td><strong>TIme.&lt;xxx&gt;</strong></td>
<td>Timing information.</td>
</tr>
</tbody>
</table>

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

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 **CPTTracerTrace.List**, refer to `<trace>.List`. 

---

**Format:**

CPTTracerTrace.<sub_cmd> [<cpt_channel>...] [<channel>...]

- `<cpt_channel>`:
  - PRobe
  - CYcle
  - Address
  - LAT.<xxx>
  - TRANS.<xxx>
  - THRU.<xxx>
  - TIme.<xxx>

<table>
<thead>
<tr>
<th><code>&lt;sub_cmd&gt;</code></th>
<th>For descriptions of the subcommands, please refer to the general <code>&lt;trace&gt;</code> command descriptions in “<strong>General Commands Reference Guide T</strong>” (general_ref_t.pdf).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example:</strong> For a description of <strong>CPTTracerTrace.List</strong>, refer to <code>&lt;trace&gt;.List</code>.</td>
<td></td>
</tr>
</tbody>
</table>
Example:

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

tbd.

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:**

<table>
<thead>
<tr>
<th>SYStem.CONFIG.TIOCPBASE (deprecated)</th>
<th>SYStem.CONFIG.OCP.Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYStem.CONFIG.TIOCPID (deprecated)</td>
<td>SYStem.CONFIG.OCP.TraceID</td>
</tr>
</tbody>
</table>

**See also**

- SYStem.CONFIG
- OCP
- ‘COMPonent Functions’ in ‘General Function Reference’
- ‘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.

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

See also
- `OCP.AutoIDLE`
- `OCP.DebugPort`
- `OCP.TRaceFilter`
- `OCP.TraceOFF`
- `OCPTrace`
- `SYstem.CONFIG.OCP`
- `OCP.OFF`
- `OCP.state`
- `OCP.TraceON`
- `OCP.ON`
- `OCPTraceEnable`
- `OCPTriggerOut<x>`
OCP.AutoIDLE

**OCP-WP clocking strategy**

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 | 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
Shows the OCP setup window.

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<cx>**
OCP.TraceFilter\(<x>\).NAME

Name a filter

Format: \texttt{OCP.TraceFilter\(<x>\).NAME }\texttt{<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 \texttt{TraceOCP.List FilterName}.

Example:

\begin{verbatim}
OCP.TraceFilter0.NAME "Filter0"
\end{verbatim}

OCP.TraceFilter\(<x>\).MCmd

Filters traffic by transaction type

Format: \texttt{OCP.TraceFilter\(<x>\).MCmd }\texttt{<command>}

Default: ALL.

Only transactions of type \texttt{<command>} will pass filter \texttt{<x>}.

See also

\begin{itemize}
  \item \texttt{OCP.TraceFilter}\(<x>\).Initiator
  \item \texttt{OCP.TraceFilter}\(<x>\).MCmd
  \item \texttt{OCP.TraceFilter}\(<x>\).NAME
  \item \texttt{OCP.TraceFilter}\(<x>\).REQinfo
  \item \texttt{OCP.state}
\end{itemize}
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

©1989-2019 Lauterbach GmbH
**OCP.TraceEnable**

Filter OCP traffic by address range

<table>
<thead>
<tr>
<th>Format:</th>
<th>OCP.TraceEnable  &lt;range&gt;</th>
</tr>
</thead>
</table>

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 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
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.

### Format:

```
OCPTrace.<sub_cmd> [<ocp_channel>] [<channel>...]
```

<table>
<thead>
<tr>
<th>&lt;sub_cmd&gt;</th>
<th>For descriptions of the subcommands, please refer to the general &lt;trace&gt; command descriptions in “General Commands Reference Guide T” (general_ref_t.pdf).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong>:</td>
<td>For a description of OCPTrace.List, refer to &lt;trace&gt;.List</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;ocp_channel&gt;</th>
<th>The following channel is relevant for the analysis of OCP trace data:</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCPFN</td>
<td>FilterName: Name of the filter the OCP message has passed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;channel&gt;</th>
<th>For a description of the default channels, see &lt;trace&gt;.List ... &lt;items&gt;.</th>
</tr>
</thead>
</table>

**Example:**

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

**See also**

- OCP
### SYStem.CONFIG.PMI

Inform TRACE32 about PMI component

**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.

<table>
<thead>
<tr>
<th><code>&lt;generic&gt;</code></th>
<th>For descriptions of the generic subcommands, click here.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TraceID</td>
<td>Sets the STM master ID of the PMI component.</td>
</tr>
</tbody>
</table>

#### Deprecated vs. New Commands:

<table>
<thead>
<tr>
<th>SYStem.CONFIG.TIPMIBASE (deprecated)</th>
<th>SYStem.CONFIG.PMI.Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYStem.CONFIG.TIPMIID (deprecated)</td>
<td>SYStem.CONFIG.PMI.TraceID</td>
</tr>
</tbody>
</table>

**See also**

- SYStem.CONFIG
- PMI
- ‘COMPonent Functions’ in ‘General Function Reference’
- ‘Generic Subcommands, Parameters, and Options’ in ‘System Trace User’s Guide’
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.

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]

\(<event_msg>:\quad\text{LogicVoltage}\quad\text{MemoryVoltage}\quad\text{LogicPower}\quad\text{MemoryPower}\)

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 | 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
PMI.SamplingWindow

**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.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.
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.

Format: PMITrace.<sub_cmd> [<pmi_channel>…] [<channel>…]

<pmi_channel>: PMILV.<domain> | PMILVOFF | PMIMV.<domain> | PMILP.<domain> | PMIMP.<domain> | PMISTAT | PMILAT | CYcle

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

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.

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
### SYStem.CONFIG.SC

**Inform TRACE32 about StatCol component**

<table>
<thead>
<tr>
<th>Format:</th>
<th>SYStem.CONFIG.SC.&lt;sub_cmd&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sub_cmd&gt;:</td>
<td>&lt;generic&gt;</td>
</tr>
<tr>
<td>&lt;component_specific&gt;:</td>
<td>TraceID &lt;id&gt;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>&lt;generic&gt;</th>
<th>For descriptions of the generic subcommands, click here.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TraceID</td>
<td>Set the STM master ID of the statistics collector.</td>
</tr>
</tbody>
</table>

#### Deprecated vs. New Commands:

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

**See also**

- SYStem.CONFIG
- StatCol
- ‘COMPonent Functions’ in ‘General Function Reference’
- ‘Generic Subcommands, Parameters, and Options’ in ‘System Trace User’s Guide’

©1989-2019 Lauterbach GmbH
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.

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

**See also**

- `StatCol.<probe>.CollectTime`
- `StatCol.<probe>.OFF`
- `StatCol.<probe>.REQuestE venT`
- `StatCol.RESet`
- `StatColTrace`
- `StatCol.<probe>.Counter`
- `StatCol.<probe>.ON`
- `StatCol.<probe>.ReSPonseE venT`
- `StatCol.state`
- `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.

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

©1989-2019 Lauterbach GmbH
**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
StatCol.<probe>.REQuestEVenT

Select event detector

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

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
StatCol.<probe>.Counter <counter> ADDRMEN

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

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
Set max threshold for events

Format: \texttt{StatCol.<probe>.Counter <counter> MAX.<value>}

Default: 0.

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

See also
- \texttt{StatCol.<probe>.Counter}

Set min threshold for events

Format: \texttt{StatCol.<probe>.Counter <counter> MIN.<value>}

Default: 0.

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

See also
- \texttt{StatCol.<probe>.Counter}
StatCol.<probe>.Counter <counter> SELect

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

Set filter criteria

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;item&gt;:</td>
<td>Master address (NTTP) or MConnId (OCP).</td>
<td>RequestUserInfo bits (NTTP only)</td>
<td>ResponseUserInfo bits (NTTP only)</td>
<td>Slave address (NTTP only)</td>
</tr>
<tr>
<td>MasterADDRess</td>
<td>Read bit</td>
<td>Error bit (NTTP only)</td>
<td>RequestUserInfo bits (NTTP only)</td>
<td>Slave address (NTTP only)</td>
</tr>
<tr>
<td>Read</td>
<td>Write bit</td>
<td></td>
<td>ResponseUserInfo bits (NTTP only)</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>ERRor Error bit (NTTP only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRor</td>
<td>RequestUserInfo bits (NTTP only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RequestUserInfo</td>
<td>ResponseUserInfo bits (NTTP only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResponseUserInfo</td>
<td>Slave address (NTTP only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SlaveADDRess</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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
StatCol Macro Functions

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.

AvgPayloadLength

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

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional additional configuration</td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; MaSTerADDResS</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDResS</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; ReaD</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; WRite</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; REQUserInfo</td>
</tr>
</tbody>
</table>

THRoughput

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

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MUX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.CollectTime</td>
</tr>
</tbody>
</table>

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>Optional additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; MaSTerADDRess</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDRess</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; ReaD</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; WRite</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; REQUUserInfo</td>
</tr>
</tbody>
</table>

**LnkOcc**

Link occupancy: Percentage of non-idle cycles.

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.CollectTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional additional configuration</td>
<td>-</td>
</tr>
</tbody>
</table>

**ArbConf**

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

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.CollectTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional additional configuration</td>
<td>-</td>
</tr>
</tbody>
</table>

**TransUflow**

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

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.CollectTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional additional configuration</td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; MaSTerADDRess</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDRess</td>
</tr>
</tbody>
</table>
**IBusy**

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

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.CollectTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional additional configuration</td>
<td>-</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MIN (Minimum payload length in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MAX (Maximum payload length in bytes)</td>
</tr>
<tr>
<td>Optional additional configuration</td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; MASTerADDResS</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDResS</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; ReaD</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; WRite</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; REQUserInfo</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MIN (Minimum pressure)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MAX (Maximum pressure)</td>
</tr>
<tr>
<td>Optional additional configuration</td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDRess</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; ReaD</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; WRite</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; REQUserInfo</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MIN (Minimum latency)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; MAX (Maximum latency)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDRess</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; MaSTerADDRess</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; REQUserInfo</td>
</tr>
</tbody>
</table>

**AvgLatDist**

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

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>-</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Optional additional configuration</th>
<th>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; SLaVeADDRess</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; MaSTerADDRess</td>
</tr>
<tr>
<td></td>
<td>StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;element&gt; REQUserInfo</td>
</tr>
</tbody>
</table>
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
```
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.

### Example:
```plaintext
StatColTrace.List CYCle SCC0 SCC1 SCC2 SCC3
```

### Format:
```
StatColTrace.<sub_cmd> [<statcol_channel>...] [<channel>...]
```

### <statcol_channel>:
- **SCC0.<probe>** | **SCC1.<probe>** | **SCC2.<probe>** | **SCC3.<probe>** | **SCC4.<probe>**

### <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**

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

**Example:**
```plaintext
; Display trace data
StatColTrace.List CYCle SCC0 SCC1 SCC2 SCC3
```
This section describes the `<generic>` subcommands, parameters, and options that are common to the `SYStem.CONFIG.<component>` commands.

### SYStem.CONFIG.<component>.<generic>

<table>
<thead>
<tr>
<th>Format:</th>
<th>SYStem.CONFIG.&lt;component&gt;.&lt;generic&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;comp.t&gt;</code>:</td>
<td>CMI</td>
</tr>
<tr>
<td><code>&lt;generic&gt;</code>:</td>
<td>Base</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>RESet</td>
</tr>
<tr>
<td></td>
<td>view</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>&lt;component&gt;</code></th>
<th>Click a blue <code>&lt;component&gt;</code> name to jump to the respective SYStem.CONFIG.&lt;component&gt; command: CMI, OCP, PMI, SC, STM.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;generic&gt;</code></td>
<td>Generic subcommands of the SYStem.CONFIG.&lt;component&gt; commands.</td>
</tr>
</tbody>
</table>

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>

<br />

**<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> [I<option>]

<br />

**<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.

<table>
<thead>
<tr>
<th>CORE</th>
<th>tbd.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTinue</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

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

Reset of a component

**Format:**

SYStem.CONFIG.<component>.RESet

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

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.

| CONTROLS | tbd. |
FAQ

No information available