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

STM.PrintfTraceFormat

Define format of printftrace style messages

STM.Register

Display STM register

©1989-2021 Lauterbach GmbH
STM.RESet                      Reset STM settings 26
STM.SetMaster                  Set master ID manually 26
STM.state                      Display STM settings 27
STM.SyncTime                   Trace decoder resync time 28
STM.SyncPeriod                 Add synchronization packets 28
STM.TimeStamps                 Enables timestamps 28
STM.TimeStampCLOCK            Configure debugger for STM timestamp clock 29

**STM Component - TI specific Target Configuration**  .......................................................... 30
STM.HWMasters                  Enable hardware masters for tracing 30
STM.IdleCount                  Maximum idle packets 30
STM.IgnoreHeader               Ignore leading dword in printftrace message 31
STM.SWMasters                  Enable software masters for tracing 31
STM.OCPAutoIdle                Set OCP idle clock behavior 32
STM.PATTERN                    Enable test pattern generator 32
STM.ChannelRepeat              Period of channel packet insertion 33
STM.MasterRepeat               Period of master packet insertion 33

**STM Component - CoreSight specific Target Configuration**  .................................................. 34
STM.DMArequests                DMA requests enable 34
STM.COMPression                Data compression enable 34
STM.EventMASK                  Mask hardware event inputs 35
STM.PortMASK                   Mask stimulus ports 35
STM.TraceID                    Sets trace ID 36
STM.TracePriority              Set priority for STM manually 36

**STM<trace> - Trace Data Analysis**  .......................................................................................... 37
STM<trace>                      Command groups for STM<trace> recording and analysis 37
Overview STM<trace>            37
STMAnalyzer                    Analyze STM data recorded by TRACE32 PowerTrace 38
STMAnalyzer                    Analyze STM data recorded by TRACE32 CombiProbe 39
STMAnalyzer                    Display and analyze STM data recorded by the host 40
STMAnalyzer                    Display and analyze STM data from binary file 40
STMAnalyzer                    Display and analyze STM data stored on target memory 41
STMAnalyzer                    Method-independent display and analysis of STM trace data 42

**PrintfTrace**  .......................................................................................................................... 44
PrintfTrace                     Decoder for STP-based software messages 44

**CMI Component**  ..................................................................................................................... 47
SYStem.CONFIG.CMI              Inform TRACE32 about CMI component 47
CMI                            Configure CMI component on target 48
CMI.EnableMessage              Enables event or activity message generation 49
CMI.Mode                       Set event or activity mode 50
CMI.OFF                        Switch CMI off 50
CMI.ON                         Switch CMI on 50
CMI.Register                   Display the CMI register 51
CMI.RESet
CMI.SamplingWindow
CMI.SamplingWindow.CLock
CMI.SamplingWindow.Size
CMI.state
CMI Example
CMITrace

CMN Component

SYStem.CONFIG.CMN
CMN
CMN.EnhancedFilter
CMN.Init
CMN.OFF
CMN.ON
CMN.Opcde
CMN.PortRoute
CMN.Register
CMN.RESet
CMN.NodeID
CMN.state
CMN.SyncPeriod
CMN.TimeStampPeriod
CMN.TraceChannel
CMN.TraceID
CMN.TracePriority
CMN Example
CMNTrace

CPTracer Component

CPTracer
CPTracer.RESet
CPTracer.state
CPTracer.TraceID
CPTracer.<aggregator>.ON
CPTracer.<aggregator>.OFF
CPTracer.<aggregator>.SYNC
CPTracer.<aggregator>.ADDRessLOW
CPTracer.<aggregator>.ADDRessHIGH
CPTracer.<aggregator>.CHannel
CPTracer.<aggregator>.OPeration
CPTracer.<aggregator>.PERiod
CPTracer Example
CPTracerTrace
OCP Component ........................................................................................................... 77
SYStem.CONFIG.OCP Inform TRACE32 about OCP component 77
OCP Configure OCP component on target 78
OCP.AutoIDLE OCP-WP clocking strategy 79
OCP.DebugPort Select target to be traced 79
OCP.OFF Switch OCP off 79
OCP.ON Switch OCP on 80
OCP.Register Display OCP registers 80
OCP.RESet Reset OCP settings to their defaults 80
OCP.state Display OCP settings 81
OCP.TraceFilter Set filter criteria 82
OCP.TraceFilter<x>.NAME Name a filter 82
OCP.TraceFilter<x>.MCmd Filters traffic by transaction type 82
OCP.TraceFilter<x>.Initiator Filters traffic by transaction initiator 83
OCP.TraceFilter<x>.REQinfo Filters traffic by transaction qualifier 83
OCP.TraceEnable Filter OCP traffic by address range 84
OCP.TraceOFF Stop tracing 84
OCP.TraceON Start tracing 85
OCP.TriggerOut<x> Generate trigger event 85
OCPTrace Display and analyze OCP trace data 86

PMI Component ....................................................................................................... 87
SYStem.CONFIG.PMI Inform TRACE32 about PMI component 87
PMI Configure PMI component on target 88
PMI.EnableMessage Enables event message generation 89
PMI.OFF Switch PMI off 89
PMI.ON Switch PMI on 90
PMI.Register Display the PMI registers 90
PMI.RESet Resets PMI settings to their defaults 90
PMI.SamplingWindow Sampling window 91
PMI.SamplingWindow.CLocK Set sampling window clock 91
PMI.SamplingWindow.Size Set sampling window size 91
PMI.state Display PMI settings 92
PMI Example 93
PMITrace Display and analyze PMI trace data 94

StatCol Component (Statistics Collector) ........................................................................... 96
SYStem.CONFIG.SC Inform TRACE32 about StatCol component 96
StatCol Configure StatCol component on target 97
StatCol.RESet Resets all statistics collector settings to their default 98
StatCol.state Display statistics collector settings 98
StatCol.<probe>.OFF Switch probe off 99
StatCol.<probe>.ON Switch probe on 99
StatCol.<probe>.REQuestEVenT Select event detector 100
StatCol.<probe>.ReSPonseEVenT Select event detector 101
StatCol.<probe>.CollectTime  Set up collection period  101
StatCol.<probe>.Counter     Counter configuration  102
StatCol.<probe>.Counter <counter> ADDRMAX Filter max address  102
StatCol.<probe>.Counter <counter> ADDRMIN Filter min address  102
StatCol.<probe>.Counter <counter> ADDREN Enable address filtering  103
StatCol.<probe>.Counter <counter> EventInfo Select ‘EventInfo’ to count  103
StatCol.<probe>.Counter <counter> MAX Set max threshold for events  104
StatCol.<probe>.Counter <counter> MIN Set min threshold for events  104
StatCol.<probe>.Counter <counter> SELect Set counter input  105
StatCol.<probe>.Counter <counter> Filter Set filter criteria  106
StatCol.<probe>.Counter <counter> Filter <filter> MUX Input port  107
StatCol.<probe>.Counter <counter> Filter <filter> OFF Switch filter off  107
StatCol.<probe>.Counter <counter> Filter <filter> ON Switch filter on  107
StatCol.<probe>.Counter <counter> FunCTioN Predefined settings  108
            OFF  108
            AvgPayloadLength  109
            THRoughput  109
            LnkOcc  109
            ArbConf  110
            TransUflow  110
            IBusy  110
            HistPayloadLen  111
            HistPresDist  111
            HistLatDist  111
            AvgLatDist  112
StatCol Example  113
StatColTrace   Display and analyze StatCol trace data  114

Generic Subcommands, Parameters, and Options ......................................................... 115
SYStem.CONFIG.<component>..<generic> Base address of a component  115
SYStem.CONFIG.<component>.Base Base address of a component  116
SYStem.CONFIG.<component>.Name Name of a component  116
SYStem.CONFIG.<component>.RESet Reset of a component  116
SYStem.CONFIG.<component>.view Display component settings  117

FAQ  ................................................................................................................. 117
History

27-Nov-20  "uTrace" renamed to "µTrace (MicroTrace)".

24-Aug-20  Changed command name CMN.SourceID to CMN.NodeID.

08-Jul-20   New command groups CMN and CMNT to support flexible interconnect Coherent Mesh Network used in Cotex-A/R based chips.

11-Jul-19   Updated, revised, and restructured the manual.

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>
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
Example STP Data to Parallel Trace Interface

Target: OMAP4430

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

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

; Route STP data to PTI
STM.PortRoute CAnalyzer

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

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

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

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

; Turn on STM component
STM.ON
Example Attach to Onchip Trace

NOTE: This example applies to CoreSight compliant STMs only.

; Setup target
SYStem.CPU <cpu>
SYStem.CONFIG <config>
... ; Do not attach to the target yet!
SystemTrace.Method Onchip
Onchip.TraceConnect <buffer>
SYStem.Attach
Onchip.Disable
Onchip.Attach
SystemTrace.List

Example Save/Load STP Data Embedded in CoreSight Trace Stream

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

Save file:

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

Load file:

TPIU.PortMode WRAPPED
CoreSightTrace.METHOD LA
SystemTrace.METHOD LA
LA.IMPORT.TracePort <file>
LA.IMPORT.StartValid
SystemTrace.List
STM Component - Configuration of TRACE32

SYStem.CONFIG.STM

Inform TRACE32 about STM component

Format:  SYStem.CONFIG.STM[<instance>] <sub_cmd>

<instance>:  1 | 2

<sub_cmd>:  <generic> | <component_specific>

<component_specific>:  ACCESS [Denied | ReadWrite]
                      Type <type>
                      Mode <mode>
                      AutoSync [ON | OFF]

Default: Device specific.

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

| <instance> | For a description of <instance>, refer to the introduction to the command group STM. |
| <generic>  | For descriptions of the generic subcommands, click here.                      |
| STM        | Single STM. If the chip contains more than one STM, the individual STM can be addressed by adding a number to the keyword STM, i.e. STM1 or STM2. |
| STM1       | Same as STM command. Used to differentiate between STM1 and STM2.          |
| STM2       | Used to configure a 2nd STM, if present.                                   |
| ACCESS [Denied | ReadWrite] | Only if SYStem.CONFIG.STM.Type is set to ARM. Set this property to Denied if TRACE32 is not supposed to write to the configuration registers of an STM. Default: ReadWrite. |
| AutoSync [ON | OFF]   | If ON, TRACE32 tries to synchronize to the trace stream even if no synchronization packets can be found. This setting only has an effect for STP version >= 2. Default: ON. |
| **Mode** <span class="lauterbach-cmd" style="font-size: 0.8em;">**<span class="lauterbach-cmd-lable">mode</span>**</span> | For details, see **SYStem.CONFIG.STM.Mode**. |
| **Type** <span class="lauterbach-cmd" style="font-size: 0.8em;">**<span class="lauterbach-cmd-lable">type</span>**</span> | For details, see **SYStem.CONFIG.STM.Type**. |

### Example 1:

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

### Example 2:

```
SYStem.CONFIG STM1 Mode STPv2 ; chip contains a STM that uses
     STM1.state ; open STM1 configuration window
       STM.state ; open STM configuration window
     SYStem.CONFIG STM2 Mode STPv2 ; chip contains a second STM that
       STM2.state ; open STM2 configuration window
       uses MIPI STPv2 protocol
```

See also

- **SYStem.CONFIG.STM.Mode**
- **SYStem.CONFIG.STM.Type**
- **SYStem.CONFIG**

▲ ‘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
Configure the STM type in TRACE32.

<table>
<thead>
<tr>
<th>&lt;type&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No STM type is configured in TRACE32.</td>
</tr>
<tr>
<td>Generic</td>
<td>STM component is generic.</td>
</tr>
<tr>
<td>ARM</td>
<td>System Trace Macrocell (STM) by ARM as a CoreSight component</td>
</tr>
<tr>
<td>SDTI</td>
<td>System Debug Trace Interface (SDTI) by Texas Instruments</td>
</tr>
<tr>
<td>TI</td>
<td>System Trace Module (STM) by Texas Instruments</td>
</tr>
</tbody>
</table>

**See also**
- SYStem.CONFIG.STM
STM Component - General Target Configuration

STM

Configure STM component on target

Format: STM[<instance>].<sub_cmd>

<instance>: 1 | 2

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

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

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

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

See also

- STM.ChannelRepeat
- STM.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
STM.FilterMasters

Display specified masters only

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

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

The trace hardware is initialized and set to its defaults.

See also
- STM
- STM.state

STM.OFF

Switch STM off

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

Disables the STM functionality.

See also
- STM
- STM.state

STM.ON

Switch STM on

Format: \texttt{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

**Format:**

```
STM[<instance>].PortMode <mode>
```

**<mode>:**

- Continuous
- Gated
- Autoldle
- Bypass
- Continuous
- HalfRate
- FullRate
- 1/<divisor>

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

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

See also

- STM
- STM.state

©1989-2021 Lauterbach GmbH
### STM.PortRoute

Select output of STP data

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

<table>
<thead>
<tr>
<th></th>
<th>SDTI (TI)</th>
<th>STM (TI)</th>
<th>STM (CS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTO</strong></td>
<td>Data are directed to the Parallel Trace Interface (PTI) and recorded by the attached trace hardware (default).</td>
<td>Data are directed to the Parallel Trace Interface (PTI) and recorded by the attached trace hardware (default).</td>
<td>n.a.</td>
</tr>
<tr>
<td><strong>Analyzer</strong></td>
<td>Data are directed to the Parallel Trace Interface (PTI) and recorded by the PowerTrace II.</td>
<td>Data are directed to the Parallel Trace Interface (PTI) and recorded by the PowerTrace II.</td>
<td>n.a.</td>
</tr>
<tr>
<td><strong>CAnalyzer</strong></td>
<td>Data are directed to the Parallel Trace Interface (PTI) and recorded by the CombiProbe.</td>
<td>Data are directed to the Parallel Trace Interface (PTI) and recorded by the CombiProbe.</td>
<td>n.a.</td>
</tr>
<tr>
<td><strong>Onchip</strong></td>
<td>n.a.</td>
<td>Data are directed to the Embedded Trace Buffer.</td>
<td>n.a.</td>
</tr>
</tbody>
</table>

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
- AlternatingBackGround
- CORE <core_number>

Displays the STM registers.

<option>

For a description of the options, see PER.view.

See also

- STM
- STM.state
STM.RESet

Reset STM settings

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

All STM settings are reset to their defaults.

See also

- STM
- STM.state

STM.SetMaster

Set master ID manually

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

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

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

See also

- STM
- STM.state
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.

### See also
- STM
- STM.EventMASK
- STM.IdleCount
- STM.OCPIdle
- STM.PortEndianness
- STM.SetMaster
- STM.TimeStampCLOCK
- STM.ChannelRepeat
- STM.FilterChannels
- STM.IgnoreHeader
- STM.OFF
- STM.PortMASK
- STM.PrintTraceFormat
- STM.SWMasters
- STM.TimeStamps
- STM.COMPression
- STM.FilterMasters
- STM.Init
- STM.ON
- STM.PortMode
- STM.Register
- STM.SyncPeriod
- STM.TraceID
- STM.DMArequests
- STM.HWMasters
- STM.MasterRepeat
- STM.PATTERN
- STM.PortRoute
- STM.RESet
- STM.SyncTime
- STM.TracePriority

---

**STM.state**

Display STM settings

<table>
<thead>
<tr>
<th>Format:</th>
<th>STM[&lt;instance&gt;].state</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;instance&gt;:</td>
<td>1</td>
</tr>
</tbody>
</table>

A For descriptions of the commands in the **STM.state** window, please refer to the **STM.*** commands in this chapter.

**Example:** For information about **ON**, see **STM.ON**.

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

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

Trace decoder resync time

Format:  
\[
\text{STM}[<\text{instance}>].\text{SyncTime} <\text{time}>
\]

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

**See also**

- STM
- STM.state

---

**STM.SyncPeriod**

Add synchronization packets

Format:  
\[
\text{STM}[<\text{instance}>].\text{SyncPeriod} [<\text{value}>]
\]

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:  
\[
\text{STM}[<\text{instance}>].\text{TimeStamps} [\text{ON} \mid \text{OFF}]
\]

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

Available \textit{<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: \texttt{STM[<instance>].IdleCount <count>}

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

See also
- STM
- STM.state
STM.IgnoreHeader  Ignore leading dword in printftrace message

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

Default: OFF.

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

See also
- STM
- STM.state

STM.SWMasters  Enable software masters for tracing

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

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

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

See also
- STM
- STM.state
STM.OCPAutoIdle

Set OCP idle clock behavior

| Format: | STM[<instance>].OCPAutoIdle [ON | OFF] |
|---------|--------------------------------------|
| OFF (default) | OCP clock is free running. |
| ON | OCP clock may be gated if interface is in idle mode. |

See also

- STM
- STM.state

STM.PATTERN

Enable test pattern generator

<table>
<thead>
<tr>
<th>Format:</th>
<th>STM[&lt;instance&gt;].PATTERN &lt;pattern&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>If &lt;pattern&gt; in nonzero, the selected test pattern is output instead of STP messages.</td>
<td></td>
</tr>
</tbody>
</table>

See also

- STM
- STM.state
STM.ChannelRepeat

**Period of channel packet insertion**

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

**OFF (default)**

No extra channel packets are inserted into the STP data stream.

**<value>**

If <value> subsequent STP messages are written to the same software channel, an extra STP channel packet is inserted into the data stream. Due to the working load of the STM component it may happen that extra channel packets are inserted only every 2 * <value> packets from the same channel.

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

**See also**

- STM
- STM.state

STM.MasterRepeat

**Period of master packet insertion**

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

**OFF (default)**

No extra master packets are inserted into the STP data stream.

**<value>**

If <value> subsequent STP packets are generated by the same master, an extra STP master packet is inserted into the data stream. Due to the working load of the STM component it may happen that extra master packets are inserted only every 2 * <value> packets from the same master.

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

**See also**

- STM
- STM.state
STM 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: | STM[<instance>].EventMASK <mask> |

Default: 0xFFFFFFFF

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

**See also**
- STM
- STM.state

**STM.PortMASK**

Mask stimulus ports

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

Default: 0xFFFFFFFF

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

**See also**
- STM
- STM.state
STM.TraceID

Sets trace ID

Format: 

```
STM[<instance>].TraceID <id> | <id_start>---<id_end>
```

Default: 0x11

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

<table>
<thead>
<tr>
<th>&lt;id_start&gt;</th>
<th>Must be an even number.</th>
</tr>
</thead>
</table>

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 “Related Trace Command Groups” (general_ref_t.pdf).

Example:

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

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

STMTrace.List ;display a trace listing of the STM trace data
;recorded with the trace method Analyzer.
;<your_configuration>
STMAnalyzer.List ;this is the equivalent and explicit command.
```
The STMAalyzer 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>&lt;stm_channel&gt;,</td>
<td>For information about the channels, see STMTrace.</td>
</tr>
<tr>
<td>&lt;channel&gt;</td>
<td></td>
</tr>
</tbody>
</table>

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

See also

- STM<trace>
- 'STM Component - General Target Configuration’ in 'System Trace User's Guide’
The **STMCAnalyzer** 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.
- Or via Serial Wire Output (SWO), which is also configured with the **TPIU** command group.

The emitted STM information is recorded by the TRACE32 CombiProbe.

### Format:

```
STMCAnalyzer.<sub_cmd> [<stm_channel>...] [<channel>...]
```

<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>STMCAnalyzer.List</strong>, refer to <code>&lt;trace&gt;.List</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;stm_channel&gt;</code>, <code>&lt;channel&gt;</code></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**’
STMAnalyzer

Display and analyze STM data recorded by the host

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

The **STMAnalyzer** command group allows to display and analyze the information emitted by the system trace implementations listed in the "Introduction", page 6.

Trace data is transferred off-chip via the USB port and recorded in the trace memory of the TRACE32 host analyzer.

<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> For a description of <strong>STMAnalyzer.List</strong>, refer to &lt;trace&gt;.List</td>
<td></td>
</tr>
<tr>
<td>&lt;stm_channel&gt;, &lt;channel&gt;</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

Format: **STMLA.**<sub_cmd> [</sub_cmd>[<stm_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> For a description of <strong>STMLA.List</strong>, refer to &lt;trace&gt;.List</td>
<td></td>
</tr>
<tr>
<td>&lt;stm_channel&gt;, &lt;channel&gt;</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'

©1989-2021 Lauterbach GmbH
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 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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’
STMTrace

Method-independent display and analysis of STM trace data

Format:  

<table>
<thead>
<tr>
<th>Command Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMTrace</td>
<td>The STMTrace command group can be used as a generic replacement for the above STM&lt;trace&gt; command groups.</td>
</tr>
<tr>
<td>&lt;sub_cmd&gt;</td>
<td>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).</td>
</tr>
<tr>
<td>&lt;stm_channel&gt;</td>
<td>The following, additional channels are available for the analysis of STP trace data:</td>
</tr>
<tr>
<td>STMTITS</td>
<td>Displays raw timestamp information of DxxTS messages. Only for TI Onchip traces.</td>
</tr>
<tr>
<td>STMMASTER</td>
<td>Displays the master ID of each message.</td>
</tr>
<tr>
<td>STMCHANNEL</td>
<td>Displays the channel ID of each message.</td>
</tr>
<tr>
<td>&lt;channel&gt;</td>
<td>For a description of the default channels, see &lt;trace&gt;.List &lt;items&gt;.</td>
</tr>
</tbody>
</table>

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

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

STMTrace.List STMMASTER STMCHANNEL CYcle Data TIme.Back
Example 2: In case of TI, the recommended way to display STP onchip data is:

```
STMTTrace.List STMMASTER STMCHANNEL Cycle Data STMITS 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**

<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> For a description of <strong>PrintfTrace.List</strong>, refer to &lt;trace&gt;.List</td>
<td></td>
</tr>
</tbody>
</table>

### 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></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><strong>Start of message</strong></td>
<td></td>
<td>D4TS, D8TS, D16TS, D32TS, D64TS</td>
</tr>
<tr>
<td><strong>Message body</strong></td>
<td></td>
<td>D4, D8, D16, D32, D64</td>
</tr>
<tr>
<td><strong>End of message</strong></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 16 bit 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><strong>Start of message</strong></td>
<td>n.a.</td>
<td>D32TS (0x????0001)</td>
</tr>
<tr>
<td><strong>Process ID</strong></td>
<td>n.a.</td>
<td>D32</td>
</tr>
<tr>
<td><strong>Target address</strong></td>
<td>n.a.</td>
<td>D32</td>
</tr>
<tr>
<td><strong>Source address</strong></td>
<td>n.a.</td>
<td>D32</td>
</tr>
<tr>
<td><strong>End of message</strong></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:

| 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
```
SYStem.CONFIG.CMI

Inform TRACE32 about CMI component

Format: SYStem.CONFIG.CMI<instance>,<sub_cmd>

<instance>:  1 | 2

<sub_cmd>:  <generic> | <component_specific>

<component_specific>: TraceID <id>

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

<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>
</tbody>
</table>

TraceID <id> | Sets the ATB trace ID of the corresponding CMI component.

Example:

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

See also

- CMI
- ‘Generic Subcommands, Parameters, and Options’ in ‘System Trace User's Guide’

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

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

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

See also

- CMI.EnableMessage
- CMI.Register
- CMITrace
- CMI.Mode
- CMI.RESet
- SYSTEM.CONFIG.CMI
- CMI.OFF
- CMI.SamplingWindow
- CMIBASE()
CMI.EnableMessage

Enables event or activity message generation

Format:  \texttt{CMI<instance>.EnableMessage.<message> [ON | OFF]}

\texttt{<message>}:
- \texttt{ClockDomain} (event message)
- \texttt{ClockFrequency8} (event message)
- \texttt{ClockFrequency4} (event message)
- \texttt{ClockSource} (event message)
- \texttt{DPLLmask} (event message)
- \texttt{TargetActivity8} (activity message)
- \texttt{TargetActivity4} (activity message)
- \texttt{InitiatorActivity8} (activity message)
- \texttt{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 \texttt{CMI<instance>.Mode EVenT} has been selected.

<table>
<thead>
<tr>
<th>\texttt{ClockDomain}</th>
<th>Trace clock domain state changes (on / off).</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ClockFrequency8}</td>
<td>Trace clock frequency changes (8-bit divider ratio).</td>
</tr>
<tr>
<td>\texttt{ClockFrequency4}</td>
<td>Trace clock frequency changes (4-bit divider ratio).</td>
</tr>
<tr>
<td>\texttt{ClockSource}</td>
<td>Trace clock source selection changes (MUX input).</td>
</tr>
<tr>
<td>\texttt{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 \texttt{CMI<instance>.Mode ACTivity} in addition.

| \texttt{TargetActivity8} | Count target activity cycles.  
(\texttt{If CMI<instance>.SamplingWindow.Size} >= 16) |
|-------------------------|--------------------------------------------------|
| \texttt{TargetActivity4} | Count target activity cycles.  
(\texttt{If CMI<instance>.SamplingWindow.Size} < 16) |
| \texttt{InitiatorActivity8} | Count initiator activity cycles.  
(\texttt{If CMI<instance>.SamplingWindow.Size} >= 16) |
| \texttt{InitiatorActivity4} | Count initiator activity cycles.  
(\texttt{If CMI<instance>.SamplingWindow.Size} < 16) |

See also
- \texttt{CMI}
- \texttt{CMI.state}
### CMI.Mode

**Set event or activity mode**

Format: \[\text{CMI}<\text{instance}>,\text{Mode \{EVENT | ACTivity\}}\]

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

**See also**
- \text{CMI}  
- \text{CMI.state}  

### CMI.OFF

**Switch CMI off**

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

Switches the CMI component off.

**See also**
- \text{CMI}  
- \text{CMI.state}  

### CMI.ON

**Switch CMI on**

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

Switches the CMI component on.

**See also**
- \text{CMI}  
- \text{CMI.state}  

©1989-2021 Lauterbach GmbH
CMI.Register

Display the CMI register

<table>
<thead>
<tr>
<th>Format:</th>
<th>CMI&lt;instance&gt;.Register [/&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;option&gt;:</td>
<td>SpotLight</td>
</tr>
<tr>
<td></td>
<td>CORE</td>
</tr>
</tbody>
</table>

Displays the CMI registers.

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

See also

- CMI
- CMI.state

CMI.RESet

Resets CMI settings to their defaults

| Format:       | CMI<instance>.RESet |

All CMI settings are reset to their defaults.

See also

- CMI
- CMI.state
CMI.SamplingWindow.SamplingWindow

See also
- CMI.SamplingWindow.CLocK
- CMI.SamplingWindow.Size
- CMI
- CMI.state

CMI.SamplingWindow.CLocK

Set sampling window ratio

Format:

\[
\text{CMI}<\text{instance}>.\text{SamplingWindow.CLocK} \ <\text{ratio}>
\]

Default: 1/1

<table>
<thead>
<tr>
<th>&lt;\text{ratio}&gt;</th>
<th>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.</th>
</tr>
</thead>
</table>

See also
- CMI.SamplingWindow

CMI.SamplingWindow.Size

Set sampling window size

Format:

\[
\text{CMI}<\text{instance}>.\text{SamplingWindow.Size} \ <\text{cycles}>
\]

Default: 1

<table>
<thead>
<tr>
<th>&lt;\text{cycles}&gt;</th>
<th>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.</th>
</tr>
</thead>
</table>

See also
- CMI.SamplingWindow

©1989-2021 Lauterbach GmbH
Opens the **CMI1.state** and **CMI2.state** window, where you can configure the CMI trace source 1 and 2.

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

A For descriptions of the commands in the **CMI<instance>.state** window, please refer to the **CMI.*** commands in this chapter.

**Example:** For information about **ON**, click **CMI.ON**.

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

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

---

**See also**

<table>
<thead>
<tr>
<th>CMI</th>
<th>CMI.EnableMessage</th>
<th>CMI.Mode</th>
<th>CMI.OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMI.ON</td>
<td>CMI.Register</td>
<td>CMI.RESet</td>
<td>CMI.SamplingWindow</td>
</tr>
</tbody>
</table>
This example for an OMAP4430 which has got two CMI components. In this case the components are addressed as CMI1 or CMI2, respectively.

```
CAnalyzer.AutoArm OFF ; In activity mode CMI messages are output permanently, so we
CAnalyzer.ARM ; must arm the CAnalyzer before the CMI is activated.

CMI1.RESET
CMI1.SamplingWindow.Size 15
CMI1.Mode ACTivity
CMI1.ON
...
CMI1.OFF ; Turn off CMI ...
CAnalyzer.OFF ; ... before shutting down the CAnalyzer.

CMITrace.List CYcle CMITA.<name1> ; Display cycle activity of target
CMITA.<name2> ; <name1> and <name2>.
```
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><strong>Format:</strong></th>
<th><code>CMITrace.&lt;sub_cmd&gt; [cmi_channel]... [channel]...</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cmi_channel&gt;</code>:</td>
<td>`CMICD.&lt;domain&gt;</td>
</tr>
<tr>
<td><code>&lt;sub_cmd&gt;</code></td>
<td>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).</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>For a description of <strong>CMITrace.List</strong>, refer to <code>&lt;trace&gt;.List</code></td>
</tr>
</tbody>
</table>

<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>

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

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

See also
- CMI
If the Coherent Mesh Network (CMN) component is not enabled for your specific device, use the following commands for configuration. At least the base address must be set in order to enable the CMN. If the addresses of DeBugBases are not set, TRACE32 tries to detect the DBG subcomponents on its own. Depending on the target device, this might fail. In this case, you have to specify the starting addresses of the CMN DBG subcomponent manually.

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

For descriptions of the generic subcommands, click here.

DeBugBases
All base addresses of the CMNs Debug/Trace subcomponents.

XYBits
Number of bits used to address the dimension of a single XP inside the mesh network.

Example:

```plaintext
SYStem.CONFIG.CMN.Base           AXI:0x50D00000
SYStem.CONFIG.CMN.DeBuGBases     AXI:0x50D30000  AXI:0x50D40000
SYStem.CONFIG.XYBits             2bit
```
CMN stands for Coherent Mesh Network. It is a scalable configurable coherent interconnect designed by ARM and used in high-end networking and enterprise compute applications. The CMN's Debug and Trace subcomponent allows for non-intrusive tracing of the messages sent or received at each CMN crosspoint (XP).

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

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

![CMN Configuration Interface](image)

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

### CMN.EnhancedFilter

Set an individual filter on a CMN XP

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

Configures the filter for a single watchpoint on a single CMN XP. Only the filtered messages are visible to the ATB. For further details, refer to the *Arm CoreLink CMN-600 Coherent Mesh Network Technical Reference Manual*.

- `<x_coord>`: X coordinate of the XP which is configured
  - The used value can range from 0. to 7.
- `<y_coord>`: Y coordinate of the XP which is configured
  - The used value can range from 0. to 7.
**Example:**

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

CMN.EnhancedFilter 2. 1. 0. 0x1941 0x80000000 0xFFFFFFFFE07FFFFFFF
CMN.EnhancedFilter 2. 1. 1. 0x1911 0x123450 0xFFF000000000000F

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

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

---

**NOTE:** This command resets all global filters which have been set by **CMN.Opcodel**, **CMN.SourceID**, and **CMN.TraceChannel**.
CMN.Init

Initialize CMN on target

Format: CMN.Init

Initializes the CMN registers on the target.

CMN.OFF

Switch CMN trace off

Format: CMN.OFF

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

CMN.ON

Switch CMN trace on

Format: CMN.ON

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

CMN.Opcde

Set global filter for opcode

Format: CMN.Opcde <value>

Default: None

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

NOTE: Depending on the selected CMN channel, the same opcode value can describe different trace packets.

This command resets the configurations done by CMN.EnhancedFilter
CMN.PortRoute
Select output of CMN data

Format:  
CMN.PortRoute [AUTO | Analyzer | CAnalyzer | Onchip]

Default: AUTO

Prepares the selected trace hardware for CMN trace capture.

AUTO  
Automatic detection

Analyzer  
PowerTrace (via TPIU)

CAnalyzer  
Compact-Analyzer: CombiProbe or µTrace (MicroTrace)

Onchip  
Onchip trace buffer (ETB, ETF or ETR)

CMN.Register
Display CMN register

Format:  
CMN.Register [<file>] [/<option>]

<option>:  
SpotLight | DualPort | Track | CORE <core_number>

Displays the CMN registers.

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

CMN.RESet
Reset CMN settings

Format:  
CMN.RESet

Resets all CMN settings to default.
**CMN.NodeID**

Set global filter for xp source

**Format:**

```
CMN.NodeID <xp_nodeid>
```

```
CMN.SourceID (deprecated)
```

Default: None

This command sets a global filter for flits originated at the specified XP.

```
<xp_nodeid> XP node id in the form “(X,Y,Port,DevID)”
```

**NOTE:**

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

**Example:**

```
;Trace all CMN REQ flits originated by Subdevice at port 1 of XP (2,3)
CMN.NodeID “(2,3,1,0)”
```

---

**CMN.state**

Display CMN settings

**Format:**

```
CMN.state
```

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

---

©1989-2021 Lauterbach GmbH

System Trace User’s Guide 62
For descriptions of the commands in the CMN.state window, please refer to the CMN.* commands in this chapter.

**Example:** For information about ON, see CMN.ON.

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

---

**CMN.SyncPeriod**

Set period of synchronisation packet

<table>
<thead>
<tr>
<th>Format:</th>
<th>CMN.SyncPeriod &lt;period&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;period&gt;</td>
<td>256bytes</td>
</tr>
</tbody>
</table>

Default: 512bytes

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

**CMN.TimeStampPeriod**

Set period of timestamp packet

<table>
<thead>
<tr>
<th>Format:</th>
<th>CMN.TimeStampPeriod &lt;period&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;period&gt;</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

Default: Disabled

Configures the timestamp packet insertion period in clock cycles.

**CMN.TraceChannel**

Set global filter for CMN channel

<table>
<thead>
<tr>
<th>Format:</th>
<th>CMN.TraceChannel &lt;channel&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;channel&gt;</td>
<td>REQ</td>
</tr>
</tbody>
</table>

Default: REQ
Sets a global filter for the specified CMN channel at each XP.

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

---

### CMN.TraceID

Sets trace ID

Format:  

```
CMN.TraceID  <id> | <id_start>--<id_end>
```

Default: 0x61

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

- `<id_start>` Must be an even number.

---

### CMN.TracePriority

Set priority for CMN manually

Format:  

```
CMN.TracePriority  <priority>
```

TRACE32 automatically assigns an appropriate priority to the CMN. This command allows the user to change the priority for the CMN trace information.
This example for a typical onchip setup with a NeoverseN1.

```
CMN.Reset
CMN.TraceID 3.
CMN.TraceChannel RSP ; Configure CMN to trace
CMN.Opscode 0x3 ; only RetryAck packets
CMNTrace.METHOD Onchip
CMNTrace.ARM
...
CMNTrace.OFF
CMNTrace.List CYcle AAddress SRCNODE TGTNODE ; Display trace packets
; with CMN operation,
; phys. address,
; source node,
; and target node
```

SYStem.CONFIG.CMN.Base AXI:0x50d00000 ; Informs TRACE32 about
SYStem.CONFIG.CMN.DebuGBases AXI:0x50d30000 ; the CMN component at
SYStem.CONFIG.CMN.XYBits 2bit ; CoreSight ETF1
SYStem.CONFIG.ETF1.Base DAP:0x80900000
SYStem.CONFIG.ETF1.ATBSource CMN
CMNTrace
Display and analyze CMN trace data

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

Format: CMNTrace.<sub_cmd> [<cmn_channel>…] [<channel>…]

<cmn_channel>: CYcle | SRCNODE | TGTNODE

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

<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 CMNTrace.List, refer to <trace>.List

<cmi_channel> The following, additional channels are relevant for the analysis of CMN trace data:

CYcle Name of the trace packet type according to the opcode and channel
SRCNODE CMN subcomponent identifier of the trace packet source in the following format: (X,Y,Port,DevID)
TGTNODE CMN subcomponent identifier of the trace packet destination in the following format: (X,Y,Port,DevID)

Example:

; Display cycle activity between SRCNODE and TGTNODE
CMNTrace.List CYcle SRCNODE TGTNODE
CPTracer Component

No extra TRACE32 configuration → Configure target → Record trace data → Display trace data

CPTracer

Configure CPTracer component on target

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.

See also
- CPTracer.<aggregator>.<probe>.ADDRessHIGH
- CPTracer.<aggregator>.<probe>.ADDRessLOW
- CPTracer.<aggregator>.<probe>.PERiod
- CPTracer.<aggregator>.OFF
- CPTracer.<aggregator>.SYNC
- CPTracer.state
- CPTracerTrace
CPTracer.RESet

Resets CPT settings to their defaults

Format: CPTracer.RESet

All CPT settings are reset to their defaults.

See also
- CPTracer
- CPTracer.state

CPTracer.state

Display CPT settings

Format: CPTracer.state

Shows the CPT setup window.

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**

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

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

**See also**
- CPTracer
- CPTracer.state

**CPTracer.<aggregator>.OFF**

Switch aggregator off

Perform 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

---

**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

---

©1989-2021 Lauterbach GmbH
CPTracer.<aggregator>.<probe>.ADDRessHIGH

Upper filter address

Format:

```
CPTracer.<aggregator>.<probe>.ADDRessLOW <value>
```

Default: 0xFFFFFFFFFFFF

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

See also
- CPTracer
- CPTracer.state

---

CPTracer.<aggregator>.<probe>.CHannel

Filter by channel ID

Format:

```
CPTracer.<aggregator>.<probe>.CHannel <bitmask>
```

Default: 0yXXXXXXXXXXXX

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

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

<table>
<thead>
<tr>
<th>Format:</th>
<th>CPTracer.&lt;aggregator&gt;.&lt;probe&gt;.OPeration &lt;mode&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;:</td>
<td>OFF, LATency, THRoUput, TRANSaction</td>
</tr>
</tbody>
</table>

Default: OFF.

Defines which type of trace packet is to be generated.

- **OFF**
  - No trace packets.

- **LATency**
  - Trace packets carrying latency information.

- **THRoUput**
  - Trace packets carrying throughput information.

- **TRANSaction**
  - Trace packets carrying transaction information.

**See also**

- CPTracer
- CPTracer.state

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

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

Default: 0x3FFF

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

**See also**

- CPTracer
- CPTracer.state
CPTracer.<aggregator>.<probe>.RouteID

Filter by route ID

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

Default: 0yXXXXXXXXXXXX

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

See also

- CPTracer
- CPTracer.state
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.

- **Format:**
  
  CPTTracerTrace.<sub cmd> [cpt_channel>...] [channel>...]

  - 
    - <cpt_channel>:
      - **PRobe** | **CYcle** | **Address** | **LAT.<xxx>** | **TRANS.<xxx>** | **THRU.<xxx>** | **Time.<xxx>**

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

**<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).

**<cpt_channel>**

The following channels are relevant for the analysis of CPT trace data:

- **<xxx>**
  
  A list of all valid replacements for the placeholder **<xxx>** will be displayed as softkeys in TRACE32 as soon as the dot `'.'` is entered in the TRACE32 command line.

- **PRobe**
  
  Name of the probe. See example.

- **CYcle**
  
  Type of trace packet.

- **Address**
  
  Traced address of transaction packet. Also see note below.

- **LAT.<xxx>**
  
  Latency packet specific information **<xxx>**. See example.

- **TRANS.<xxx>**
  
  Transaction packet specific information **<xxx>**.

- **THRU.<xxx>**
  
  Throughput packet specific information **<xxx>**.

- **Time.<xxx>**
  
  Timing information.

**<channel>**

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

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

NOTE: In case of transaction packets, the Address column in the t32marm executable (32 bit targets) only outputs the lower 32 bits of an address. The upper bits are available via channel TRANS.highaddress.
The Address column in the t32marm64 executable (64 bit targets) already outputs 64 bit addresses and hence does have a TRANS.highaddress channel.

See also

- CPTracer
### OCP Component

#### SYStem.CONFIG.OCP

**Inform TRACE32 about OCP component**

**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.

<table>
<thead>
<tr>
<th>&lt;generic&gt;</th>
<th>For descriptions of the generic subcommands, click here.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 4</td>
<td>Currently only supported for OMAP4.</td>
</tr>
<tr>
<td>TraceID</td>
<td>Sets the STM master ID of the OCP component.</td>
</tr>
</tbody>
</table>

**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**

- OCP
- ‘Generic Subcommands, Parameters, and Options’ in ‘System Trace User's Guide’
OCP stands for OpenCoreProtocol WatchPoint (OCP-WP). The OCP-WP monitors OCP requests directed to a selected target attached to the L3 interconnect of the OMAP4. Tracing the bus traffic is non-intrusive and enables the developer to capture all requests addressed to a target or only a subset of it defined by up to four different filters (see OCP.TraceFilter<x> commands).

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

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

See also
- OCP.AutoIDLE
- OCP.DebugPort
- OCP.PRESet
- OCP.TraceOFF
- OCP.state
- OCP.TraceON
- OCP TriggerOut<x>
- SYstem.CONFIG.OCP

OCP.AutoIDLE

Format: OCP.AutoIDLE [ON | OFF]

Default: OFF.

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

See also
- OCP
- OCP.state

OCP.DebugPort

Select target to be traced

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

Default: OFF.

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

See also
- OCP
- OCP.state

OCP.OFF

Switch OCP off

Format: OCP.OFF

Switches the OCP component off.

See also
- OCP
- OCP.state
**OCP.ON**

Switches the OCP component on.

See also
- OCP
- OCP.state

**OCP.Register**

Display OCP registers

Format: `OCP.Register [<file>] [/<option>]`

`<option>`:
- SpotLight
- DualPort
- Track
- AlternatingBackGround
- CORE `<core_number>`

Displays the OCP registers.

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

See also
- OCP
- OCP.state

**OCP.RESet**

Reset OCP settings to their defaults

Format: `OCP.RESet`

All OCP settings are reset to their defaults.

See also
- OCP
- OCP.state
OCP.state

**Format:**

OCP.state

Shows the OCP setup window.

A For descriptions of the commands in the OCP.state window, please refer to the OCP.* commands in this chapter.

**Example:** For information about **OFF**, see **OCP.OFF**.

See also

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

Name a filter

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

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

Example:

OCP.TraceFilter0.NAME "Filter0"

See also
- OCP.TraceFilter

OCP.TraceFilter<x>.MCmd

Filters traffic by transaction type

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

Default: ALL.

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

See also
- OCP.TraceFilter
**OCP.TraceFilter<x>.Initiator**

Filters traffic by transaction initiator

Format: \texttt{OCP.TraceFilter<x>.Initiator [ALL | <initiator>]}

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

\textbf{See also}\n
- \texttt{OCP.TraceFilter}

---

**OCP.TraceFilter<x>.REQinfo**

Filters traffic by transaction qualifier

Format: \texttt{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 '0ybb', whereas \(b = [0, \text{Cleared} , 1, \text{Set} , X], \text{Don't Care}\).

\textbf{Example:}\n
\begin{verbatim}
OCP.TraceFilter0.REQinfo MReqDomain.0y11X ;Trace transactions which
;have the two upper bits
;set ignore the state of
;the lowest bit.
\end{verbatim}

\textbf{See also}\n
- \texttt{OCP.TraceFilter}
OCP.TraceEnable

Filter OCP traffic by address range

Format:

\texttt{OCP\_TraceEnable \langle range\rangle}

Default: 0x00000000-0xffffffff

OCP traffic is only captured if the address is within the specified \texttt{\langle range\rangle}. The range must be specified as the offset from the base address of the selected debug port (\texttt{OCP\_DebugPort}), not to the global address! \texttt{OCP\_TraceEnable} and \texttt{OCP\_TraceON} / \texttt{OCP\_TraceOFF} cannot be applied at the same time!

Example:

\begin{verbatim}
; Debug port base address = 0xa0001000
; Range to be monitored = 0xa0001000 to 0xa0001020
OCP\_TraceEnable 0x00000000--0x00000020
\end{verbatim}

See also
- OCP
- OCP\_state

OCP.TraceOFF

Stop tracing

Format:

\texttt{OCP\_TraceOFF [EMU1 | \langle address\rangle]}

Stops tracing if the trigger condition or address match occurs. Tracing will continue on an \texttt{OCP\_TraceON} condition.

\texttt{OCP\_TraceEnable} and \texttt{OCP\_TraceOFF} cannot be applied at the same time!

\texttt{OCP\_TriggerOut\(<x>\)} and \texttt{OCP\_TraceOFF EMU1} cannot be used at the same time!

Default: \texttt{OCP\_TraceEnable}

- \texttt{EMU1} Stops tracing upon a HIGH-TO-LOW transition of the EMU1 trigger input.
- \texttt{\langle address\rangle} Stops tracing upon an address match.

See also
- OCP
- OCP\_state
Start tracing

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

**OCP.TraceEnable** and **OCP.TraceON** cannot be applied at the same time!

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

Default: **OCP.TraceEnable**

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

See also
- OCP
- OCP.state

### OCP.TriggerOut<x>

Generate trigger event

Format:

```
OCP.TriggerOut [\<address\> | \<range\>]
```

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

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

See also
- OCP
- OCP.state
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>... ]
```

### Example

```
See also

- OCP

```
PMI Component

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>&lt;generic&gt;</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

- PMI
- ‘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: \texttt{PMI.EnableMessage.<event\_msg> [ON | OFF]}

\texttt{<event\_msg>}: LogicVoltage
MemoryVoltage
LogicPower
MemoryPower

Default: OFF.

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

- \textbf{LogicVoltage}: Voltage levels of logic blocks.
- \textbf{MemoryVoltage}: Voltage levels of memory blocks.
- \textbf{LogicPower}: Power FSM states of logic blocks.
- \textbf{MemoryPower}: Power state of memory blocks.

See also
- PMI

---

PMI.OFF

Switch PMI off

Format: \texttt{PMI.OFF}

Switches the PMI component off.

See also
- PMI
PMI.ON  

Switch PMI on

Format: **PMI.ON**

Switches the PMI component on.

See also
- PMI

PMI.Register  

Display the PMI registers

Format: **PMI.Register** [<option>]

<option>: SpotLight | DualPort | Track | AlternatingBackGround
CORE <core_number>

Displays the PMI registers.

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

See also
- PMI

PMI.RESet  

Resets PMI settings to their defaults

Format: **PMI.Reset**

All PMI settings are reset to their defaults.

See also
- PMI
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

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
PMI.state

Display PMI settings

Format: PMI.state

Shows the PMI setup window.

A For descriptions of the commands in the **PMI.state** window, please refer to the **PML.*** 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.
Format: `PMITrace.<sub_cmd> [pmi_channel] [...] [channel] ...`

<table>
<thead>
<tr>
<th>pmi_channel:</th>
<th>PMILV.&lt;domain&gt;</th>
<th>PMILVOFF</th>
<th>PMIMV.&lt;domain&gt;</th>
<th>PMILP.&lt;domain&gt;</th>
<th>PMIMP.&lt;domain&gt;</th>
<th>PMISTAT</th>
<th>PMILAT</th>
<th>CYcle</th>
</tr>
</thead>
</table>

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.

For descriptions of the subcommands, please refer to the general `<trace>` command descriptions in “General Commands Reference Guide T” (general_ref_t.pdf).

**Example**: For a description of `PMITrace.List`, refer to `<trace>.List`.

### <pmi_channel>

The following channels are relevant for the analysis of PMI trace data:

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

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

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

See also

- PMI
StatCol Component (Statistics Collector)

SYStem.CONFIG.SC

Inform TRACE32 about StatCol component

Format:

\[ \text{SYStem.CONFIG.SC.}<\text{sub\_cmd}> \]

<sub\_cmd>:

\[ <\text{generic}> | <\text{component\_specific}> \]

(component\_specific):

\[ \text{TraceID} <\text{id}> \]

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

<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

- StatCol
- 'Generic Subcommands, Parameters, and Options' in 'System Trace User’s Guide'
The NoC statistics collector provides information about the workload of an on-chip 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>.REquestEvent
- StatCol.RESet
- StatColTrace
- StatCol.<probe>.Counter
- StatCol.<probe>.ON
- StatCol.<probe>.ReSPonseEvent
- StatCol.state
- SYStem.CONFIG.SC
StatCol.RESet

Resets all statistics collector settings to their default values.

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
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

Format:

```
StatCol.<probe>.RequestEVENT.<event>
```

Default: NONE.

Selects the event detector for the probe's response link. See `StatCol.<probe>.RequestEVENT`.

**See also**

- StatCol
- StatCol.state

---

**StatCol.<probe>.CollectTime**

Set up collection period

Format:

```
StatCol.<probe>.CollectTime <cycles>
```

Default: 255.

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

**See also**

- StatCol
- StatCol.state
StatCol.<probe>.Counter <counter> ADDRM<counter> ADDRM<counter> Filter max address

Format: StatCol.<probe>.Counter <counter> ADDRESSMAX.<value>

Default: 0.

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

See also
- StatCol.<probe>.Counter

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

Format: StatCol.<probe>.Counter <counter> ADDRESSMIN.<value>

Default: 0.

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

See also
- StatCol.<probe>.Counter
StatCol.<probe>.Counter <counter> ADDREN

Enable address filtering

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

Default: OFF.

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

See also

- StatCol.<probe>.Counter

StatCol.<probe>.Counter <counter> EventInfo

Select ‘EventInfo’ to count

Format: StatCol.<probe>.Counter <counter> EventInfo.<eventinfo>

<Eventinfo>: LENgth
               PRESsure
               LATency

Default: LENgth.

Detects additional event information:

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

See also

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

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

Default: 0.

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

See also
■ StatCol.<probe>.Counter

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

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

Default: 0.

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

See also
■ StatCol.<probe>.Counter
### 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:

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIT</td>
<td>Increment the counter by one each time an event has passed the counter’s filter. (See StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;filter&gt; commands).</td>
</tr>
<tr>
<td>MINMAX</td>
<td>Increment the counter by one each time the selected EventInfo is within the range Min.&lt;value&gt; and Max.&lt;value&gt;.</td>
</tr>
<tr>
<td>ADD</td>
<td>Add the selected EventInfo value to the counter if an event has passed the counter’s filter. (See StatCol.&lt;probe&gt;.Counter &lt;counter&gt; Filter &lt;filter&gt; commands)</td>
</tr>
<tr>
<td>AND</td>
<td>Increment the counter by one if an event has passed all filters of &lt;probe&gt;.</td>
</tr>
<tr>
<td>OR</td>
<td>Increment the counter by one if an event has passed at least one of all filters of &lt;probe&gt;.</td>
</tr>
<tr>
<td>REQ</td>
<td>Increment the counter by one each time a request message is detected on any port of &lt;probe&gt;.</td>
</tr>
<tr>
<td>RSP</td>
<td>Increment the counter by one each time a response message is detected on any port of &lt;probe&gt;.</td>
</tr>
<tr>
<td>ALL</td>
<td>Increment the counter by one each time a response or request message is detected on any port of &lt;probe&gt;.</td>
</tr>
<tr>
<td>EXT</td>
<td>Increment the counter by one each time the external event input is sampled high.</td>
</tr>
</tbody>
</table>

**See also**
- StatCol.<probe>.Counter
StatCol.<probe>.Counter <counter> Filter

Set filter criteria

Format:  

\[
\text{StatCol.<probe>.Counter <counter> Filter <filter> [<item>][,<value> | <mask>]} \\
\]

\(<item>:\)

- MaSTerADDRess
- ReaD
- WRite
- ERRor
- REQuestUserInfo
- ReSPonseUserInfo
- SLaVeADDRess

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

MaSTerADDRess  
Master address (NTTP) or MConnId (OCP).

ReaD  
Read bit.

WRite  
Write bit.

ERRor  
Error bit (NTTP only).

REQuestUserInfo  
RequestUserInfo bits (NTTP only).

ReSPonseUserInfo  
ResponseUserInfo bits (NTTP only).

SLaVeADDRess  
Slave address (NTTP only).

\(<value>:\)

(Hexa)decimal, octal or binary value that defines the required packet item.

\(<mask>:\)

(default: Don’t care)

Bitmask of format ‘0y…..’:

- \(x = \text{don’t care}\)
- \(1 = \text{set}\)
- \(0 = \text{cleared}\)

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

<table>
<thead>
<tr>
<th>Input port</th>
</tr>
</thead>
</table>

| 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

| 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

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

Switches the filter on.

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

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

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

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

### OFF

Clears all filters.

<table>
<thead>
<tr>
<th>Mandatory additional configuration</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional additional configuration</td>
<td>-</td>
</tr>
</tbody>
</table>
### 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>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>

### 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>
<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>

**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>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>
AvgLatDist

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

<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>

Mandatory additional configuration -

Optional additional configuration

StatCol.<probe>.Counter <counter> Filter <element> SLaVeADDResS
StatCol.<probe>.Counter <counter> Filter <element> MaSTerADDResS
StatCol.<probe>.Counter <counter> Filter <element> REQUserInfo

See also
- StatCol.<probe>.Counter
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

... ; Turn off the statistics
... before shutting down ; collector ... ; the CAnalyzer.

StatCol.SDRAM.OFF
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.

**Format:**

```
StatColTrace.<sub_cmd> [statcol_channel] [channel]...
```

**<statcol_channel>:**

```
SCC0.<probe> | SCC1.<probe> | SCC2.<probe> | SCC3.<probe> | SCC4.<probe>
```

For descriptions of the subcommands, please refer to the general `<trace>` command descriptions in “**General Commands Reference Guide T**” (general_ref_t.pdf).

**Example:** For a description of **StatColTrace.List**, refer to `<trace>.List`.

---

**<statcol_channel>**

The following channels are relevant for the analysis of StatCol trace data:

- **<probe>**
  
  A list of all valid replacements for the placeholder `<probe>` will be displayed as softkeys in TRACE32 as soon as the dot '.' is entered in the TRACE32 command line.

- **SCC0.<probe>**
  
  Value of statistics collector counter 0.

- **SCC1.<probe>**
  
  Value of statistics collector counter 1.

- **SCC2.<probe>**
  
  Value of statistics collector counter 2.

- **SCC3.<probe>**
  
  Value of statistics collector counter 3.

- **SCC4.<probe>**
  
  Value of statistics collector counter 4.

---

**<channel>**

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

**Example:**

```
; Display trace data
StatColTrace.List CYCle SCC0 SCC1 SCC2 SCC3
```
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><code>SYStem.CONFIG.&lt;component&gt;.&lt;generic&gt;</code></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 <code>&lt;parameter&gt;</code></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 <code>SYStem.CONFIG.&lt;component&gt;</code> command: CMI, OCP, PMI, SC, STM, CMN.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;generic&gt;</code></td>
<td>Generic subcommands of the <code>SYStem.CONFIG.&lt;component&gt;</code> commands. For descriptions of the generic subcommands, see:</td>
</tr>
<tr>
<td></td>
<td>• <code>SYStem.CONFIG.&lt;component&gt;.Base</code></td>
</tr>
<tr>
<td></td>
<td>• <code>SYStem.CONFIG.&lt;component&gt;.Name</code></td>
</tr>
<tr>
<td></td>
<td>• <code>SYStem.CONFIG.&lt;component&gt;.RESet</code></td>
</tr>
<tr>
<td></td>
<td>• <code>SYStem.CONFIG.&lt;component&gt;.view</code></td>
</tr>
</tbody>
</table>
**SYStem.CONFIG.<component>.Base**

Base address of a component

Format: `SYStem.CONFIG.<component>.Base <parameter>`

<parameter>: NONE | <address>

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

<table>
<thead>
<tr>
<th>NONE</th>
<th>Removes the base address of the &lt;component&gt;.</th>
</tr>
</thead>
</table>

**SYStem.CONFIG.<component>.Name**

Name of a component

Format: `SYStem.CONFIG.<component>.Name <name> [/<option>]`

<option>: CORE <number> | CONTinue

Assigns a user-defined name to a component.

<table>
<thead>
<tr>
<th>&lt;name&gt;</th>
<th>Parameter Type: String. User-defined names for &lt;components&gt; allow you to distinguish between different instances having the same parameters, such as the same addresses on different buses.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE</td>
<td>Sets partial addresses.</td>
</tr>
<tr>
<td>CONTinue</td>
<td>Collects calls without triggering any action until the next call without the CONTinue parameter.</td>
</tr>
</tbody>
</table>

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

Reset of a component

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

Resets the settings of the <component>.

©1989-2021 Lauterbach GmbH
Opens the **SYS**tem**.CONF**IG.<**component**>.**.view** window, displaying the settings of the `<component>`.

**Example**: For information about **RE**Set for the component **STM**, see **SYS**tem**.CONF**IG.<**component**>.**.RESet**.

**FAQ**

Please refer to our Frequently Asked Questions page on the Lauterbach website.