LAUTERBACH
DEVELOPMENT TOOLS

GTL Debug Back-End

Release 09.2023

MANUAL

GTL Debug Back-End

GTL Debug Back-End

Version 10-Oct-2023

History

26-Aug-13	Initial version.

Introduction

The Generic Transactor Library (GTL) is used to interact with a RTL simulators or emulators. This document describes how to load a GTL plug-in library and how to adapt TRACE32 for special use cases.

Related Documents

- **"T32Start"** (app_t32start.pdf): The T32Start application assists you in setting up multicore / multiprocessor debug environments, and software-only debug environments. T32Start is only available for Windows.

  For more information about software-only debug environments, please refer to: **"Software-only Debugging (Host MCI)"** (app_t32start.pdf).

Contacting Support

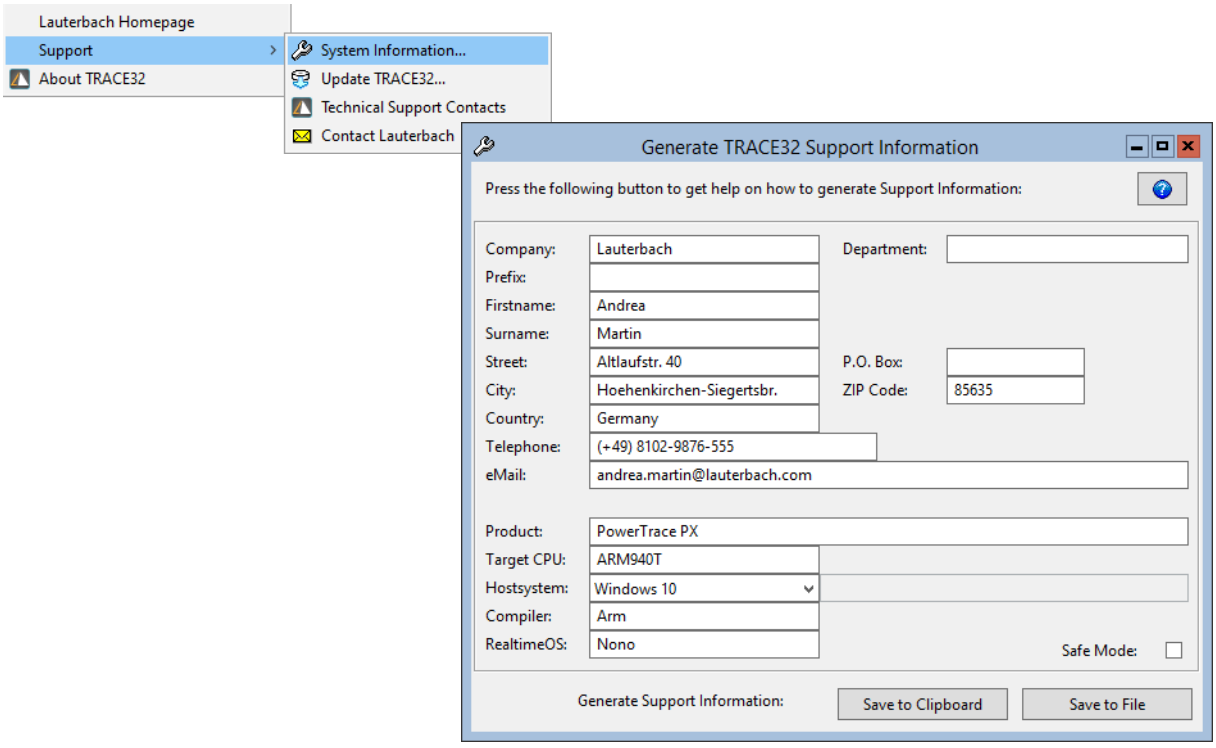
Use the Lauterbach Support Center: https://support.lauterbach.com

- To contact your local TRACE32 support team directly.
- To register and submit a support ticket to the TRACE32 global center.
- To log in and manage your support tickets.
- To benefit from the TRACE32 knowledgebase (FAQs, technical articles, tutorial videos) and our tips & tricks around debugging.

Or send an email in the traditional way to support@lauterbach.com.

Be sure to include detailed system information about your TRACE32 configuration.

1. To generate a system information report, choose **TRACE32** > **Help** > **Support** > **Systeminfo**.



NOTE:	Please help to speed up processing of your support request. By filling out the system information form completely and with correct data, you minimize the number of additional questions and clarification request e-mails we need to resolve your problem.

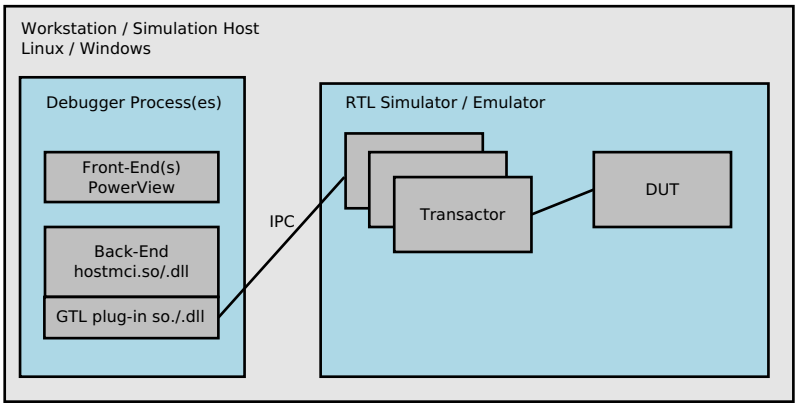
2. Preferred: click **Save to File**, and send the system information as an attachment to your e-mail.
3. Click **Save to Clipboard**, and then paste the system information into your e-mail.

Abbreviations and Definitions

AMP	Asymmetric Multi-Processing
Back-end	A back-end contain high performance multi-core debugger driver and the interface to the simulator / emulator.
DUT	Device Under Test. A DUT is the part of the model that is being tested.
GTL	Generic Transactor Library. A plug-in interface for the debuggers back-end to access the transactors.
IPC	Inter Process Communication. A method to communicate between multiple processes of an Operating System e.g. Semaphores, Shared Memory, File Pipes, TCP
RTL	Register Transfer Level. Models of this level describe a digital system by registers, signals and processes, not using a complete net list with timing information.
Simulator	A simulator executes a model on RTL level without using special acceleration hardware.
SMP	Symmetric Multi-Processing
Transactor	A transactor is a part of a system that interacts with the DUT in order to analyze and control the DUT by an external tool.

System Architecture

The total system consists of two process groups. The debugger processes containing front-end and back-end and the RTL simulation / emulation. The debuggers back-end (hostmci) contain the high performance multicore debug driver that is also used together with real Lauterbach Hardware. The debuggers back-end is extended by a third party GTL plug-in to access transactors within the simulator/emulator. The GTL plug-in and the RTL simulator/emulator communicate by a proprietary protocol using Inter Process Communication of the Operating System.



The debuggers back-end need to run with a low latency to the simulation/emulation due to the very high amount of accesses to the transactors. Therefore the back-end and the RTL simulator/emulator should run at the same machine.

The debuggers front-ends (PowerView) can run at a different machine. Multiple PowerView instances can be connected to one back-end in oder to perform AMP debugging.

PowerView System Configurations

The TRACE32 PowerView instances can be set up in different ways.

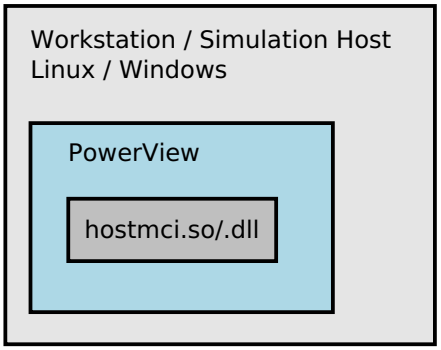
1. A single TRACE32 PowerView instance runs on the same host as the back-end, see Setup 1. This configuration can't handle AMP debug scenarios.
2. Multiple TRACE32 PowerView instances run on the same host as the back-end, see Setup 2.
3. The TRACE32 PowerView instances run on a dedicated workstation; the back-end runs on another host, see Setup 3.

The Lauterbach Debug Driver library (`hostmci.so` for Linux/Mac users and `hostmci.dll` for Windows users) can be integrated into the TRACE32 PowerView application or run as a separate process, called `t32mciserver`. Running it as a separate process provides two main benefits:

1. The MCI server can execute on one host, whilst one or more instances of TRACE32 PowerView execute on another host.
2. Multiple instances of TRACE32 PowerView can execute on a single host, sharing the MCI connection.

Setup 1

Setup with a single TRACE32 PowerView instance running on the same host as the back-end:

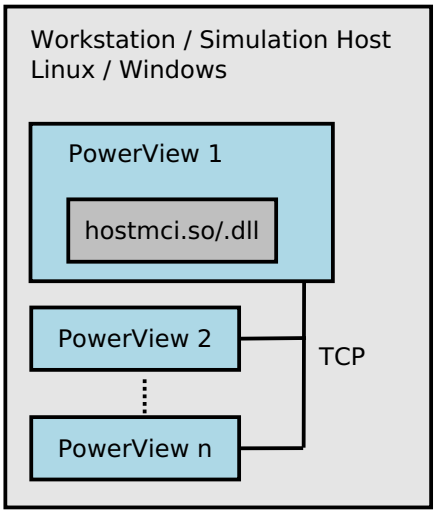


Modify the config.t32 file as follows:

```
PBI=MCILIB                          ; configure system to use hostmci.so
```

Setup 2

Setup with multiple TRACE32 PowerView instances (AMP) running on the same host as the back-end:

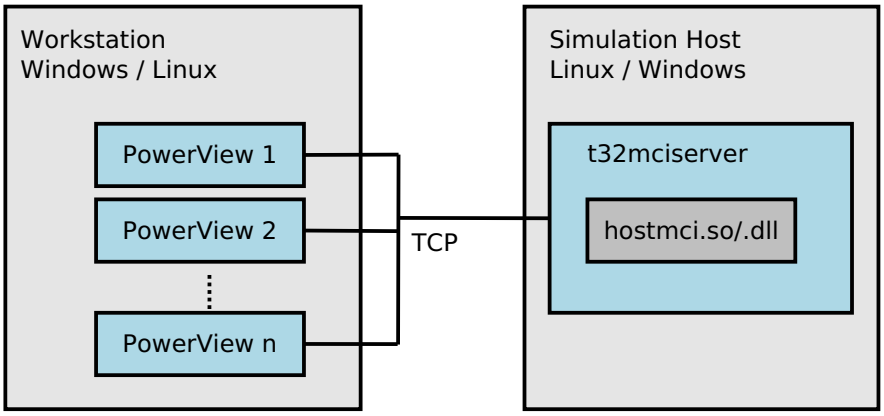


Modify the config.t32 as follows:

```
PBI=MCISERVER                ; set up the usage of hostmci.so and open
PORT=30000                   ; server at 30000 for the first instance.
INSTANCE=AUTO                ; consecutive number of instance or AUTO
```

Setup 3

Setup with multiple TRACE32 PowerView instances (AMP) running on another host:



Start t32mciserver on the simulation host:

```
./t32mciserver port=30000        ; start t32mciserver at port 30000
```

Modify the config.t32 file as follows:

```
PBI=MCISERVER                    ; set up connection to t32mciserver
NODE=192.168.0.1                 ; connect to IP 192.168.0.1
PORT=30000                       ; at port 30000
INSTANCE=AUTO                    ; consecutive number of instances
DEDICATED                        ; avoid to fall into Setup2 case
```

Linux example: To start TRACE32 PowerView with a specific config file, use e.g.:

```
bin/pc_linux/t32marm -c config.t32
```

A typical start sequence is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

```
; select GTL as back-end and a certain port
SYStem.CONFIG.DEBUGPORT GTL0

; (optional) tell the system how to connect to the simulation server
SYStem.GTL.SERVERCONFIG "server1:10000"

; (optional) pass configuration option to connect to the model
SYStem.GTL.MODELCONFIG "OPTION1=0|OPTION2=1"

; (optional) tell the system to connect to a certain DUT
SYStem.GTL.MODELNAME "MODEL_JTAG"

; tell the system the usage of transactors
SYStem.GTL.JTAGPROBENAME "JTAGPROBE0"

; library name of GTL plug-in
SYStem.GTL.LIBname "gtlplugin.so"

; configure usage of model time base instead host base to avoid timeouts
; while the emulation is paused.
SYStem.VirtualTiming.TimeinTargetTime ON
SYStem.VirtualTiming.PauseinTargetTime ON

; continue with CPU configuration
SYStem.CPU CortexM3        ; select CPU
SYStem.JtagClock 1Mhz      ; setup JTAG frequency
SYStem.Up                  ; connect to the emulation
```

Additional Commands to Configure ARM Bus Transactors

SYStem.GTL.ARMDAPNAME	Configure system wide DAP transactor
SYStem.CONFIG.DAPNAME	Configure and override DAP transactor for core DAP instance 1 accesses
SYStem.CONFIG.DAP2NAME	Configure and override second DAP transactor for core DAP instance 2 accesses
SYStem.CONFIG.DEBUGBUSNAME	Configure APB transactor to debug registers of the core
SYStem.CONFIG.APBNAME	Configure APB transactor for APB: memory class
SYStem.CONFIG.DAP2DEBUGBUSNAME	Configure APB bus used for DAP2: memory class

SYStem.CONFIG.DAP2APBNAME	Configure APB transactor for APB2: memory class
SYStem.CONFIG.MEMORYBUSNAME	Configure AHB transactor that is used for E: access when the CPU is running
SYStem.CONFIG.AHBNAME	Configure AHB transactor that is used for AHB: memory class
SYStem.CONFIG.DAP2MEMORYBUSNAME	(currently not in use)
SYStem.CONFIG.DAP2AHBNAME	Configure AHB transactor that is used for AHB2: memory class
SYStem.CONFIG.AXINAME	Configure AXI transactor that is used for AXI: memory class
SYStem.CONFIG.DAP2AXINAME	Configure second AXI transactor that is used for AXI2: memory class

In case the JTAG probe transactor is not used, it is recommended to configure an additional GPIO transactor to modify and sense the system reset signal.

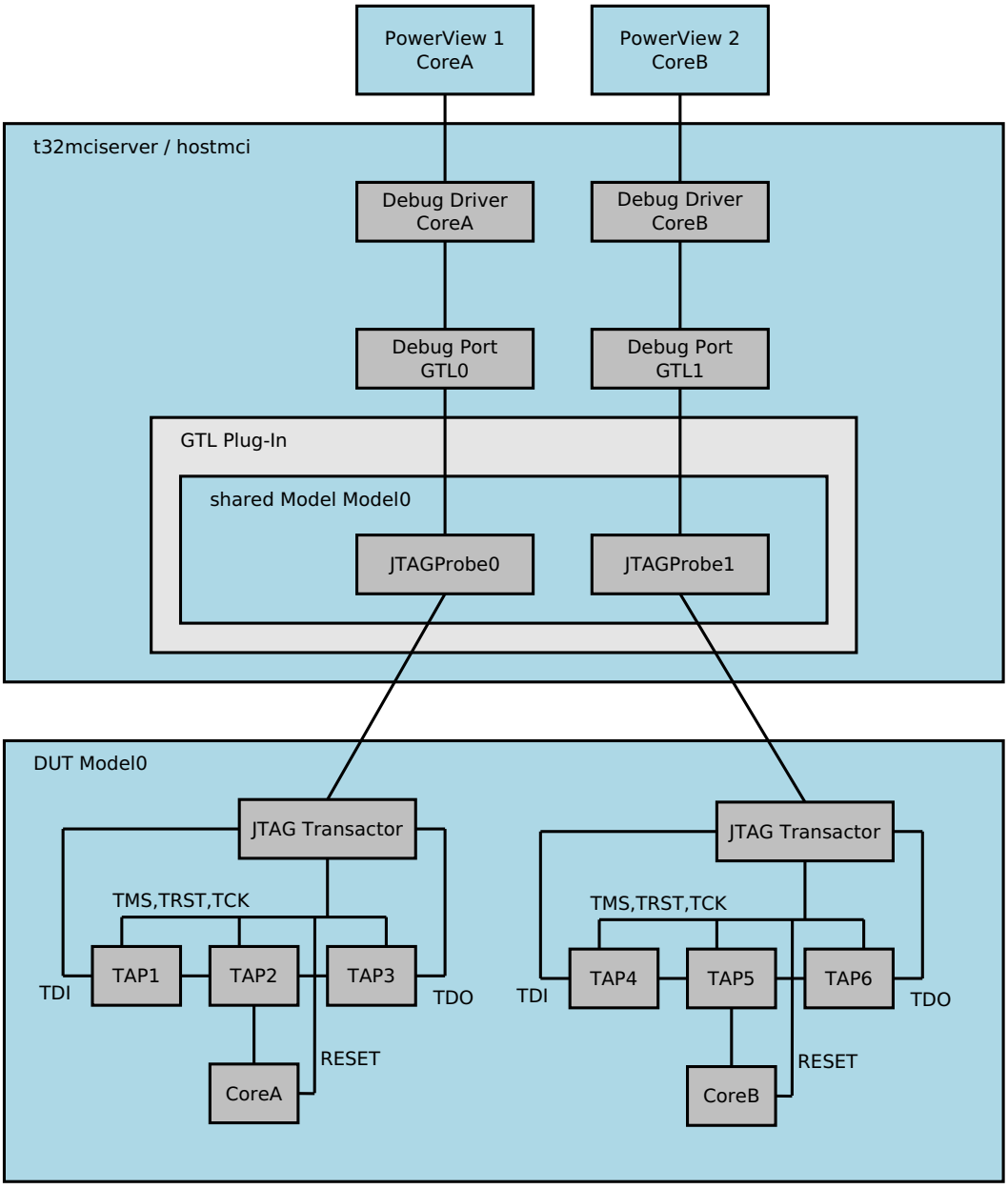
SYStem.GTL.GPIONAME	Configure GPIO transactor to access extra signals as system reset when jtag probe transactor is not used.

GTL Functions

For a description of the GTL functions, see **"SYStem.GTL.CONNECTED() Connection status"** (general_func.pdf).

Shared Models

The DUT can contain multiple debug ports that are independent and provide a different complete feature set as the reset signal or the control of a certain JTAG chain. The way to connect to those asymmetric multi-core systems is to start multiple PowerView instances with individual debug ports that share the same Model of the loaded GTL plug-in. The following picture illustrate the scenario:



Configuration for TRACE32 PowerView 1

```
; select GTL as back-end and a certain port
SYStem.CONFIG.DEBUGPORT GTL0

; (optional) tell the system to connect to a certain DUT
SYStem.GTL.MODELNAME "MODEL_JTAG"

; tell the system the usage of transactors
SYStem.GTL.JTAGPROBENAME "JTAGPROBE0"

; library name of GTL plug-in
SYStem.GTL.LIBname "gtlplugin.so"

; connect to transactors
SYStem.GTL.CONNECT
```

Configuration TRACE32 PowerView 2:

```
; select GTL as back-end and a certain port
SYStem.CONFIG.DEBUGPORT GTL1

; share the plug-in and Model as for debug port GTL0
SYStem.GTL.SHAREDMODEL GTL0

; tell the system the usage of transactors
SYStem.GTL.JTAGPROBENAME "JTAGPROBE1"

; connect to additional transactors
SYStem.GTL.CONNECT
```

Influence of configuration commands

System options can be shared by the whole system, the debug port or are individual for one PowerView instance. The sharing level is classified by the command path:

SYStem.GTL.*	Affects the current selected GTL debug port is shared by all PowerView instances
SYStem.CONFIG.*	Affects only the current PowerView instance
SYStem.VirtualTiming.*	Affects the whole system

Keep the Graphical User Interface Responsive

Due to slow RTL simulation, small operations such as reading the state or showing memory dumps take a long time. This chapter describes how to adjust the virtual time scale to ultra-slow simulators and how to reduce screen flicker caused by slow RTL simulation. To keep the user interface smooth multiple tuning options can be set.

The most important setting is **SETUP.URATE** to configure the update rate of the TRACE32 windows. The processors state is also polled by this rate.

```
SETUP.URATE 10s                    ; screen will be updated every 10s
```

To avoid screen update while PRACTICE scripts are running:

```
SCREEN.OFF                         ; switch off update of the windows when
                                   ; a PRACTICE script is executed

SCREEN                             ; trigger a manual update of the windows
                                   ; inside a PRACTICE script
```

To switch off state polling when the CPU is stopped, the command **SYStem.POLLING** can be used, but the debugger can't detect when another CPU changes the state from stopped to running e.g. by soft reset.

```
SYStem.POLLING DEF OFF             ; disable processor state polling when
                                   ; stopped
```

The command **MAP.UpdateOnce** can be used to read memory regions only one time after a break is detected.

```
MAP.UpdateOnce 0x0++0x1000         ; read memory of regions 0x0--0x1000
                                   ; only one time after break
```

For analysis and data display purposes it is recommended that you use the code from the TRACE32 virtual memory (VM:) instead of the code from the target memory. Therefore, the code needs to be copied to the virtual memory when an *.elf file is being loaded.

```
Data.Load.ELF *.elf /VM            ; download code to target and copy it to
                                   ; VM:
Data.List VM:                      ; open source window, but use VM: memory

Onchip.Access VM                   ; use VM memory for trace analysis
```

Timing Adaption

TRACE32 software includes of a set of efficient low-level driver routines to access the target. These routines have a certain timing that must be adjusted to ultra-slow simulators that can be million times slower than real silicon. In general, there are code parts that pause the execution, wait until a time-out is reached or just use a certain point of time.

For example, when the simulation is 1,000,000 times slower than real time, these commands can be used to adjust the timing in most cases:

```
; configure usage of model time base instead host base to avoid timeouts
; while the emulation is paused.
SYStem.VirtualTiming.TimeinTargetTime ON
SYStem.VirtualTiming.PauseinTargetTime ON

;make the pauses and timeouts 100 times shorter
SYStem.VirtualTiming.TimeScale 0.01

;this will limit any pause statements to 10us target time
SYStem.VirtualTiming.MaxPause 10us

;this will limit any small time-out to read register to 1ms
SYStem.VirtualTiming.MaxTimeout 1ms
```

The following timing **SYStem** commands are available:

SYStem.VirtualTiming.MaxPause	Limit pause
SYStem.VirtualTiming.MaxTimeout	Override time-outs
SYStem.VirtualTiming.PauseinTargetTime	Set up pause time-base
SYStem.VirtualTiming.PauseScale	Multiply pause with a factor
SYStem.VirtualTiming.TimeinTargetTime	Set up general time-base
SYStem.VirtualTiming.TimeScale	Multiply time-base with a factor
SYStem.VirtualTiming.HardwareTimeout	Can disable hardware timeout
SYStem.VirtualTiming.HardwareTimeoutScale	Multiply hardware timeout
SYStem.VirtualTiming.InternalClock	Base for artificial time calculation
SYStem.VirtualTiming.OperationPause	Insert a pause after each action to slow down timing.

Symptom	Cause	Remedy
HostMCI:GTL:Error : can't load library	different elf classes are used for hostmci.so/.dll and the GTL plug-in	start compatible combinations of hostmci.so and the plug-in e.g. both must be 32bit or 64bit. The elf class of hostmci is the same as the elf-class of the process it loads it (t32m????? or t32mciserver).
Status line shows "power down"	TRACE32 can't connect to the simulator.	Check that the simulation is running when TRACE32 start to connect. View the AREA window for any diagnostic messages.
Error "emulator subcore communication timeout"	Debug Driver algorithm took longer than expected	Increase the value of SYStem.VirtualTiming.TimeScale or SYStem.VirtualTiming.HardwareTimeoutScale.

JTAG specific

After the signals and parameters are connected with the TAP of the DUT, PowerView JTAG diagnostic should run:

```
;show results and errors
AREA.view

;set up JTAG clock (simulation clock based)
SYStem.JtagClock 1Mhz

;analyze JTAG chain for testing purposes
SYStem.DETECT DAISYCHAIN
```

Symptom	Cause	Remedy
When the IR and DR length are both "0"	Probably TDI is connected to TDO without a DUT JTAG TAP between them.	connect TDI and TDO with the JTAG chain of the DUT.

Symptom	Cause	Remedy
TDO stays constantly high or low	TDO signal is not connected or the DUT TAP does not work, e.g. is held in reset.	connect TDO correctly, check the signals around the JTAG chain in the simulation/emulation and find out why TDO don't toggle.
JTAG Chain lengths cannot be determined	JTAG frequency might be too high.	Use **SYStem.JtagClock** to lower the JTAG frequency.

NOTE:	The maximum clock of the TAP can be determined by the command **SYStem.DETECT JtagClock**, but the final frequency that can be used also depends to model behind the TAP. The detected frequency is just the upper limit. The optimal frequency depends to the state of the simulation and can change during one debug session.

SYStem.GTL — Configure GTL debug port

Using the **SYStem.GTL** command group, you can configure a GTL debug port (GTL, Generic Transactor Library). The command group is active after GTL has been selected as debug port. It allows to define and configure the used transactors and GTL 3rd-party library. The settings are shared among the TRACE32 instances connected to a certain MCI Server.

```
;optional step: open the SYStem.CONFIG dialog showing the DebugPort tab
SYStem.CONFIG.state /DebugPort

;selecting the GTL back-end activates the SYStem.GTL commands
SYStem.CONFIG.DEBUGPORT GTL0
```

See also

- SYStem.GTL.ARMDAPNAME
- SYStem.GTL.DISCONNECT
- SYStem.GTL.EXPLore
- SYStem.GTL.JTAGPROBENAME
- SYStem.GTL.MODELCOMMAND
- SYStem.GTL.MODELNAME
- SYStem.GTL.RESet
- SYStem.GTL.SERVERCONFIG
- SYStem.GTL.SWDNAME
- SYStem.GTL.TransactorConfig
- SYStem.GTL.CALLCOUNTER()
- SYStem.GTL.CYCLECOUNTER()
- SYStem.GTL.PLUGINVERSION()
- SYStem.GTL.VERSION()
- SYStem.GTL.CONNECT
- SYStem.GTL.DMANAME
- SYStem.GTL.GPIONAME
- SYStem.GTL.LIBname
- SYStem.GTL.MODELCONFIG
- SYStem.GTL.PREBUNDLE
- SYStem.GTL.RESetRESistant
- SYStem.GTL.SHAREDMODEL
- SYStem.GTL.TRACENAME
- SYStem.state
- SYStem.GTL.CONNECTED()
- SYStem.GTL.LIBname()
- SYStem.GTL.VENDORID()

▲ 'Introduction' in 'GTL Debug Back-End'

SYStem.GTL.ARMDAPNAME — Configure name of DAP level transactor

Format: **SYStem.GTL.ARMDAPNAME** *<name>*

By using **SYStem.GTL.ARMDAPNAME** the name for a DAP level transactor can be configured. This transactor is active in all connected TRACE32 instances.

See also

- SYStem.GTL

Format: **SYStem.GTL.CONNECT** [**/TRY**]

Uses the settings previously configured with the **SYStem.GTL** commands to load the GTL library and connect to the emulation or simulation.

TRY Forces the command to continue quietly when the connection could not be established.

Example:

```
;selecting the GTL back-end activates the SYStem.GTL commands
SYStem.CONFIG.DEBUGPORT GTL0

;configure GTL
SYStem.GTL.JTAGPROBENAME "PROBE1"
SYStem.GTL.LIBname "gtllib.so"

;connect to the emulation or simulation
SYStem.GTL.CONNECT
```

See also

- SYStem.GTL

SYStem.GTL.DISCONNECT Disconnect from emulation or simulation

Format: **SYStem.GTL.DISCONNECT** ["*<transactor_name>*"] [**/UNUSED**]

Disconnects from existing connection to the emulation or simulation and disables the periodic re-connection tries.

<transactor_name> Disconnects a named transactor when it is not used anymore.

UNUSED Disconnects from all transactors that are not used anymore.

See also

- SYStem.GTL

SYStem.GTL.DMANAME Name of DMA transactor

Format: **SYStem.GTL.DMANAME "***<transactor_name>***"**

Configures name and usage of DMA transactor to have back-door memory access to the emulation or simulation. The back-door access can be used by **Data.LOAD** command with the parameter **/DMALOAD**.

See also

- SYStem.GTL

SYStem.GTL.EXPLore Display plug-in capabilities

Format: **SYStem.GTL.EXPLore** [*<column>*]

<column>: **DEFault** | **Structure** | **Connected** | **tYpe** | **UsedByCommand** | **CoNFig**

The dialog can show the available transactor interface instances of the plug-in, provided the optional enumeration interface functions have been implemented by the plug-in.

DEFault	Displays a pre-defined set of columns.
Structure	Contains a tree with the abstractions layers of the GTL API. The top level enumerates all instances of the models or scenarios. The available transactor interface instances are displayed below the model.
Connected	Displays whether TRACE32 has an active connection to a model or transactor instance. Mainly the commands **SYStem.GTL.CONNECT** and **SYStem.GTL.DISCONNECT** are used to change the connection state.
tYpe	Type of the node, e.g. model or certain transactor type.
UsedByCommand	Displays a list of configuration commands that are active and point to the transactor instance.
CoNFig	Displays the configuration string of the corresponding **SYStem.GTL.TransactorConfig** command.

Example:

```
SYStem.GTL.EXPLore DEFault
```

See also

- SYStem.GTL

SYStem.GTL.GPIONAME Name of GPIO transactor

Format: **SYStem.GTL.GPIONAME** "*<transactor_name>*"

Configures name and usage of a GPIO transactor. A GPIO transactor can provide a set of signals to access the DUT, e.g. the Reset signal or the JTAG pins. A GPIO transactor can be used in case no JTAG probe transactor is available or when it doesn't implement those signals.

See also

- SYStem.GTL

SYStem.GTL.JTAGPROBENAME Name of JTAG probe transactor

Format: **SYStem.GTL.JTAGPROBENAME** "*<transactor_name>*"

Configures name and usage of a JTAG probe transactor. A JTAG probe transactor can interact with a whole JTAG chain of the DUT.

See also

- SYStem.GTL

SYStem.GTL.LIBname Name of 3rd-party plug-in library

Format: **SYStem.GTL.LIBname** "*<transactor_name>*"

Configures the 3rd-party GTL library that is used to access the emulation or simulation. This command should be issued as the last configuration command.

See also

- SYStem.GTL

Format: **SYStem.GTL.MODELCOMMAND** "*<command>*"

Executes a plug-in specific command.

Example:

```
SYStem.GTL.MODELCOMMAND "do something important"
LOCAL &result
&result=EVAL.STRing()
PRINT "Result was: &result"
```

See also

- SYStem.GTL

SYStem.GTL.MODELCONFIG Configure emulation options

Format: **SYStem.GTL.MODELCONFIG** "*<configuration>*"

Configures the options to connect to the emulation or simulator. The particular options are defined by the 3rd-party plug-in.

See also

- SYStem.GTL

SYStem.GTL.MODELNAME Select emulation

Format: **SYStem.GTL.MODELNAME** "*<model_name>*"

Selects a certain emulation out of a set of emulations.

See also

- SYStem.GTL

Format:	**SYStem.GTL.PREBUNDLE** [*<option>*]
<option>:	**AUTO** | **ON** | **OFF**

Default: AUTO.

The option controls whether TRACE32 shall collect write accesses and perform them later on, or perform them immediately. Collecting write accesses increases the performance but may cause problems with the original error handling or introduce new effects in plug-in implementations.

(no parameter)	Displays the current setting in the TRACE32 message line.
AUTO	The setting depends on the plug-in and transactor interface.
ON	Pre-bundling is active for all transactor interfaces.
OFF	Pre-bundling is not active for all transactor interfaces.

See also

- SYStem.GTL

SYStem.GTL.RESet Reset GTL settings

Format:	**SYStem.GTL.RESet**

Resets the connection to the transactor plug-in and the GTL configuration.

This command should only be used on the TRACE32 command line.

See also

- SYStem.GTL.RESetRESistant
- SYStem.GTL

Format: **SYStem.GTL.RESetRESistant** [**ON** | **OFF**]

Controls the effect that the two reset commands **RESet** and **SYStem.RESet** have on the GTL settings.

ON The two reset commands have no effect on the configuration and the connection to the transactor plug-in.

OFF The configuration and the connection to the transactor plug-in can be reset by the two reset commands.

Example:

```
;selecting the GTL back-end activates the SYStem.GTL commands
SYStem.CONFIG.DEBUGPORT GTL0

;exempt the GTL settings from the two reset commands
SYStem.GTL.RESetRESistant ON
;...
```

See also

- SYStem.GTL.RESet
- SYStem.GTL

SYStem.GTL.SERVERCONFIG Configure server options

Format: **SYStem.GTL.SERVERCONFIG** "*<configuration>*"

Configures options to connect to the server knowing all emulations. The particular options are defined by the 3rd-party plug-in.

See also

- SYStem.GTL

Format: **SYStem.GTL.SHAREDMODEL** *<gtl_debug_port>*

Links two GTL debug ports in order to share a connection to the DUT across multiple debug ports. More information about the scenario can be found in the **backend manual**.

<gtl_debug_port> Can be **GTL0**...**GTL***<n>*

See also

- SYStem.GTL

SYStem.GTL.SWDNAME Communicate with target via SWD

Format: **SYStem.GTL.SWDNAME** "*<name>*"

Configures the transactor *<name>* that is used to perform raw SWD communication with the target (SWD = (serial wire debug).

Usually the name is the same as configured by **SYStem.GTL.JTAGPROBENAME** because the raw SWD communication is an extension of the JTAG transactor interface and one single transactor instance is used. When **SYStem.GTL.JTAGPROBENAME** and **SYStem.GTL.SWDNAME** have been configured, then the command **SYStem.CONFIG.DEBUGPORTTYPE** can switch between JTAG and SWD.

Example:

```
; configure JTAG/SWD mixed mode
SYStem.GTL.JTAGPROBENAME "JTAGSWDXTOR"
SYStem.GTL.SWDNAME "JTAGSWDXTOR"

; switch to SWD
SYStem.CONFIG.DEBUGPORTTYPE SWD

; connect to the CPU using SWD
SYStem.Up
```

See also

- SYStem.GTL

Format: **SYStem.GTL.TRACENAME** "*<transactor_name>*"

Configures name and usage of a Trace transactor. A Trace transactor can record off-chip trace data.

Example:

```
;select name for Trace transactor
SYStem.GTL.TRACENAME "TRACE0"

;connect to emulation or simulation
SYStem.GTL.CONNECT

;select trace method, initialize the trace and show control the window
Trace.METHOD Analyzer
Analyzer.Init
Analyzer.state
```

See also

- SYStem.GTL

SYStem.GTL.TransactorConfig Preconfigure a certain transactor

Format: **SYStem.GTL.TransactorConfig** "*<transactor_name>*" "*<configuration>*"

Sets up a configuration string that is passed to the GTL plug-in when the transactor is connected. When the configuration string for a certain transactor changes the transactor need to be disconnected. It is recommended to pass the configuration before the transactors are defined, because this avoids unnecessary reconnections.

<transactor_name> Name of the transactor that shall be configured.

<configuration> Specific configuration string passed to the GTL plug-in.

See also

- SYStem.GTL

Example:

```
;pass TARGETSEL option to SWD transactor
SYStem.GTL.TransactorConfig "SWD_DAP1" "TARGETSEL=1"

;use DAP level transactor by debugger
SYStem.Config.DAPName "SWD_DAP1"

;connect to emulation or simulation
SYStem.GTL.CONNECT
```