Verilog Debug Back-End

Verilog Debug Back-End

Version 12-Feb-2014

01-Nov-12 New manual.

Introduction

The Verilog Transactor is used to interact with a software Verilog RTL Simulator. Software RTL simulator models can be extremely slow. Therefore the focus of using the transactor is to test the design together with TRACE32 rather than to debug an application. TRACE32 is no ASIC verification tool, but it provides PRACTICE as scripting language to automate tests and access the model by its debug modules.

It is not intended to use the Verilog Transactor with emulators because the transactor is not accelerated and would slow down the emulation.

TRACE32 PowerView provides special commands to allow a minimum of sequences to be send to the simulation to test a certain feature.

Related Documents

For Windows only - The T32Start application assists you in setting up multicore/multiprocessor debug environments, and software-only debug environments.

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

Contacting Support

LAUTERBACH GmbH
Altlaufstrasse 40
85635 Hoehenkirchen-Siegertsbrunn
Germany

Phone (+49) 8102-9876-555

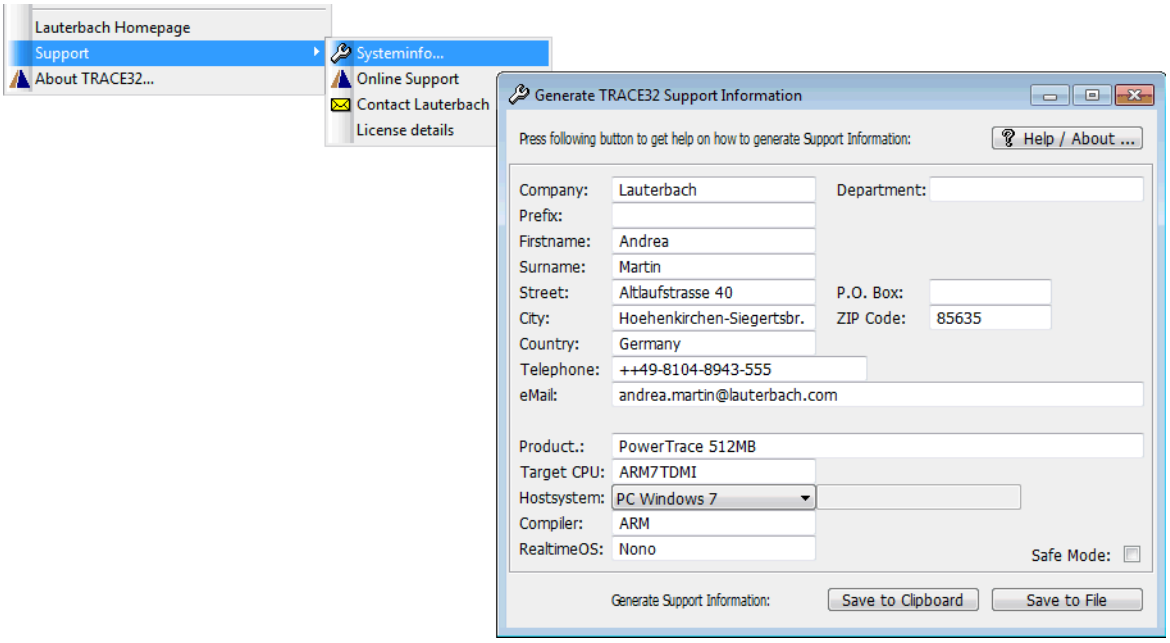
Fax (+49) 8102-9876-187

Internet http://www.lauterbach.com/tsupport.html or http://www.lauterbach.com/report.html
Here you'll find local and special support addresses.

E-mail support@lauterbach.com
General support address where your request will be answered within a short time if it is a basic support request or redirected to the appropriate address.

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
DUT	Device Under Test. A DUT is the part of the model that is being tested.
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.
RTL simulator	A software RTL simulator executes a model on RTL level without using special acceleration hardware.
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.
Verilog	Hardware description language on RTL level.
VPI	Verilog Procedural Interface. The Verilog Procedural Interface is used to interface from Verilog models into behavioral system parts written in the programming language “C”.

Supported Transactors and Simulators

Supported transactors are:

- JTAG Transactor

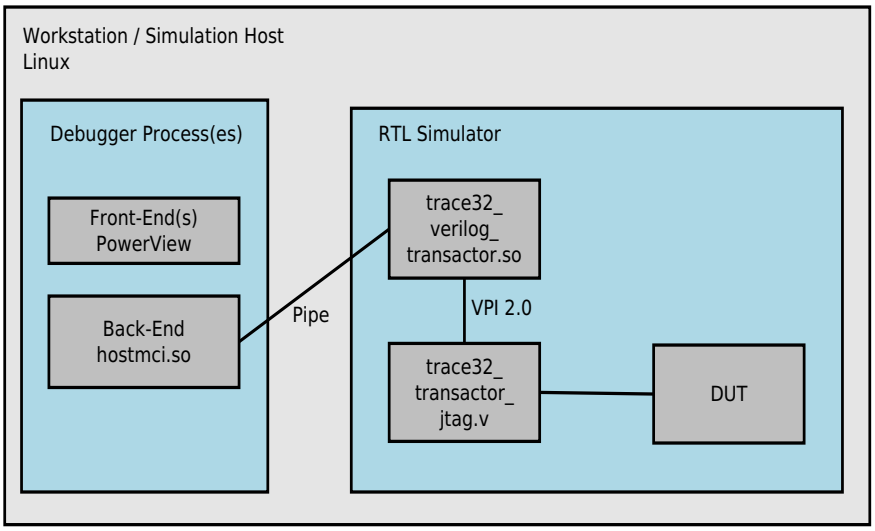
All simulators are supported that interface by VPI 2.0 (Verilog Procedural Interface) such as:

- Cadence NC-Verilog
- Synopsys VCS
- GPLCVER
- and many others

JTAG Transactor

The JTAG Transactor provides the following features:

- Access to JTAG signals using **JTAG** commands and API functions described in **"API for Remote Control and JTAG Access"** (api_remote.pdf), chapter **"ICD TAP Access API Functions"**.
- JTAG shift engine including ARM RTCK
- Runtime Counter by Run-Line or Stopped-Trigger-Line
- Virtual PodBus Trigger
- Reset signal trigger
- Artificial high JTAG frequency to encounter JTAG protocol overhead



In all cases, the shared library file `trace32_verilog_transactor.so` must be started together with the RTL simulator. The RTL simulator calls the library through the VPI 2.0 interface in the module `trace32_transactor_jtag.v` that exports the JTAG signals.

The transactor library communicates through named pipes with another shared library `hostmci.so`. This library contains the low-level algorithms that do high-performance accesses, requiring low latency.

PowerView System Configurations

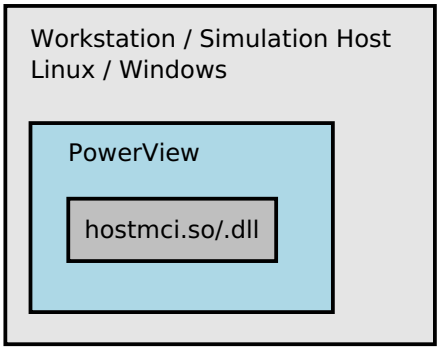
The TRACE32 PowerView instances can be set up in different ways.

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 library `hostmci.so` (back-end) can either run in a dedicated communication process `t32mciserver` or can be integrated into the TRACE32 PowerView process. The dedicated communication process is used when TRACE32 PowerView runs on another hosts in the network or it can be used to run multiple TRACE32 PowerView instances on the back-end host, too.

Setup 1

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

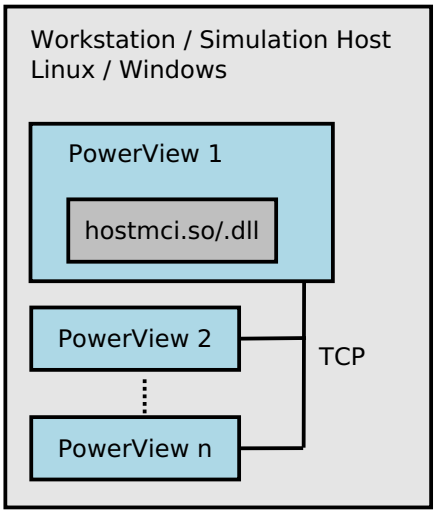


Modify the config.t32 file as follows:

```
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:

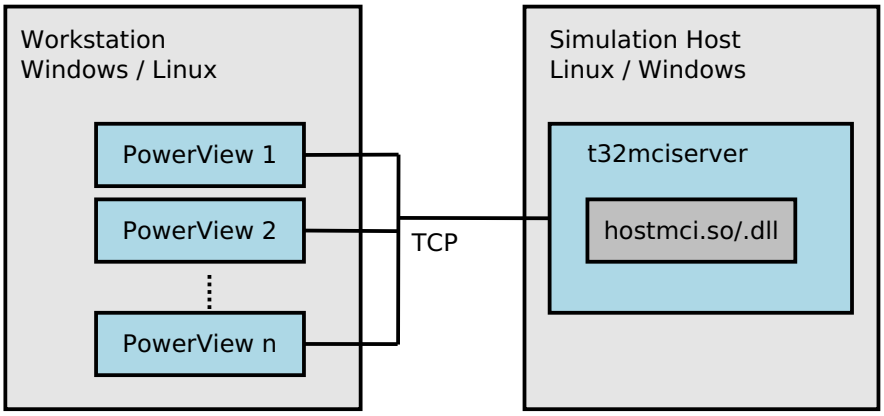


Modify the config.t32 as follows:

```
PBI=MCISERVER                   ; set up the usage of hostmci.so and open
PORT=30000                      ; server at 30000 for the first instance.
INSTANCE=1                      ; consecutive number of instance
```

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=1                         ; consecutive number of instance
DEDICATED                          ; avoid to fall into Setup2 case
```

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

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

In a multi-user, multi-simulation environment, the pipe name needs to be unique. Both shared libraries use a default file name that is derived from the `USER` environment variable:

```
/tmp/t32verilog_transactor_$USER
```

This pipe name can be redefined by specifying the environment variable `T32VERILOGTRANSACTORPIPE`. For `hostmci.so` the default pipe name can also be set up with the command **SYStem.CONFIG.TRANSACTORPIPENAME**.

RTL Simulator Integration

This section describes how to integrate the Verilog Transactor into the simulation. In a first step, the DUTs signals must be connected to the Verilog part of the transactor. In a second step, transactor library must be loaded together with the simulator.

Step 1: Connecting Signals

The JTAG transactor is implemented in the module `trace32_transactor_jtag.v`. The module `trace32_transactor_jtag_debugport_v1` is used to interface with TRACE32 by connecting it to the JTAG TAP Controller of the design. The module interfaces through multiple signals and parameters.

In most cases, the smaller interface of the module `trace32_transactor_jtag` can be used to connect only mandatory JTAG signals to the TAP of the DUT.

The following table describes the signals of `trace32_transactor_jtag_debugport_v1` and `trace32_transactor_jtag`:

Mandatory Signal	Direction	Description
tck_o	Out	TCK signal to DUT
tms_o	Out	TMS signal to DUT
tdi_o	Out	TDI signal to DUT
tdo_i	In	TDO signal from DUT
ntrst_o	Out	NTRST signal to DUT (low active)

The following table describes the additional signals of `trace32_transactor_jtag_debugport_v1:`

Additional Signal	Direction	Description
nreset_o	Out	NRESET signal to DUT (low active). The debugger uses this signal to reset the DUT.
nreset_i	In	NRESET signal from DUT (low active). The signal is used to detect a reset of the DUT e.g. when other transactors reset the simulation model.
power_i	In	POWER signal from DUT. When the level is 0, the TRACE32 will show Power Down.

Additional Signal	Direction	Description
trigger_i	In	A trigger signal from DUT. The trigger signal appears as PodBus trigger in TRACE32. By the **TrBus.state** dialog different actions can be programmed when a trigger appears.
trigger_o	Out	A trigger signal to the DUT. The trigger signal appears as PodBus trigger in TRACE32.
runline_i	In	A line to allow exact runtime measurement of code. When the line is “1” the runtime counter counts in simulation time. The time is displayed in the **RunTime.state** dialog.
rtck_i	In	RTCK signal from the DUT (used by some ARM architecture JTAG TAPs in order to signal that TDO is ready to sample.).

The following parameters are used to configure the input signals. A value of “0” means that a signal is ignored by the transactor. The input signal will be used when the value is “1”.

Parameter	Default	Description
instance	-1	Used to manage multiple instances in future versions. **Overwrite by “0” always!**
poll_frequency_hz	1000	Internal poll rate in Hz to trigger communication from PowerView. A high poll rate reduces the latency to hostmci.so, but consumes more simulation time.
has_reset_i	1	Specify if nreset_i signal shall be used.
has_power_i	0	Specify if power_i signal shall be used.
has_trigger_i	0	Specify if trigger_i signal shall be used.
has_runline_i	0	Specify if runline_i signal shall be used.
has_rtck_i	0	Specify if rtck_i signal shall be used.

Step 2: Loading the trace32_verilog_transactor.so

GPLCVER-2.12a Simulator

On the Linux command line, run `cver` with the transactor and use `trace32_jtag_tb.v` as top level module:

```
/usr/bin/cver \
+loadvpi=trace32_verilog_transactor.so:vpi_compat_boootstrap \
trace32_jtag_tb.v
```

Connecting TRACE32 to the Verilog Transactor

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

```
RESet

;set up connection to Simulator and try to connect
SYStem.CONFIG.DebugPort VerilogTransactor0

;used to configure pipe name in case former transactor is used
;SYStem.CONFIG.TRANSACTORPIPENAME "/tmp/t32verilog_actuator_user"

;set up the JTAG clock (simulation clock based)
;find out the maximum JTAG frequency to speedup all operations!
SYStem.JtagClock 1Mhz

;modify timing constraints
;will make the host time 1000.0 times slower
SYStem.VirtualTiming.TimeScale 1000.0
;this will limit any pause statements to 1000ms host time
SYStem.VirtualTiming.MaxPause 1000ms
;this will limit any small timeout to read register to 5min
SYStem.VirtualTiming.MaxTimeout 300s

;select the CPU
SYStem.CPU CortexA9
;tell the system that a DAP is present
SYStem.CONFIG COREBASE APB:0x80009000

;connect to JTAG quickly
SYStem.Mode.Prepare

;access to busses now working
Data.Set DAP:0x00000000 %long 0x0 /Verify
```

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
```

To switch off polling when the CPU is running at all, the command **SYStem.POLLING** can be used, but the debugger can't detect running-stopped transitions any more, unless break is executed.

```
SYStem.POLLING OFF                 ; disable processor state polling
Go
WAIT 10s
Break                              ; this will halt the target and poll the
                                   ; processors state gain
```

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 /CopyVM        ; 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.VirtualTime.TimeinTargetTime ON
SYStem.VirtualTime.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.

Troubleshooting the JTAG Transactor

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 connection to Simulator
SYStem.CONFIG.DebugPort VerilogTransactor0

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

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

Symptom	Cause	Remedy
Status line shows “power down”	TRACE32 can’t connect to the simulator.	Check that the simulation is running when TRACE32 start to connect. Check that hostmci.so is started in the same user context or modify the pipe names by **SYStem.CONFIG.TRANSACTORPIPENAME**.
When the IR and DR length are both “0”	Probably TDI is connected to TDO without a DUT JTAG TAP between them.	See **Step 1: Connecting Signals**.
TDO stays constantly high or low	TDO signal is not connected or the DUT TAP does not work, e.g. is held in reset.	See **Step 1: Connecting Signals**.
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.

Product Information

Order No.	Code	Text
LA-8950D	BE-V-CORTEX-A/R-UD	Verilog Debug Back-End Cortex-A/R USB Dongle
LA-8950L	BE-V-CORTEX-A/R-FL	1 User Floating Lic Verilog Back-End Cortex-A

Order Information

Back-End

OrderNo Code	Text
LA-8950D BE-V-CORTEX-A/R-UD	**Verilog Debug Back-End Cortex-A/R USB Dongle** TRACE32 Verilog Back-End for Cortex-A/R support for JTAG access licensed for one host system via USB dongle for Windows32, Windows64, Linux32, Linux64 and MAC OS X
LA-8950L BE-V-CORTEX-A/R-FL	**1 User Floating Lic Verilog Back-End Cortex-A** TRACE32 Verilog Back-End for Cortex-A/R support for JTAG access for Windows32, Windows64, Linux32, Linux64 and Solaris, other platforms on request floating license via RLM (Reprise License Manager) Please add the RLM HostID of the license server to your order (please see our FAQ)

Order No.	Code	Text
LA-8890D	FRONTEND-ARM-UD	TRACE32 Front-End ARM (Dongle)
LA-8890L	FRONTEND-ARM-FL	1 User Floating License ARM Front-End
LA-8892D	FRONTEND-PPC-UD	TRACE32 Front-End PPC (Dongle)
LA-8892L	FRONTEND-PPC-FL	1 User Floating License PPC Front-End
LA-8898D	FRONTEND-MIPS32-UD	TRACE32 Front-End MIPS32 (Dongle)
LA-8898L	FRONTEND-MIPS32-FL	1 User Floating License MIPS32 Front-End
LA-8903D	FRONTEND-ARC-UD	TRACE32 Front-End ARC (Dongle)
LA-8903L	FRONTEND-ARC-FL	1 User Floating License ARC Front-End
LA-8895D	FRONTEND-SH-UD	TRACE32 Front-End SH (Dongle)
LA-8895L	FRONTEND-SH-FL	1 User Floating License SH Front-End
LA-8896D	FRONTEND-STARCORE-UD	TRACE32 Front-End StarCore (Dongle)
LA-8896L	FRONTEND-STARCORE-FL	1 User Floating License StarCore Front-End
LA-8894D	FRONTEND-TEAK-UD	TRACE32 Front-End TeakLite (Dongle)
LA-8894L	FRONTEND-TEAK-FL	1 User Floating License TeakLite Front-End
LA-8891D	FRONTEND-C5000-UD	TRACE32 Front-End TMS320C5X (Dongle)
LA-8891L	FRONTEND-C5000-FL	1 User Floating License TMS320C5X Front-End
LA-8893D	FRONTEND-TC-UD	TRACE32 Front-End TriCore (Dongle)
LA-8893L	FRONTEND-TC-FL	1 User Floating License TriCore Front-End
LA-8889D	FRONTEND-V850-UD	TRACE32 Front-End V850 (Dongle)
LA-8889L	FRONTEND-V850-FL	1 User Floating License V850 Front-End
LA-8899D	FRONTEND-X86-UD	TRACE32 Front-End x86 (Dongle)
LA-8899L	FRONTEND-X86-FL	1 User Floating License x86 Front-End
LA-8902L	FRONTEND-MULTICORE-F	1 User Floating License Multicore
LA-8025	SOFT-MAIN-DONGLE-ST	TRACE32-ICD Maintenance Contract for Dongle
LA-8027L	SOFT-WARR-FL	1 User Front-End Warranty - Floating Lic.
LA-8034L	SOFT-UPDATE-FL	1 User Front-End Software Update + Maintenanc