TRACE32 Debug Back-Ends

TRACE32 Debug Back-Ends

TRACE32 Debug Back-Ends

Version 06-Jun-2024

History

22-Aug-23	Initial version of the manual.

Introduction

TRACE32 supports various debug back-ends to enable a low-level debug communication for debugging and tracing:

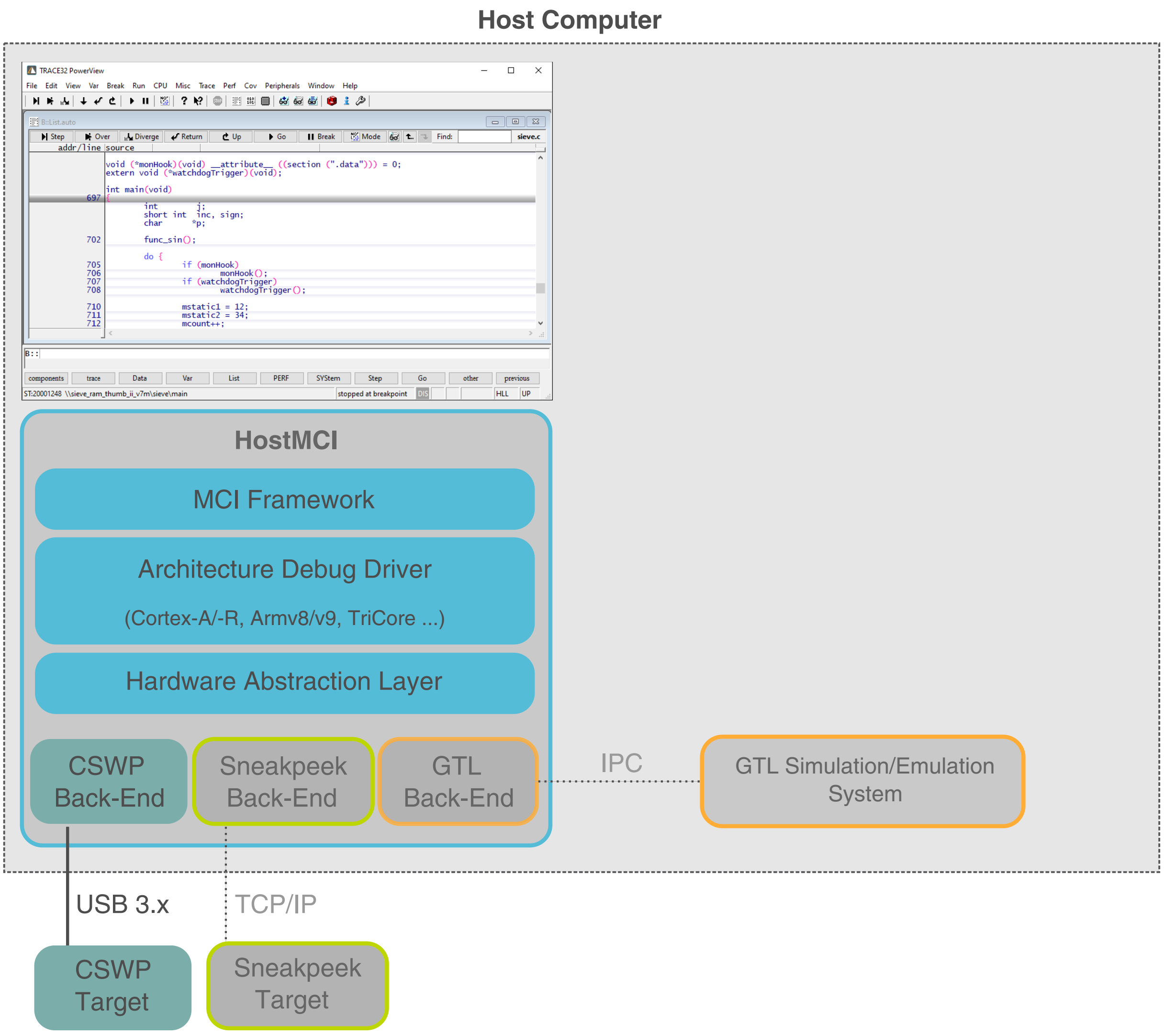
- **A target over a functional interface of the host computer such as USB or ethernet.**

  Low-level communication must be wrapped in a specified protocol.

- **An RTL simulation or emulation system.**

  Inter-process communication is used here.

The low level communication is handled by the TRACE32 HostMCI library. While the back-ends are responsible for the communication with the target/emulation system.The following block diagram gives an overview of the TRACE32 system architecture when debug back-ends are deployed.



Host Computer
TRACE32 PowerView
File Edit View Var Break Run CPU Misc Trace Perf Cov Peripherals Window Help
B::List.auto
Step Over Diverge Return Up Go Break Mode Find: sieve.c
addr/line source
B::
components trace Data Var List PERF SYStem Step Go other previous
HostMCI
MCI Framework
Architecture Debug Driver
(Cortex-A/-R, Armv8/v9, TriCore ...)
Hardware Abstraction Layer
CSWP Back-End
Sneakpeek Back-End
GTL Back-End
IPC
GTL Simulation/Emulation System
USB 3.x
TCP/IP
CSWP Target
Sneakpeek Target

```
void (*monHook)(void) __attribute__ ((section (".data"))) = 0;
extern void (*watchdogTrigger)(void);

int main(void)
{
        int        j;
        short int  inc, sign;
        char      *p;

        func_sin();

        do {
                if (monHook)
                        monHook();
                if (watchdogTrigger)
                        watchdogTrigger();

                mstatic1 = 12;
                mstatic2 = 34;
                mcount++;
```

```
ST:20001248 \\sieve_ram_thumb_ii_v7m\sieve\main    stopped at breakpoint  DIS  HLL  UP
```

Debug communication is established in two steps.

1. Start a single or multiple TRACE32 instances (see chapter **“PowerView System Configurations”**, page 6) that use the TRACE32 HostMCI library.

2. Select the appropriate debug back-end (see chapter **“Supported Debug Back-Ends”**, page 9).

PowerView System Configurations

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

1. Setup 1: A single TRACE32 PowerView instance runs on the same host as the back-end.

   The SneakPeek Debug Back-End supports setup 1 only. If you need setup 2 or 3, please contact us via https://support.lauterbach.com/

2. Setup 2: Multiple TRACE32 PowerView instances run on the same host as the back-end.

3. Setup 3: The TRACE32 PowerView instances run on a dedicated workstation; the back-end runs on another host.

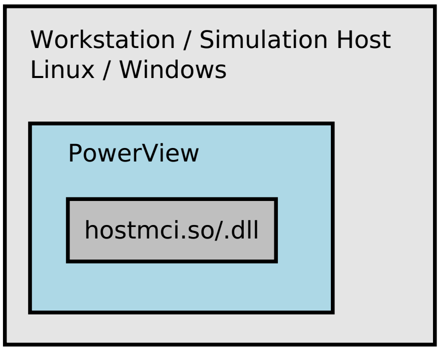
The Lauterbach Debug Driver library (`hostmci.so` for Linux/Mac users and `hostmci.dll` for Windows users) can be integrated into the TRACE32 PowerView application or run as a separate process, called `t32mciserver`. Running it as a separate process provides two main benefits:

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:

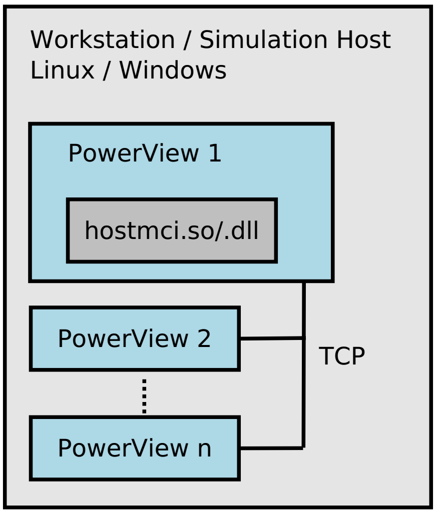


Modify the config.t32 file as follows:

```
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:

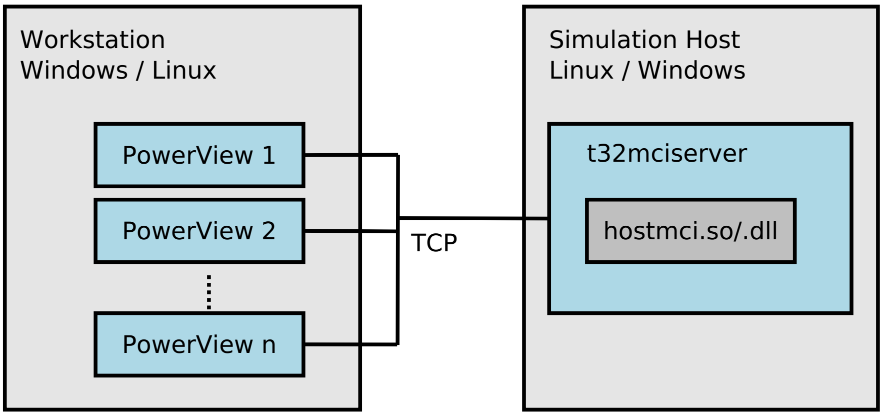


Modify the config.t32 as follows:

```
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
```

Supported Debug Back-Ends

CSWP Debug Back-End

The CSWP Debug Back-End provides debug and trace capabilities for targets that support the Arm CoreSight Wire Protocol.

Use this back-end to debug and trace CoreSight SoC-600 based SoCs via functional interfaces such as USB.

```
SYStem.CONFIG.DEBUGPORT CSWP0

…
```

The CSWP Debug Back-End manual is still under construction.

GTL Debug Back-End

The GTL Debug Back-End provides debug and trace capabilities through the Generic Transactor Library (GTL), an API designed by Lauterbach.

Use this back-end to connect PowerView to emulation systems or RTL simulators, via custom GTL transactors.

```
SYStem.CONFIG.DEBUGPORT GTL0

…
```

A detailed description of the setup for the GTL debug back-end can be found in **“GTL Debug Back-End”** (backend_gtl.pdf).

Sneakpeek Debug Back-End

The SneakPeek Debug Back-End provides debug capabilities, using the MIPI SneakPeek Protocol to perform address-mapped read and write transactions.

Use this back-end to connect PowerView to SneakPeek-capable target systems over TCP/IP.

```
SYStem.CONFIG.DEBUGPORT SNEAKPEEK0

…
```

A detailed description of the setup for the Sneakpeek debug back-end can be found in **“Sneakpeek Debug Back-End”** (backend_sneakpeek.pdf).

Verilog Debug Back-End

The Verilog Debug Back-End provides capabilities to debug an SoC running in a software Verilog RTL Simulator via a Verilog Transactor. This transactor is provided for backwards compatibility. Please use the GTL Debug Back-End for new developments.

Use the Verilog debug back-end to run low-level tests of the design along with TRACE32 instead of debugging an application.

```
SYStem.CONFIG.DEBUGPORT VerilogTransactor0

…
```

A detailed description of the setup for the Verilog debug back-end can be found in **“Verilog Debug Back-End”** (backend_verilog.pdf).

XCP Debug Back-End

The XCP Debug Back-End provides capabilities for debugging and using the on-chip trace through third-party XCP slaves, e.g. calibration tools. The XCP communication is done via Ethernet connection, and allows to share debug and trace ports between TRACE32 and the XCP slave.

Use the XCP Debug Back-End to debug target systems connected to an XCP slave, e.g. for in-field testing and/or closed-chassis debugging.

```
SYStem.CONFIG.DEBUGPORT XCP0

…
```

A detailed description of the setup for the XCP debug back-end can be found in **“XCP Debug Back-End”** (backend_xcp.pdf).

Infineon DAS Debug Back-End

The Infineon DAS Debug Back-End provides debug and on-chip trace capabilities via an Infineon DAS Server.

Use the Infineon DAS Debug Back-End to connect to an emulation system connected to a DAS server

```
SYStem.CONFIG.DEBUGPORT InfineonDAS0

…
```

A detailed description of the setup can be found in **“Debugging via Infineon DAS Server”** (backend_das.pdf).

Intel DCI Debug Back-End

The Intel DCI Debug Back-End provides debug and trace capabilities using the Intel DCI protocol.

Use the Intel DCI Debug Back-End to debug and trace for Intel® systems over USB.

```
SYStem.CONFIG.DEBUGPORT IntelUSB0

…
```

A detailed description of the setup can be found in the chapter **“DCI DbC”** in Debugging via Intel® DCI User´s Guide, page 7 (dci_intel_user.pdf).

Tessent Embedded Analytics Debug Back-End

The Tessent Embedded Analytics Debug Back-End provides debug and trace capabilities to Tessent Embedded Analytics enabled SoCs.

Use this back-end to debug and trace SoCs with Tessent Embedded Analytics ecosystem over USB.

The manual is confidential. It is subject to the terms of an agreement between you and Siemens Industry Software Inc. (SISW).