LAUTERBACH
DEVELOPMENT TOOLS

Simulator for XTENSA

MANUAL

Simulator for XTENSA

Version 06-Jun-2024

History

20-Jul-22	For the MMU.SCAN ALL command, CLEAR is now possible as an optional second parameter.

Introduction

This manual serves as a guideline for working with the TRACE32 Instruction Set Simulator for XTENSA.

The Instruction Set Simulator for XTENSA covers the simulation of the basic instruction set of XTENSA cores. However, XTENSA configurations usually contain customized instructions, which are not known to the Instruction Set Simulator for XTENSA and thus can not be simulated.

Use the **SYStem.TIE** command group to extend the Instruction Set Simulator for XTENSA for working with customized cores.

All general commands are described in the **"PowerView Command Reference"** (ide_ref.pdf) and **"General Commands Reference"**.

TRACE32 Simulator License

[build 68859 - DVD 02/2016]

The extensive use of the TRACE32 Instruction Set Simulator requires a *TRACE32 Simulator License*.

For more information, see **www.lauterbach.com/sim_license.html**.

To start the simulator, proceed as follows:

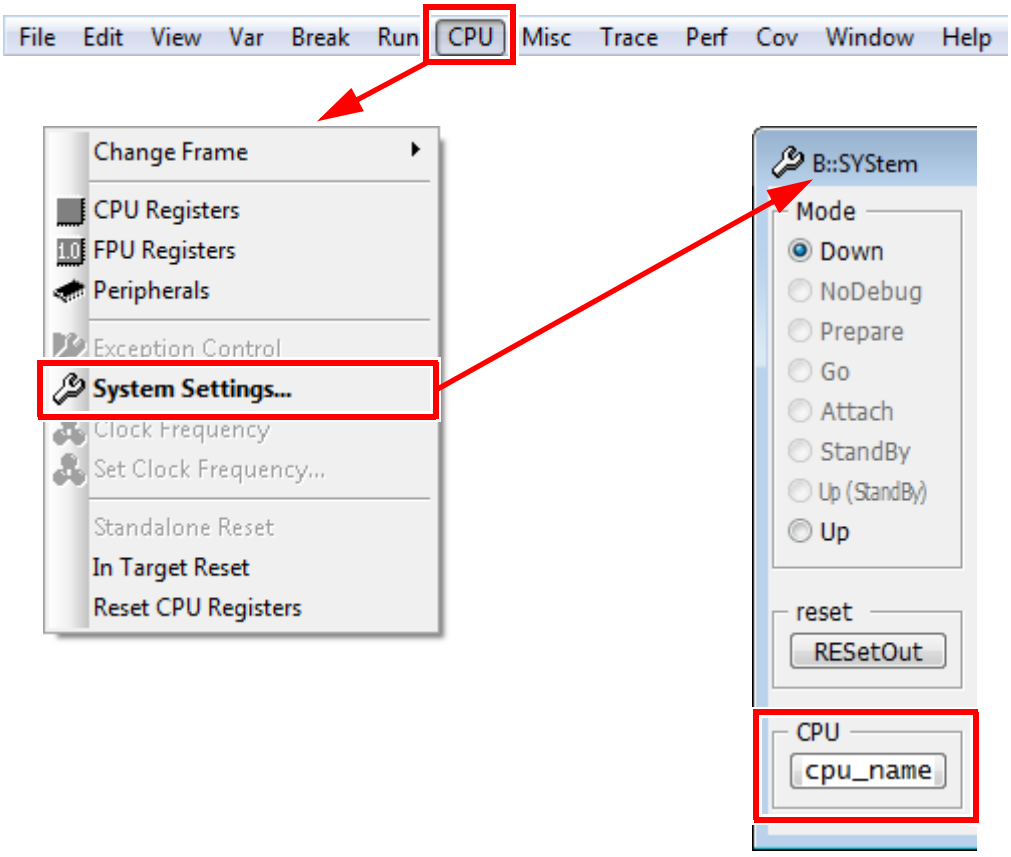
1. Select the device prompt for the Simulator and reset the system.

```
B::

RESet
```

The device prompt `B::` is normally already selected in the TRACE32 command line. If this is not the case, enter `B::` to set the correct device prompt. The **RESet** command is only necessary if you do not start directly after booting TRACE32.

2. Specify the CPU specific settings.




```
SYStem.CPU <cpu_name>
```

The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Enter debug mode.

```
SYStem.Up
```

This command resets the CPU and enters debug mode. After this command is executed it is possible to access memory and registers.

4. Load the program.

```
Data.LOAD.<file_format> <file>      ; load program and symbols
```

See the **Data.LOAD** command reference for a list of supported file formats. If uncertain about the required format, try **Data.LOAD.auto**.

A detailed description of the **Data.LOAD** command and all available options is given in the reference guide.

5. Start-up example

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>*.

```
B::                                 ; Select the ICD device prompt

WinCLEAR                            ; Clear all windows

SYStem.CPU <cpu_name>               ; Select CPU type

SYStem.Up                           ; Reset the target and enter
                                    ; debug mode

Data.LOAD.<file_format> <file>      ; Load the application

Register.Set pc main                ; Set the PC to function main

PER.view                            ; Show clearly arranged
                                    ; peripherals in window      *)

List.Mix                            ; Open source code window    *)

Register.view /SpotLight            ; Open register window       *)

Frame.view /Locals /Caller          ; Open the stack frame with
                                    ; local variables            *)

Var.Watch %Spotlight flags ast      ; Open watch window for
                                    ; variables                  *)
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

Peripheral Simulation

For more information, see **“API for TRACE32 Instruction Set Simulator”** (simulator_api.pdf).

FAQ

Please refer to https://support.lauterbach.com/kb.

SYStem.CONFIG.state — Display target configuration

Format:	**SYStem.CONFIG.state** [**/***<tab>*]
<tab>:	**DebugPort** | **Jtag** | **MultiTap** | **DAP** | **COmponents**

Opens the **SYStem.CONFIG.state** window.

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**XTENSA** | **DC108MINI** | **DC212GP** | **DC232L** | **DC330HIFI** | **DC545CK** | **DC570T** | …

Selects the processor type.

SYStem.LOCK — Tristate the JTAG port

Command has no effect on the TRACE32 Instruction Set Simulator.

SYStem.MemAccess — Select run-time memory access method

Command has no effect on the TRACE32 Instruction Set Simulator.

Format:	**SYStem.Mode** *<mode>* **SYStem.Attach** (alias for SYStem.Mode Attach) **SYStem.Down** (alias for SYStem.Mode Down) **SYStem.Up** (alias for SYStem.Mode Up)
<mode>:	**Down** **Attach** **Up**

Default: Down.

Selects the target operating mode.

Down	The CPU is in reset. Debug mode is not active.
Attach	Same as Up on Simulator.
Up	Resets the target, sets the CPU to debug mode and stops the CPU. After the execution of this command the CPU is stopped and all register are set to the default level.

SYStem.Option.DisMode Define disassembler mode

Format:	**SYStem.Option.DisMode** *<mode>*
<mode>:	**INTernal** **LIBrary** **INTbeforeLIB** **LIBbeforeINT**

Defines the disassembler mode.

INTernal	Use TRACE32 internal disassembler.
LIBrary	Use disassembler from TIE library defined with **SYStem.TIE** commands, see example below.

INTbeforeLIB Prefer TRACE32 internal disassembler, use TIE library as fallback.

LIBbeforeINT Prefer TIE library, use TRACE32 internal disassembler as fallback.

Example:

```
SYStem.TIE.DELete                       ; Delete all already added files

SYStem.TIE.AddCoreLibrary libisa-core-hw.dll  ; Add TIE library files
SYStem.TIE.AddCoreLibrary libisa-core.dll
SYStem.TIE.AddCoreLibrary libisa-DC_330HiFi.dll

SYStem.TIE.ENAble                      ; Load and enable TIE Instructions

SYStem.Option.DisMode LIBbeforeINT   ; set disassembler preference
```

SYStem.Option.Endianness Specify the byte ordering

Format: **SYStem.Option.Endianness** [**AUTO** | **Little** | **Big**]

Default: AUTO.

The instructions for the JTAG connection to the XTENSA core depend on the byte ordering.

SYStem.Option.IMASKASM Disable interrupts while single stepping

Command has no effect on the TRACE32 Instruction Set Simulator.

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

Command has no effect on the TRACE32 Instruction Set Simulator.

Format:	**SYStem.Option.MMUSPACES** [**ON** | **OFF**] **SYStem.Option.MMUspaces** [**ON** | **OFF**] (deprecated) **SYStem.Option.MMU** [**ON** | **OFF**] (deprecated)

Default: OFF.

Enables the use of space IDs for logical addresses to support **multiple** address spaces.

For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Concepts"** (trace32_concepts.pdf).

NOTE:	**SYStem.Option.MMUSPACES** should not be set to **ON** if only one translation table is used on the target. If a debug session requires space IDs, you must observe the following sequence of steps: 1. Activate **SYStem.Option.MMUSPACES**. 2. Load the symbols with **Data.LOAD**. Otherwise, the internal symbol database of TRACE32 may become inconsistent.

Examples:

```
;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x012A:
Data.dump D:0x012A:0xC00208A

;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x0203:
Data.dump D:0x0203:0xC00208A
```

Command has no effect on the TRACE32 Instruction Set Simulator.

SYStem.Option.RUNSTALLMASKASM Disable “RunStall” while step

Command has no effect on the TRACE32 Instruction Set Simulator.

SYStem.Option.SnoopAddressPC Program counter snoop address

Format: **SYStem.Option.SnoopAddressPC** *<address>* | *<addressrange>* | *<name>*

Some SoCs allow to read the program counter during runtime. Use this option to tell the simulator where to read the program counter.

Example:

```
SYStem.Option.SnoopAddressPC EAXI:0x12345678
```

SYStem.Option.SPILLLOC Temporary memory

Command has no effect on the TRACE32 Instruction Set Simulator.

Format: **SYStem.Option.WinRegOption** [*/<option>*]

<option>: **AUTO** | **OFF** | **32** | **64**

Tells the debugger if the Architectural Option with the name Windowed Register Option was configured. It tells you the number of physical registers. You can have:

32 32 core registers if this Architectural Option is not configured to have 32 registers.

64 64 core registers if this Architectural Option is not configured to have 32 registers.

AUTO The option **AUTO** tells the debugger to detect the used setting from the hardware.

OFF 16 core registers if this Architectural Option is not configured.

The **SYStem.TIE** command group is used to configure TRACE32 to deal with architectural extensions. One important extension, the Tensilica Instruction Extension gave the name for this set of commands.

The Tensilica tool chain generates libraries for a custom configuration. These libraries can be used to extract information on the usage of architectural options, additional instructions and registers.

SYStem.TIE.AddCoreLibrary Add library file

Format: **SYStem.TIE.AddCoreLibrary** *<file>*
SYStem.TIE.ADDtiedll *<file>* (deprecated)
SYStem.TIE.ADPerdll *<file>* (deprecated)

Adds TIE library file to the TRACE32, which can be used to improve disassembly for custom configurations. It is important to add all needed library files. The TIE library files need to be added in the correct order, since they are internally dependent. If any file is missing an error may appear after executing **SYStem.TIE.ENAble** command.

The order is as follows:
First "`libisa-core-hw.dll`", then "`libisa-core.dll`" and finally the remaining "`libisa-*.dll`" libraries (if available).

On Linux systems it might be required to add the path to the LD_LIBRARY_PATH environment variable *before* starting TRACE32.

Example:

```
SYStem.TIE.AddCoreLibrary libisa-core.dll
```

For a complete example see **SYStem.TIE.ENAble** command.

SYStem.TIE.CMList Instructions to display custom registers

Format: **SYStem.TIE.CMList** *<file>*

Generates the instructions the debugger needs to display custom registers, see example at **SYStem.TIE.GENper** command.

Format: **SYStem.TIE.DELete**
SYStem.TIE.DEPerdll (deprecated)

Removes all added TIE library files from TRACE32. This command is recommended before **SYStem.TIE.AddCoreLibrary** to be sure that there are no other library files added.

SYStem.TIE.DISable Unload and disable TIE instructions

Format: **SYStem.TIE.DISable**

All loaded TIE library files are unloaded from disassembler decoder. Instructions are decoded only by the internal TRACE32 decoder. To restart decoding with TIE library files use the command **SYStem.TIE.ENAble**.

SYStem.TIE.ENAble Load and enable TIE instructions

Format: **SYStem.TIE.ENAble**

Loads all added TIE library files to the TRACE32 disassembler. From this moment all instructions are decoded by internal TRACE32 decoder and TIE library files. Before you execute this command, it is necessary to add all needed library files to the TRACE32 otherwise an error will appear and TIE library files will not be loaded. To add the file use **SYStem.TIE.AddCoreLibrary** command. To set you disassembler preference use **SYStem.Option.DisMode** command.

Example:

```
SYStem.TIE.RESet

SYStem.TIE.ToolLibraryPath "./tools/lib"

SYStem.TIE.AddCoreLibrary libisa-core-hw.dll    ; Add TIE library files
SYStem.TIE.AddCoreLibrary libisa-core.dll
SYStem.TIE.AddCoreLibrary libisa-DC_330HiFi.dll

SYStem.TIE.ENAble                          ; Load and enable TIE Instructions

SYStem.Option.DisMode LIBrary              ; Set disassembler preference
```

Format: **SYStem.TIE.GENper** *<file>*

Generates a custom peripheral file from the loaded libraries.

Example:

```
SYStem.TIE.ToolLibraryPath "./tools/lib"

SYStem.TIE.AddCoreLibrary libisa-core-hw.dll
SYStem.TIE.AddCoreLibrary libisa-core.dll

SYStem.TIE.GENper "my_hifi2.per"
SYStem.TIE.CMList "my_hifi2.cmm"

DO "my_hifi2.cmm"
PER.view my_hifi2.per"
```

SYStem.TIE.GETArchOPTions Detect architectural options from libraries

Format: **SYStem.TIE.GETArchOPTions** *<file>*

Detects architectural options from the loaded libraries.

Example:

```
SYStem.TIE.ToolLibraryPath "./tools/lib"

SYStem.TIE.AddCoreLibrary libisa-core-hw.dll
SYStem.TIE.AddCoreLibrary libisa-core.dll

SYStem.TIE.GETArchOPTions

SYStem.Mode Up
```

Format: **SYStem.TIE.ToolLibraryPath** *<directory>*
SYStem.TIE.LIBpath *<directory>* (deprecated)

Tells the debugger where to search for the tools to handle core specific libraries.

To extract information from delivered libraries a set of tools is needed. These tools can be found within additional libraries like xtisa.dll, xtparams.dll and xtdebug.dll. TRACE32 needs to know where to find these files.

Be aware that the same release revision is needed as you selected for your XGP request to Cadence®.

On Linux systems it might be required to add the path to the LD_LIBRARY_PATH environment variable *before* starting TRACE32.

SYStem.TIE.RESet Reset TIE

Format: **SYStem.TIE.RESet**

Resets TIE to initial state.

TrOnchip.RESet — Reset on-chip trigger settings

Format: **TrOnchip.RESet**

Resets all TrOnchip settings.

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

MMU.DUMP Page wise display of MMU translation table

Format:	**MMU.DUMP** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.dump** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*

Displays the contents of the CPU specific MMU translation table.

- If called without parameters, the complete table will be displayed.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: displays the translation table of the specified process • else, this command displays the table the CPU currently uses for MMU translation.

KernelPageTable	Displays the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Displays the MMU translation table entries of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

CPU specific Tables

TLB	Displays the contents of the Translation Lookaside Buffer.

Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.List** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**

Lists the address translation of the CPU-specific MMU table.

- If called without address or range parameters, the complete table will be displayed.
- If called without a table specifier, this command shows the debugger-internal translation table. See **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: list the translation table of the specified process • else, this command lists the table the CPU currently uses for MMU translation.
KernelPageTable	Lists the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and lists its address translation.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Lists the MMU translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and lists its address translation. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

Format:	**MMU.SCAN** *<table>* [*<range>* *<address>*] **MMU.***<table>***.SCAN** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** [**Clear**] *<cpu_specific_tables>*

Loads the CPU-specific MMU translation table from the CPU to the debugger-internal static translation table.

- If called without parameters, the complete page table will be loaded. The list of static address translations can be viewed with **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, page table entries will only be loaded if their **logical** address matches with the given parameter.

Use this command to make the translation information available for the debugger even when the program execution is running and the debugger has no access to the page tables and TLBs. This is required for the real-time memory access. Use the command **TRANSlation.ON** to enable the debugger-internal MMU table.

PageTable	Loads the entries of an MMU translation table and copies the address translation into the debugger-internal static translation table. • if *<range>* or *<address>* have a space ID: loads the translation table of the specified process • else, this command loads the table the CPU currently uses for MMU translation.

KernelPageTable	Loads the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Loads the MMU address translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and copies its address translation into the debugger-internal static translation table. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manual**.
ALL [**Clear**]	Loads all known MMU address translations. This command reads the OS kernel MMU table and the MMU tables of all processes and copies the complete address translation into the debugger-internal static translation table. See also the appropriate **OS Awareness Manual**. **Clear:** This option allows to clear the static translations list before reading it from all page translation tables.