LAUTERBACH
DEVELOPMENT TOOLS

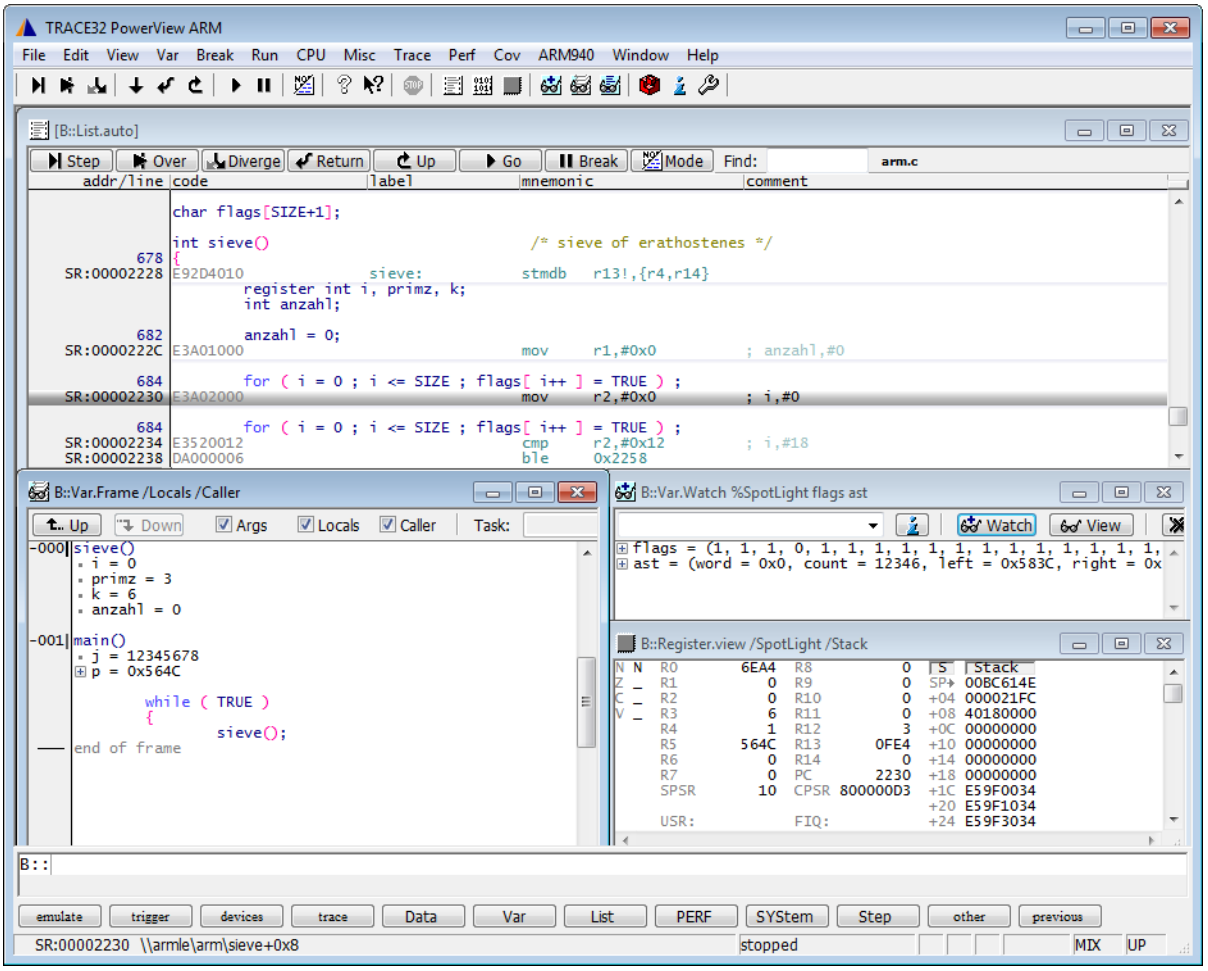
Simulator for Arm and XSCALE

MANUAL

Simulator for Arm and XSCALE

Simulator for Arm and XSCALE

Version 06-Jun-2024



Introduction

This document describes the processor-specific settings and features for the TRACE32 Instruction Set Simulator for ARM.

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**.

Brief Overview of Documents for New Users

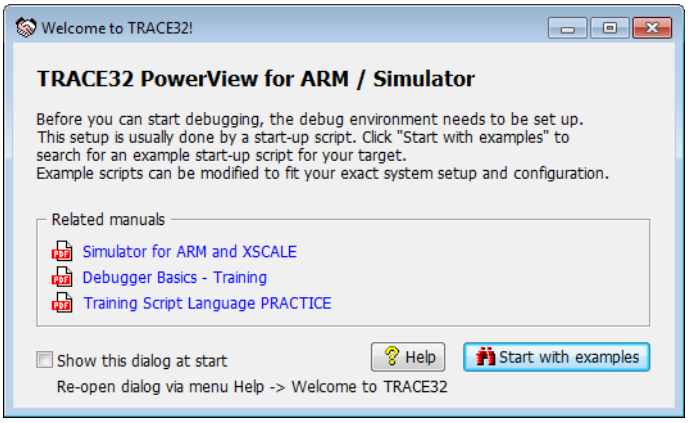
Architecture-independent information:

- **"Training Basic Debugging"** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"T32Start"** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **"General Commands"** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **"OS Awareness Manuals"** (rtos_<*os*>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

To get started with the most important manuals, use the **Welcome to TRACE32!** dialog (**WELCOME.view**):

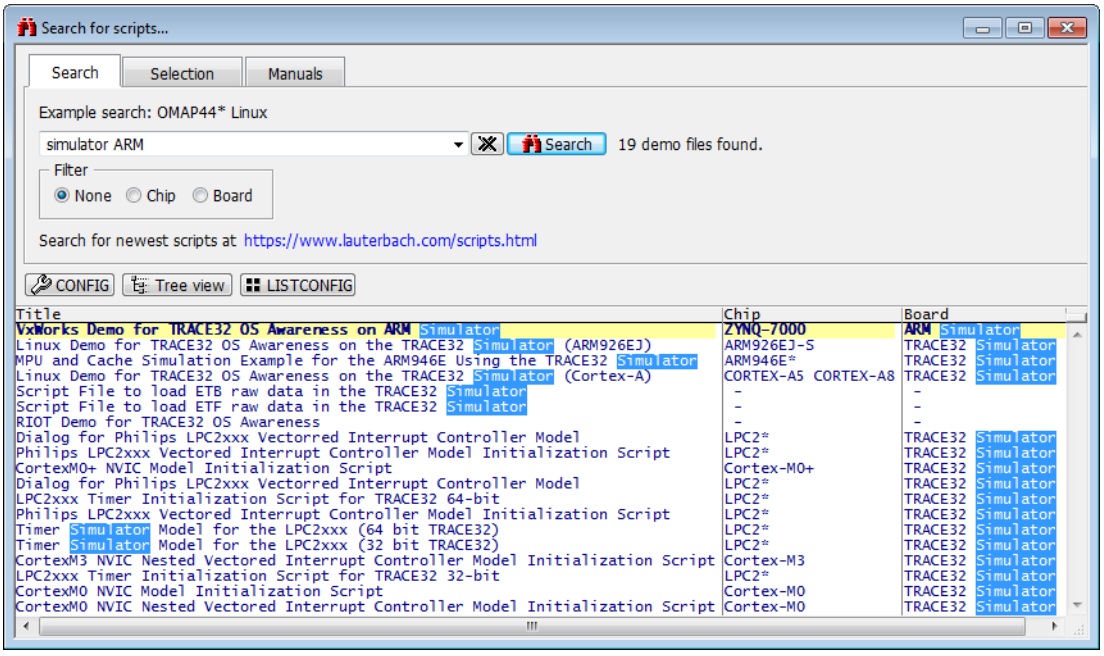


Demo and Start-up Scripts

To search for PRACTICE scripts, do one of the following in TRACE32 PowerView:

- Type at the command line: **WELCOME.SCRIPTS**
- or choose **File** menu > **Search for Script**.

  You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.



You can also manually navigate in the ~~/demo/arm/ subfolder of the system directory of TRACE32.

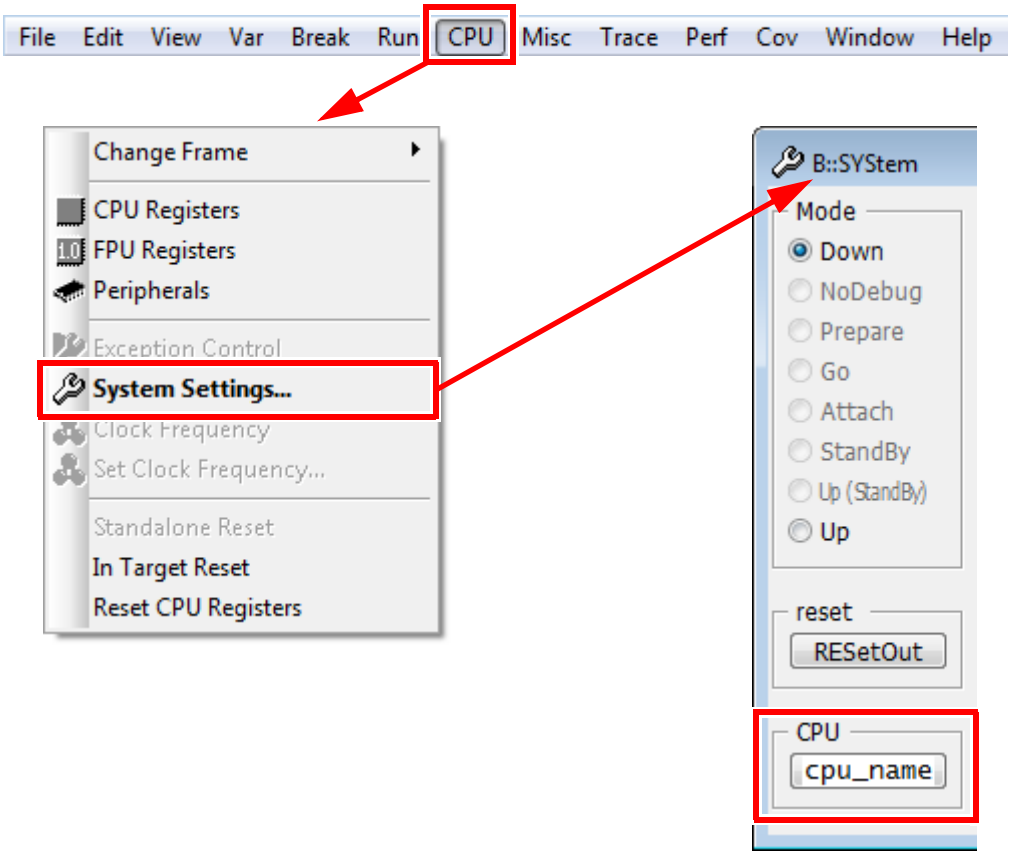
To start the simulator, proceed as follows:

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).

Troubleshooting

No information available.

FAQ

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

Memory Classes

The following ARM specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
SP	Supervisor Program Memory (privileged access)
UP	User Program Memory (non-privileged access)
SR	Supervisor ARM Memory (privileged access)
ST	Supervisor Thumb Memory (privileged access)
UR	User ARM Memory (non-privileged access)
UT	User Thumb Memory (non-privileged access)
U	User Memory (non-privileged access)
S	Supervisor Memory (privileged access)
R	ARM Memory
T	Thumb Memory
ICE	ICE Breaker Register (debug register; ARM7, ARM9)
C14	Coprocessor 14 Register (debug register; ARM10, ARM11)
C15	Coprocessor 15 Register (if implemented)
ETM	Embedded Trace Macrocell Registers (if implemented)
VM	Virtual Memory (memory on the debug system)
USR	Access to Special Memory via User-Defined Access Routines
E	Run-time memory access (see **SYStem.CpuAccess** and **SYStem.MemAccess**)

To access a memory class, write the class in front of the address.

Example:

```
Data.dump ICE:0--3
```

Normally there is no need to use the following memory classes: P, D, SP, UP, SR, ST, UR, UT, U, S, R, or T. The memory class is set automatically depending on the setting of **SYStem.Option.DisMode**.

Virtual Terminal

The command **TERM** opens a terminal window which allows to communicate with the ARM core over the ICEbreaker Debug Communications Channel (DCC). All data received from the comms channel are displayed and all data inputs to this window are sent to the comms channel. Communication occurs byte wide or up to four bytes per transfer. The four bytes ASCII mode (**DCC4A**) does not allow to transfer the byte 00. Each non-zero byte of the 32bit word is a character in this mode. The four byte binary mode (**DCC4B**) can be used to transfer non-ASCII 32bit data (e.g. to or from a file). The three bytes mode (**DCC3**) allows binary transfers of up to 3 bytes per DCC transfer. The upper byte defines how many bytes are transferred (0=one byte, 1= two bytes, 2=three bytes). This is the preferred mode of operation, as it combines arbitrary length messages with high bandwidth. The **TERM.METHOD** command selects which mode is used (**DCC**, **DCC3**, **DCC4A** or **DCC4B**).

The communication mechanism is described e.g. in the ARM7TDMI data sheet in chapter 9.11. Only three move to/from coprocessor 14 instructions are necessary.

The TRACE32 ~~/demo/etc/terminal/serial directory contains the file `TERM.CMM` which demonstrates how the communication works.

Semihosting

The command **TERM.GATE** opens a terminal window which allows to support ARM compatible semihosting. The communication can either be done by stopping the target at the SWI or by using the DCC interface channel - which provides non-stop operation of the target.

The SWI emulation mode requires to stop the target at the SWI exception vector. On ARM7 this can be done only with an on-chip or software breakpoint at location 8. On other ARM cores it can be done by enabling the ICEbreaker breakpoint at the SWI vector (**TrOnchip.Set SWI ON**). The terminal must be set to the **ARMSWI** method (**TERM.METHOD ARMSWI**). The handling of the SWI is only active when the **TERM.GATE** window is existing. An example can be found in ~~/demo/arm/etc/semihosting_arm_emulation/swisoft_<*x*>.cmm.

The DCC communication mode requires an target agent for the SWI. The communication is done in the **DCC3** method of the **TERM** command. An example and the source of the SWI agent can be found in ~~/demo/arm/etc/semihosting_arm_dcc/swidcc_arm7_arm9.cmm.

It is not possible to access coprocessors which are not included in an ARM macrocell from debug mode. This means all coprocessors which are added to ARM cores by customers cannot be accessed from debug mode.

The following coprocessors can be accessed if available in the processor:

Coprocessor 14. Please refer to the chapter **Virtual Terminal** and to your ARM documentation for details.

Coprocessor 15, which allows the control of basic CPU functions. This coprocessor can be accessed with the access class C15. For the detailed definition of the CP15 registers please refer to the ARM data sheet. The CP15 registers can also be controlled in the **PER** window.

The TRACE32 address is composed of the CRn, CRm, op1, op2 fields of the corresponding coprocessor register command

```
<MCR|MRC> p15, <op1>, Rd, CRn, CRm, <op2>
```

BIT0-3:`CRn, BIT4-7:CRm, BIT8-10:<op2>, BIT12-14:<op1>`

is the corresponding TRACE32 address (one nibble for each field)

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**ARM7TDMI** | **ARM740TD** | … (JTAG Debugger ARM7)
	ARM9TDMI | **ARM920T** | **ARM940T** | … (JTAG Debugger ARM9)
	ARM1020E | **ARM1022E** | **ARM1026EJ** | … (JTAG Debugger ARM10)
	ARM1136J | **ARM1136JF** | … (JTAG Debugger ARM11)
	JANUS2 (JTAG Debugger Janus)

Selects the processor type. If your ASIC is not listed, select the type of the integrated ARM core.

SYStem.CONFIG — Configure debugger according to target topology

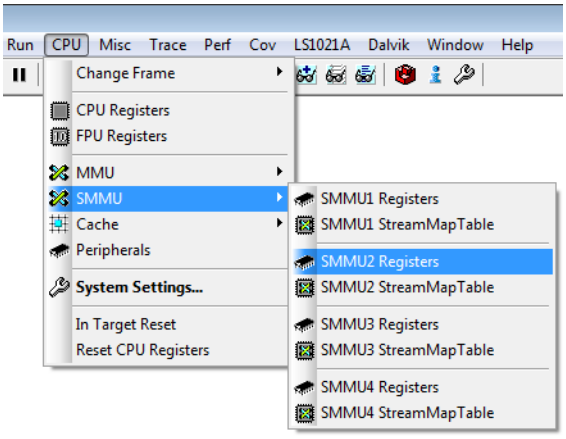
The **SYStem.CONFIG** commands have no effect on the simulator. They are only provided to allow the user to run PRACTICE scripts written for the debugger within the simulator without modifications.

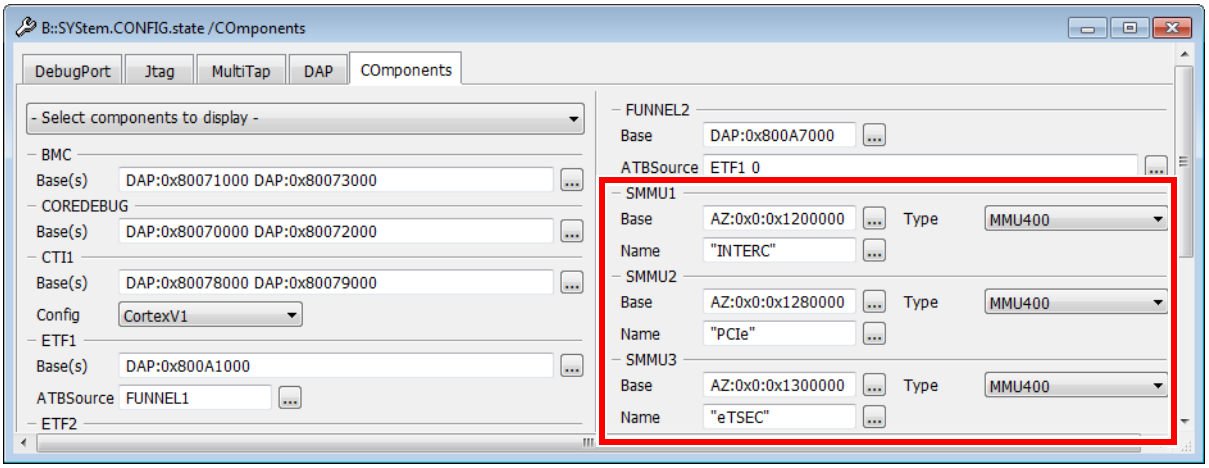
Format:	**SYStem.CONFIG.SMMU***<x> <sub_cmd>*
<x>:	**1** … **20**
<sub_cmd>:	**Base** *<base_address>* **Type MMU400** | **MMU401** | **MMU500** **Name** "*<name>*" **RESet**

For some CPUs with SMMUs, TRACE32 configures the SMMUs parameters *automatically* after you have selected a CPU with the **SYStem.CPU** command.

NOTE: For a *manual* SMMU configuration, use the **SMMU.ADD** command.

You can access the automatically configured SMMUs through the **CPU** menu **> SMMU** submenu in TRACE32. The individual SMMU configurations can be viewed in the **SYStem.CONFIG.state /COmponent** window.





<x>	Serial number of the SMMU.
Base	Logical or physical base address of the memory-mapped SMMU register space.
Type	Defines the type of the Arm system MMU IP block: **MMU400**, **MMU401**, or **MMU500**.
Name	Assigns a user-defined name to an SMMU.
RESet	Resets the configuration of an SMMU specified with *<x>*.

SYStem.Mode Establish the communication with the simulator

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

Default: Down.

Selects the target operating mode.

Down	The CPU is in reset. Debug mode is not active. Default state and state after fatal errors.
NoDebug	The CPU is running. Debug mode is not active. Debug port is tristate. In this mode the target should behave as if the debugger is not connected.
Go	The CPU is running. Debug mode is active. After this command the CPU can be stopped with the break command or if any break condition occurs.
Up	The CPU is not in reset but halted. Debug mode is active. In this mode the CPU can be started and stopped. This is the most typical way to activate debugging.

If the mode **Go** is selected, this mode will be entered, but the control button in the **SYStem.state** window jumps to the mode **Up**.

SYStem.Option.Alignment — Enable alignment exceptions

Format: **SYStem.Option.Alignment**

Causes the processor to go into a DAbort exception for any unaligned access. Otherwise the data will be handled according to the ARM core specification.

SYStem.Option.BigEndian — Define byte order (endianness)

Format: **SYStem.Option**.**BigEndian** [**ON** | **OFF**]

Default: OFF.

Selects the byte ordering mechanism. For correct operation the following settings must correspond:

- This option
- The compiler setting (-li or -bi compiler option)

SYStem.Option.DisMode — Define disassembler mode

Format: **SYStem.Option.DisMode** *<option>*

<option>: **AUTO**
ACCESS
ARM
THUMB

Default: AUTO.

Specifies the selected disassembler.

AUTO	The information provided by the compiler output file is used for the disassembler selection. If no information is available it has the same behavior as the option ACCESS.
ACCESS	The selected disassembler depends on the T bit in the CPSR or on the selected access class. (e.g. `Data.List SR:0` for ARM mode or `Data.List ST:0` for THUMB mode).
ARM	Only the ARM disassembler is used (highest priority).
THUMB	Only the THUMB disassembler is used (highest priority).

SYStem.Option.DUALPORT Implicitly use run-time memory access

Format: **SYStem.Option.DUALPORT** [**ON** | **OFF**]

All TRACE32 windows that display memory are updated while the processor is executing code (e.g. **Data.dump**, **Data.List**, **PER.view**, **Var.View**). This setting has no effect if **SYStem.MemAccess** is disabled.

If only selected memory windows should update their content during runtime, leave **SYStem.Option.DUALPORT OFF** and use the access class prefix **E** or the format option **%E** for the specific windows.

SYStem.Option.IMASKASM Disable interrupts while single stepping

[SYStem.state window > IMASKASM]

Format: **SYStem.Option.IMASKASM** [**ON** | **OFF**]

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during assembler single-step operations. The interrupt routine is not executed during single-step operations. After a single step, the interrupt mask bits are restored to the value before the step.

Format: **SYStem.Option.IMASKHLL** [**ON** | **OFF**]

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during HLL single-step operations. The interrupt routine is not executed during single-step operations. After a single step, the interrupt mask bits are restored to the value before the step.

Format:	**SYStem.Option.MACHINESPACES** [**ON** | **OFF** | **HOSTREMAP**]

Default: OFF

Enables the TRACE32 support for debugging virtualized systems. Virtualized systems are systems running under the control of a hypervisor.

After loading a Hypervisor Awareness, TRACE32 is able to access the context of each guest machine. Both currently active and currently inactive guest machines can be debugged.

ON	Addresses are extended with an identifier called machine ID. The machine ID clearly specifies to which host or guest machine the address belongs. The host machine always uses machine ID 0. Guests have a machine ID larger than 0. TRACE32 currently supports machine IDs up to 30. The debugger address translation (**MMU** and **TRANSlation** command groups) can be individually configured for each virtual machine. Individual symbol sets can be loaded for each virtual machine.
OFF	The machine ID support is disabled.
HOSTREMAP Hypervisor FIASCO	**HOSTREMAP** is only relevant for a hypervisor where: • The hypervisor itself uses tasks and • The tasks behave like virtual machines. If **SYStem.Option.MACHINESPACES** is set to **HOSTREMAP**, then such hypervisor tasks are assigned space IDs instead of machine IDs, whereas the real guest machines are assigned machine IDs. **NOTE**: This option requires a suitable Hypervisor Awareness which supports **HOSTREMAP**. You must also set **SYStem.Option.MMUSPACES** to **ON**.

Machine IDs (0 and > 0)

- On Arm CPUs with hardware virtualization, guest machines are running in the non-secure zone (N:) and use machine IDs > 0.
- The hypervisor functionality is usually running in the hypervisor zone (H:) and uses machine ID 0.
- Software running in the secure monitor mode (Z: for Arm32) or EL3 mode (M: for Arm64) is also using machine ID 0.

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
```

Format: **SYStem.Option.OVERLAY** [**ON** | **OFF** | **WithOVS**]

Default: OFF.

ON Activates the overlay extension and extends the address scheme of the debugger with a 16 bit virtual overlay ID. Addresses therefore have the format *<overlay_id>*:*<address>*. This enables the debugger to handle overlaid program memory.

OFF Disables support for code overlays.

WithOVS Like option **ON**, but also enables support for software breakpoints. This means that TRACE32 writes software breakpoint opcodes to both, the *execution area* (for active overlays) and the *storage area*. This way, it is possible to set breakpoints into inactive overlays. Upon activation of the overlay, the target's runtime mechanisms copies the breakpoint opcodes to the execution area. For using this option, the storage area must be readable and writable for the debugger.

Example:

```
SYStem.Option.OVERLAY ON
Data.List 0x2:0x11c4                    ; Data.List <overlay_id>:<address>
```

SYStem.Option.REALTIME Stall the simulator if faster than real processor

Format: **SYStem.Option.REALTIME** [**ON** | **OFF**]

Default: OFF.

Prevents the simulator from runnig faster than the frequency set with **VCO.Frequency** (default: 10MHz).

Format:	**SYStem.Option.ZoneSPACES** [**ON** | **OFF**]

Default: OFF.

The **SYStem.Option.ZoneSPACES** command must be set to **ON** if an Arm CPU with TrustZone or VirtualizationExtension is debugged. In these Arm CPUs, the processor has two or more CPU operation modes called:

- Non-secure mode
- Secure mode
- Hypervisor mode
- 64-bit EL3/Monitor mode (Armv8-A only)

Within TRACE32, these CPU operation modes are referred to as zones.

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

In each CPU operation mode (zone), the CPU uses separate MMU translation tables for memory accesses and separate register sets. Consequently, in each zone, different code and data can be visible on the same logical addresses.

To ease debug-scenarios where the CPU operation mode switches between non-secure, secure or hypervisor mode, it is helpful to load symbol sets for each used zone.

OFF	TRACE32 does not separate symbols by access class. Loading two or more symbol sets with overlapping address ranges will result in unpredictable behavior. Loaded symbols are independent of Arm zones.
ON	Separate symbol sets can be loaded for each zone, even with overlapping address ranges. Loaded symbols are specific to one of the Arm zones - each symbol carries one of the access classes N:, Z:, H: or M: For details and examples, see below.

If **SYStem.Option.ZoneSPACES** is enabled (**ON**), TRACE32 enforces any memory address specified in a TRACE32 command to have an access class which clearly indicates to which zone the memory address belongs. The following access classes are supported:

N	Non-secure mode Example: Linux user application
Z	Secure mode Example: Secure crypto routine
H	Hypervisor mode Example: XEN hypervisor
M Armv8-A only	64-bit EL3/Monitor mode Example: Trusted boot stage / monitor

If an address specified in a command is not clearly attributed to N: Z:, H: or M:, the access class of the current PC context is used to complete the addresses' access class.

Every loaded symbol is attributed to either non-secure (N:), secure (Z:), hypervisor (H:) or EL3/monitor (M:) zone. If a symbol is referenced by name, the associated access class (N:, Z:, H: or M:) will be used automatically, so that the memory access is done within the correct CPU mode context. As a result, the symbol's logical address will be translated to the physical address with the correct MMU translation table.

NOTE:	The loaded symbols and their associated access class can be examined with command **sYmbol.List** or **sYmbol.Browse** or **sYmbol.INFO**.

Example: Symbols Loading

```
SYStem.Option.ZoneSPACES ON

; 1. Load the vmlinux symbols for non-secure mode (access classes N:, NP:
; and ND: are used for the symbols) with offset 0x0:
Data.LOAD.Elf vmlinux N:0x0 /NoCODE

; 2. Load the sysmon symbols for secure mode (access classes Z:, ZP: and
; ZD: are used for the symbols) with offset 0x0:
Data.LOAD.Elf sysmon Z:0x0 /NoCODE

; 3. Load the xen-syms symbols for hypervisor mode (access classes H:,
; HP: and HD: are used for the symbols) but without offset:
Data.LOAD.Elf xen-syms H: /NoCODE

; 4. Load the sieve symbols without specification of a target access
; class and address:
Data.LOAD.Elf sieve /NoCODE
; Assuming that the current CPU mode is non-secure in this example, the
; symbols of sieve will be assigned the access classes N:, NP: and ND:
; during loading.
```

Example: Symbolic Memory Access

```
; dump the address on symbol swapper_pg_dir which belongs
; to the non-secure symbol set "vmlinux" we have loaded above:

Data.dump swapper_pg_dir

; This will automatically use access class N: for the memory access,
; even if the CPU is currently not in non-secure mode.
```

Example: Deleting Zone-specific Symbols

To delete a complete symbol set belonging to a specific zone, e.g. the non-secure zone, use the following command to delete all symbols in the specified address range.

```
sYmbol.Delete N:0x0--0xffffffff   ; non-secure mode (access classes N:)
```

Example: Zone-specific Debugger Address Translation Setup

If the option **ZoneSPACES** is enabled and the debugger address translation is used (**TRANSlation** commands), a strict zone separation of the address translations is enforced. Also, common address ranges created with **TRANSlation.COMMON** will always be specific for a certain zone.

This script shows how to define separate translations for the zones **N:** and **H:**

```
SYStem.Option.ZoneSPACES ON

Data.LOAD.Elf sysmon   Z:0 /NoCODE
Data.LOAD.Elf usermode N:0 /NoCODE /NoClear

; set up address translation for secure mode
TRANSlation.Create Z:0xC0000000++0x0fffffff A:0x10000000

; set up address translation for non-secure mode
TRANSlation.Create N:0xC0000000++0x1fffffff A:0x40000000

; enable address translation and table walk
TRANSlation.ON

; check the complete translation setup
TRANSlation.List
```

If the CPU's virtualization extension is used to virtualize one or more guest systems, the hypervisor always runs in the CPU's hypervisor mode (zone H:), and the current guest system (if a ready-to-run guest is configured at all by the hypervisor) will run in the CPU's non-secure mode (zone N:).

Often, an operation system (such as a Linux kernel) runs in the context of the guest system.

In such a setup with hypervisor and guest OS, it is possible to load both the hypervisor symbols to H: and all OS-related symbols to N:

A TRACE32 OS Awareness can be loaded in TRACE32 to support the work with the OS in the guest system. This is done as follows:

1. Configure the OS Awareness as for a non-virtualized system. See:
   - **"Training Linux Debugging"** (training_rtos_linux.pdf)
   - **TASK.CONFIG** command
2. Additionally set the default access class of the OS Awareness to the non-secure zone:

```
TASK.ACCESS N:
```

   The TRACE32 OS Awareness is now configured to find guest OS kernel symbols in the **non-secure** zone.

NOTE:	This debugger setup, which is based on the option **ZoneSPACES**, allows work with only one guest system simultaneously. If the hypervisor has configured more than one guest, only the guest that is active in the non-secure CPU mode is visible. To work with another guest, the system must continue running until an inactive guest becomes the active guest. With **SYStem.Option.MACHINESPACES** enabled, TRACE32 also supports concurrent debugging of a virtualized system with hypervisor and multiple guests. the CPU specific zones N: Z: H: and M: will be extended by machine specific zones. Each of these zones is identified by a machine ID. Each guest has its own zone because it uses a separate translation table and a separate register set.

Example: Setup for a Guest OS and a Hypervisor

In this script, the hypervisor is configured to run in zone **H:** and a Linux kernel with OS Awareness as current guest OS in zone **N:**

```
SYStem.Option.ZoneSPACES ON

; within the OS Awareness we need the space ID to separate address spaces
; of different processes / tasks
SYStem.Option.MMUSPACES ON

; here we let the target system boot the hypervisor. The hypervisor will
; set up the guest and boot Linux on the guest system.
...

; load the hypervisor symbols
Data.LOAD.Elf xen-syms H:0 /NoCODE
Data.LOAD.Elf usermode N:0 /NoCODE /NoClear

; set up the Linux OS Awareness
TASK.CONFIG    ~~/demo/arm/kernel/linux/linux-3.x/linux3.t32
MENU.ReProgram ~~/demo/arm/kernel/linux/linux-3.x/linux.men

; instruct the OS Awareness to access all OS-related symbols with
; access class N:
TASK.ACCESS N:

; set up the debugger address translation for the guest OS

; Note that the default address translation in the following command
; defines a translation of the logical kernel addresses range
; N:0xC0000000++0xFFFFFFF to the intermediate address range
; starting at I:0x40000000
MMU.FORMAT linux swapper_pg_dir N:0xC0000000++0xFFFFFFF I:0x40000000

; define the common address range for the guest kernel symbols
TRANSlation.COMMON N:0xC0000000--0xFFFFFFFF

; enable the address translation and the table walk
TRANSlation.TableWalk ON
TRANSlation.ON
```

NOTE: If **SYStem.Option.MMUSPACES ON** is used, all addresses for all zones will show a space ID (such as N:**0x024A**:0x00320100), even if the OS Awareness runs only in one zone (as defined with command **TASK.ACCESS**).

Any task-related command, such as **MMU.List.TaskPageTable** *<task_name>*, will automatically refer to tasks running in the same zone as the OS Awareness.

The command asserts nRESET on the JTAG connector in the TRACE32 In-Circuit Debugger (ICD) but is ignored by the TRACE32 Instruction Set Simulator. However, the command is allowed in the simulator so that you can run scripts which have actually been made for the debugger. For more information about the effect in the debugger, refer to your **Processor Architecture Manual** (debugger_*<arch>*.pdf).

SYStem.state Display SYStem.state window

Format:	**SYStem.state**

Displays the **SYStem.state** window.

TrOnchip.RESet — Reset on-chip trigger settings

Format: **TrOnchip.RESet**

Resets all TrOnchip settings.

TrOnchip.Set — Set bits in the vector catch register

Format: **TrOnchip.Set** *<item>* [**ON** | **OFF**]

<item>:

ARM9, ARM11, Cortex-A/R:
[**FIQ** | **IRQ** | **DABORT** | **PABORT** | **SWI** | **UNDEF** | **RESET**]

Devices having TrustZone (ARM1176, Cortex-A) additionally:
[**NFIQ** | **NIRQ** | **NDABORT** | **NPABORT** | **NSWI** | **NUNDEF** |
SFIQ | **SIRQ** | **SDABORT** | **SPABORT** | **SSWI** | **SUNDEF** | **SRESET** |
MFIQ | **MIRQ** | **MDABORT** | **MPABORT** | **MSWI**]

Devices having a Hypervisor mode (e.g. Cortex-A7, -A15) additionally:
[**HFIQ** | **HIRQ** | **HDABORT** | **HPABORT** | **HSWI** | **HUNDEF** | **HENTRY**]

Cortex-M devices:
[**SFERR** | **HARDERR** | **INTERR** | **BUSERR** | **STATERR** | **CHKERR** |
NOCPERR | **MMERR** | **CORERESET]**

ALIGNMENT

Default: DABORT, PABORT, UNDEF, RESET ON, others OFF.

On devices having TrustZone you can specify for most exceptions if the vector catch shall take effect only in non-secure (N...), secure (S...) or monitor mode (M...), on devices having a Hypervisor mode also in hypervisor mode (H...).

FIQ, ... **HENTRY** — Sets/resets the corresponding bits in the vector catch register of the core. If the bit of a vector is set and the corresponding exception occurs, the processor enters debug state as if there had been a breakpoint set on an instruction fetch from that exception vector.

SFERR — Debug trap on secure fault.

HARDERR Debug trap on hard fault.

INTERR Debug trap on interrupt/exception service errors. These are a subset of other faults and catches before BUSERR or HARDERR.

BUSERR Debug trap on normal bus error.

STATERR Debug trap on usage fault state errors.

CHKERR Debug trap on usage fault enabled checking errors.

NOCPERR Debug trap on usage fault access to coprocessor which is not present or marked as not present in CAR register.

MMERR Debug trap on memory management faults.

CORERESET Reset vector catch. Halt running system if core reset occurs.

StepVector (deprecated) Please see **TrOnchip.StepVector**.

ALIGNMENT Stops the program execution on unaligned accesses.

TrOnchip.StepVector — Step into exception handler

Format: **TrOnchip.StepVector** [**ON** | **OFF**]

Default: OFF.

ON Step into exception handler.

OFF Step over exception handler.

TrOnchip.StepVectorResume — Catch exceptions and resume single step

Format: **TrOnchip.StepVector** [**ON** | **OFF**]

Default: OFF.

When this command is set to ON, the debugger will catch exceptions and resume the single step.

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>*] [**/***<option>*] **MMU.***<table>***.dump** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>* **Fulltranslation**

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 or a specific machine if a space ID and/or a machine ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID and/or machine ID: displays the translation table of the specified process and/or machine • 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**.
MACHINE *<machine_magic>* | *<machine_id>* | *<machine_name>*	The following options are only available if **SYStem.Option.MACHINESPACES** is set to **ON**. Dumps a page table of a virtual machine. The **MACHINE** option applies to **PageTable** and **KernelPageTable** and some *<cpu_specific_tables>.* The parameters *<machine_magic>*, *<machine_id>* and *<machine_name>* are displayed in the **TASK.List.MACHINES** window.
Fulltranslation	For page tables of guest machines both the intermediate address and the physical address is displayed in the **MMU.DUMP** window. The physical address is derived from a table walk using the guest's intermediate page table.

CPU specific Tables for MMU.DUMP <table>

ITLB	Displays the contents of the Instruction Translation Lookaside Buffer.
DTLB	Displays the contents of the Data Translation Lookaside Buffer.
TLB	Displays the contents of the Translation Lookaside Buffer.
TLB0	Displays the contents of the Translation Lookaside Buffer 0.
TLB1	Displays the contents of the Translation Lookaside Buffer 1.
NonSecPageTable	Displays the translation table used if the CPU is in non-secure mode and in privilege level PL0 or PL1. This is the table pointed to by MMU registers TTBR0 and TTBR1 in non-secure mode. This option is only visible if the CPU has the TrustZone and/or Virtualization Extension. In ARMv8 this option is only enabled if Exception levels EL0 or EL1 use AArch32 mode.
SecPageTable	Displays the translation table used if the CPU is in secure mode. This is the table pointed to by MMU registers TTBR0 and TTBR1 in secure mode. This option is only visible if the CPU has the TrustZone Extension.

HypPageTable	Displays the translation table used by the MMU when the CPU is in HYP mode. This is the table pointed to by MMU register HTTBR. This table is only available in CPUs with Virtualization Extension.
IntermedPageTable	Displays the translation table used by the MMU for the second stage translation of a guest machine (intermediate address to physical address). This is the table pointed to by MMU register VTTBR. This table is only available in CPUs with Virtualization Extension.

Examples for Page Tables in Virtualized Systems

Example 1:

```
SYStem.Option.MACHINESPACES ON

; your code to load Hypervisor Awareness and define guest machine setup.

;                              <machine_id>
MMU.DUMP.PageTable /MACHINE     2.


;                              <machine_name>
MMU.DUMP.PageTable /MACHINE     "Dom0"
```

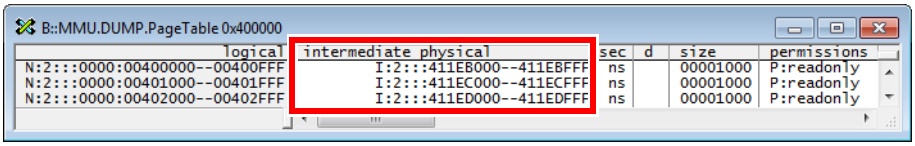
Example 2:

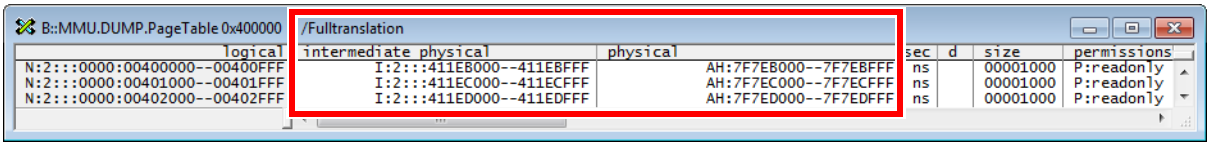
```
SYStem.Option.MACHINESPACES ON

; your code to load Hypervisor Awareness and define guest machine setup.

;                         <machine_name>:::<task_name>
MMU.DUMP.TaskPageTable           "Dom0:::swapper"
```

Example 3:

```
SYStem.Option.MACHINESPACES ON

;your code to load Hypervisor Awareness and define guest machine setup.

;a) dumps the current guest page table of the current machine, showing
;   the intermediate addresses.
;   Without the option /Fulltranslation the column "physical" is hidden.
MMU.DUMP.PageTable 0x400000

;b) With the option /Fulltranslation the intermediate addresses
;   are translated to physical addresses and shown in column "physical"
MMU.DUMP.PageTable 0x400000 /Fulltranslation

;c) dumps the current page table of machine 2
;                              <machine_id>
MMU.DUMP.PageTable  /MACHINE     2.        /Fulltranslation
```

Results for 3 a) and 3 b)





Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] [**/***<option>*]
	MMU.*<table>***.List** (deprecated)
<table>:	**PageTable**
	KernelPageTable
	TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**
	<cpu_specific_tables>
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>*
	Fulltranslation

Lists the address translation of the CPU-specific MMU table.
In contrast to **MMU.DUMP,** multiple consecutive page table entries with identical page attributes are listed as a single line, showing the total mapped address range.

- 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 or a specific machine if a space ID and/or a machine ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID and/or machine ID: list the translation table of the specified process and/or machine • 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**.
<option>	For description of the options, see **MMU.DUMP**.

CPU specific Tables for MMU.List <table>

NonSecPageTable	Displays the translation table used if the CPU is in non-secure mode and in privilege level PL0 or PL1. This is the table pointed to by MMU registers TTBR0 and TTBR1 in non-secure mode. This option is only visible if the CPU has the TrustZone and/or Virtualization Extension. In ARMv8 this option is only enabled if Exception levels EL0 or EL1 use AArch32 mode.
SecPageTable	Displays the translation table used if the CPU is in secure mode. This is the table pointed to by MMU registers TTBR0 and TTBR1 in secure mode. This option is only visible if the CPU has the TrustZone Extension.
HypPageTable	Displays the translation table used by the MMU when the CPU is in HYP mode. This is the table pointed to by MMU register HTTBR. This table is only available in CPUs with Virtualization Extension.
IntermedPageTable	Displays the translation table used by the MMU for the second stage translation of a guest machine (intermediate address to physical address). This is the table pointed to by MMU register VTTBR. This table is only available in CPUs with Virtualization Extension.

Format:	**MMU.SCAN** *<table>* [*<range>* *<address>*] [*/<option>*] **MMU.***<table>***.SCAN** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** *<cpu_specific_tables>*
<option>:	**MACHINE** *<machine_magic>* | *<machine_id>* | *<machine_name>* **Fulltranslation**

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 and/or machine ID: loads the translation table of the specified process and/or machine • 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	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**.
<option>	For description of the options, see **MMU.DUMP**.

CPU specific Table for MMU.SCAN <table>

OEMAddressTable	Loads the OEM Address Table from the CPU to the debugger-internal translation table.

SMMU — Hardware system MMU (SMMU)

Using the **SMMU** command group, you can analyze the current setup of up to 20 system MMU instances. Selecting a CPU with a built-in SMMU activates the **SMMU** command group.

```
SYStem.CPU CortexA53  ;for example, the 'CortexA53' CPU is SMMU-capable

SMMU.ADD ...      ;you can now define an SMMU, e.g. an SMMU for a
                  ;graphics processing unit (GPU)
```

Some SoC CPU types have already SMMUs predefined as component, visible in the **SYStem.CONFIG** component dialog window.

TRACE32 supports the SMMU types MMU-400, MMU-401 and MMU-500 (based on the Arm SMMU architecture specification v2, short SMMU-v2) and MMU-600 (based on the Arm SMMU architecture specification v3, short SMMU-v3).

The TRACE32 SMMU support visualizes most of the configuration settings of an SMMU. These visualizations include:

- The Stream Table with all Stream Map Register Groups (SMRG, for SMMU-v2) or all Stream Table Entries (STE, for SMMU-v3)
- Access to both the non-secure and the secure SMMU view
- Tabular overview over principal data of each SMRG or STE listed in the Stream Table such as
  - Stream matching register settings (for SMMU-v2)
  - Translation context type (stage 1 / stage 2 enabled / bypass / fault)
  - The context's stream world of a SMRG (HYPC and MONC flags) or STE (EL1/EL2/EL3)
  - Stage 1 / stage 2 context bank indices (for SMMU-v2)
  - The availability of stage1 and stage 2 page tables, their format and the MMU-enable/disableT state for the stage 1 and/or stage 2 address translation
  - VMID and the number of stage 1 Context Descriptors for a STE (for SMMU-v3)
- The stage 1 Context Descriptor Table for a given STE (for SMMU-v3)
- Page table lists or dumps for stage 1 and/or stage 2 address translation contexts
- A quick indication of contexts where a fault has occurred or contexts that are stalled (SMMU-v2)
- A quick indication of the global SMMU fault status
- CMD Queue and Event Queue dumps with filtering options (for SMMU-v3)

- Peripheral register view:
  - Global Configuration Registers of the SMMU
  - The SMRG / STE Registers
  - The Context Bank Registers (SMMU-v2) / Context Descriptor Registers (SMMU-v3)
  - MMU-specific Registers such as Performance Measurement Unit Registers, Translation Control Unit Registers or Translation Buffer Unit Registers (for SMMU-v3)

A good way to familiarize yourself with the **SMMU** command group is to start with:

- The **SMMU.ADD** command
- The **SMMU.StreamTable** command which offers GUI-based access to almost all SMMU data
- The guide **Overview - How To**
- **Glossary - SMMU**
- **Arguments in SMMU Commands**

The **SMMU.StreamTable** command and the window of the same name serve as your SMMU command and control center in TRACE32. The right-click popup menu in the **SMMU.StreamTable** window allows you to execute all frequently-used SMMU commands through the user interface TRACE32 PowerView.

The other SMMU commands are designed primarily for use in PRACTICE scripts (*.cmm) and for users accustomed to working with the command line.

NOTE:	The primary table of streams is called *Stream Map Table* in the SMMU-v2 architecture specification, whereas it is called *Stream Table* in the SMMU-v3 architecture specification. To keep the TRACE32 user interface simple, a single unified command, **SMMU.StreamTable,** is used to access the table of streams for all supported SMMU architecture versions. **SMMU.StreamTable** replaces the deprecated command **SMMU.StreamMapTable** which was used for SMMU-v2 *Stream Map Table* access in older TRACE32 versions. However, **SMMU.StreamMapTable** remains an accepted command in scripts to preserve backward compatibility.

Overview - How To

This chapter is a brief guide which commands can be used to perform common tasks. The guide is split into two parts: one for MMU-400, MMU-401 and MMU-500 which follow the SMMU-v2 specification and one for MMU-600 and newer which follow the SMMU-v3 specification.

MMU-400, MMU-401 and MMU-500:

How To...	GUI action or commands
Define a new SMMU	**SMMU.Add** To get the non-secure/secure SMMU view, specify a non-secure/secure base address.
View the Stream Table with all SMRGs View the stream configurations and see the context bank indices of stage 1 and stage 2	**SMMU.StreamTable**
List or dump stage 1 or stage 2 page tables of a stream	In **SMMU.StreamTable** window: use popup menu or double click on column **stage 1 pagetbl. fmt** or **stage 2 pagetbl. fmt** **SMMU.StreamMapRegGrp.list** **SMMU.StreamMapRegGrp.Dump**
View a stream's SMRG registers	In **SMMU.StreamTable** window: use popup menu or double click on any column of **stream matching** or **context type** **SMMU.StreamMapRegGrp.Register** **SMMU.Register.StreamMapRegGrp**
View stage 1 or stage 2 context bank registers	In **SMMU.StreamTable** window: use popup menu or double click on column **stage 1 cbndx** or **stage 2 cbndx** **SMMU.StreamMapRegGrp.ContextReg** **SMMU.Register.ContextBank**
View global SMMU registers	In **SMMU.StreamTable** window: use popup menu or double click status line **SMMU.Register.Global**
View global SMMU fault flags	Fault flags are displayed in the status line at the bottom of the **SMMU.StreamTable** window. Alternatively, open the global SMMU registers with **SMMU.Register.Global** and view register SMMU_GFSR / SMMU_sGFSR (non-sec/sec)
Check if an SMMU stream is in a fault state	Open the **SMMU.StreamTable** window: Streams in fault/stall/multi state have red **F**/**S**/**M** marks in column **stage 1 state** or **stage 2 state**
View Security State Determination Table (SSD)	In **SMMU.StreamTable** window: use popup menu **SMMU.SSDtable**

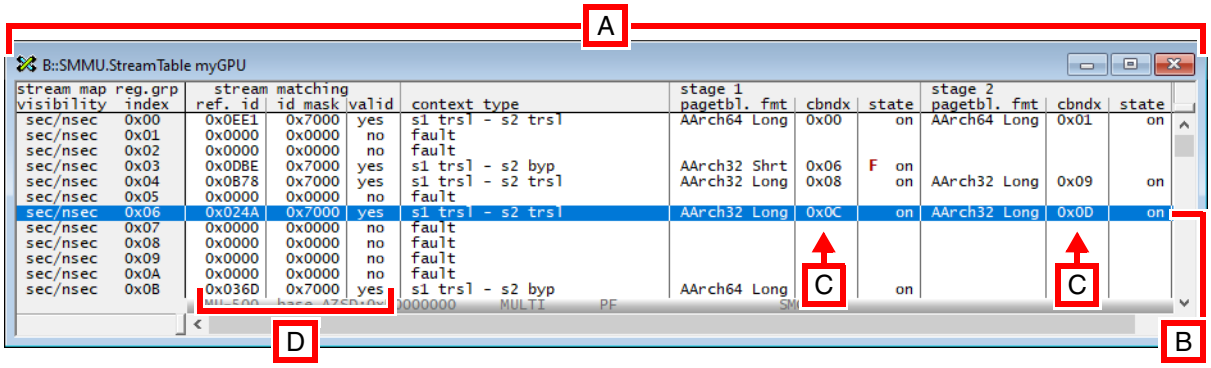
MMU-600 and newer:

How To...	GUI action or commands
Define a new SMMU	**SMMU.Add** Use a secure base address. Default SMMU view is non-secure. Switch to secure view with option **/SECure** in most commands or use check box **Show secure entries** in the header of most SMMU windows.
View the Stream Table with all valid STEs View the stream configuration, VMID, stream world, stage 2 page table format, number of CDs	**SMMU.StreamTable**
View the Context Descriptor Table of a STE with a list of all valid substreams (CDs) View the ASID, stage 1 page table format and TT0/TT1 translation enable state of substreams	In **SMMU.StreamTable** window: use popup menu or click on the STE's **list CDT** button in the **S1 PT fmt** column to open the Context Descriptor Table window. **SMMU.CtxtDescTable**
List or dump stage 2 page tables of a STE	In **SMMU.StreamTable** window: use popup menu or double click on column **S2 PT fmt** or **stage 2 pagetbl. fmt** **SMMU.StreamTblEntry.list** **SMMU.StreamTblEntry.Dump**
List or dump stage 1 page tables of a STE/CD	If STE has only one CD: use popup menu in **SMMU.StreamTable** window or double click on column **S1 PT fmt** to view the CD's page table. If STE has more than one CD: click on the STE's **list CDT** button in the **S1 PT fmt** column to open the Context Descriptor Table window. Here, use popup menu or double click on column **S1 PT fmt**. **SMMU.StreamTblEntry.list** **SMMU.StreamTblEntry.Dump**
View a stream's STE registers	In **SMMU.StreamTable** window: use popup menu or double click on column **configuration** **SMMU.StreamTblEntry.Register** **SMMU.Register.StreamTblEntry**

How To...	GUI action or commands
View the stage 1 CD registers for a substream	If STE has only one CD: use popup menu in **SMMU.StreamTable** window or double click on column **ASID** to view the CD registers. If STE has more than one CD: click on the STE's **list CDT** button in the **S1 PT fmt** column to open the Context Descriptor Table window. Here, use popup menu or double click on column **ASID**. **SMMU.Register.S1Context**
View global SMMU registers	In **SMMU.StreamTable** window: use popup menu or double click status line **SMMU.Register.Global**
View global SMMU fault flags	Fault flags are displayed in the status line at the bottom of the **SMMU.StreamTable** window. Alternatively, open the global SMMU registers with **SMMU.Register.Global** and view register SMMU_GERROR / SMMU_S_GERROR
Check if an SMMU stream or substream is in a fault state Dump Event Queue entries	In the **SMMU.StreamTable** or the **SMMU.CtxtDescTable** window: • either use popup menu **Dump Queue Entries** - **Event Queue** to dump all Event Queue entries • or, with mouse over STE or CD of interest, use popup menu **Dump associated Queue Entries** - **Event Queue** to dump Event Queue entries filtered by Stream ID and Substream ID **SMMU.DumpQueue.Event**
Dump CMD Queue entries	In the **SMMU.StreamTable** or the **SMMU.CtxtDescTable** window: • either use popup menu **Dump Queue Entries** - **CMD Queue** to dump all CMD Queue entries • or, with mouse over STE or CD of interest, use popup menu **Dump associated Queue Entries** - **CMD Queue** to dump CMD Queue entries filtered by Stream ID and Substream ID **SMMU.DumpQueue.CMD**

The following two figures illustrate a few SMMU terms. For explanations of the illustrated SMMU terms and other important SMMU terms not shown here, see below.

MMU-400, MMU-401 and MMU-500:



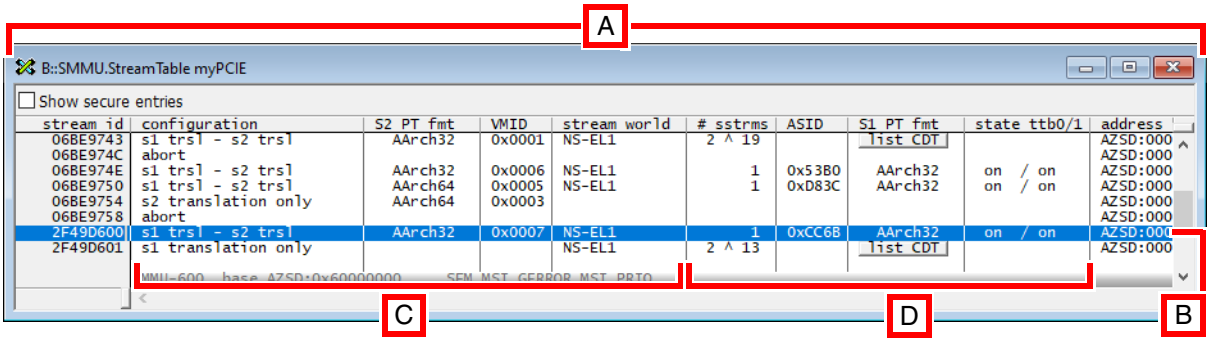
A See stream table.

B Each row stands for a stream map register group (SMRG).

C Index of a translation context bank.

D Data from stream matching registers, see stream matching.

MMU-600 and newer:



A See stream table.

B Each row stands for a stream table entry (STE).

C Stream configuration and stage 2 context.

D Substream data and either stage 1 context or button to view the STE's Context Descriptor Table.

Context Descriptor (CD)

MMU-600 and newer only
A data structure in memory containing register fields which describe a stage 1 translation context, including a pointer to the stage 1 translation table. A CD is identified by its substream ID and by the stream ID of the it belongs to.

Context Descriptor Table (CDT)

MMU-600 and newer only
A table in memory with one or two levels which holds a number of Context Descriptors. Each Context Descriptor Table belongs to one Stream Table Entry.

A CDT can be displayed using command **SMMU.CtxtDescTable**.

Memory Transaction Stream

A stream of memory access transactions sent from a device through the SMMU to the system memory bus. The stream consists of the address to be accessed and a number of design specific memory attributes such as the privilege, cacheability, security attributes or other attributes.

The streams carry a stream ID which the SMMU uses to determine a translation context for the memory transaction stream. As a result, the SMMU may or may not translate the address and/or the memory attributes of the stream before it is forwarded to the system memory bus.

Queue

MMU-600 and newer only
Data structure consisting of a circular buffer in memory which holds queue entries. Queue entries may hold commands for the SMMU (in the CMD Queue) or events generated by the SMMU (in the Event Queue). Queues can be viewed using command **SMMU.DumpQueue**.

Security State Determination Table (SSD Table)

MMU-400, MMU-401 and MMU-500 only
If the SMMU supports two security states (secure and non-secure) an SSD index qualifies memory transactions sent to the SMMU. The SSD index is a hardware signal which is used by the SMMU to decide whether the incoming memory transaction belongs to the secure or the non-secure domain.

The information whether a SSD index belongs to the secure or to the non-secure domain is contained in the SMMU's SSD table.

Stream ID

Peripheral devices connected to an SMMU issue memory transaction streams. Every incoming memory transaction stream carries a Stream Identifier which is used by the SMMU to associate a translation context to the transaction stream. The streams are stored in the Stream Table of the SMMU.

Stream Map Register Group (SMRG)

MMU-400, MMU-401 and MMU-500 only
A group of SMMU registers determining the translation context for a memory transaction stream. The Stream Table holds the SMRGs.

Stream Table (ST) / Stream Mapping Table (SMT)

An SMMU table which describes what to do with an incoming memory transaction stream from a peripheral device. In particular, this table associates an incoming memory transaction stream with a translation context, using the stream ID of the stream as selector of a translation context.

In MMU-400, MMU-401 and MMU-500 (Arm SMMU-v2 specification based), this table of streams is referred to as *Stream Mapping Table*. In MMU-600 and newer (Arm SMMU-v3 specification based), this table of streams is referred to as *Stream Table*. The Stream (Mapping) Table is the central table of the SMMU.

- MMU-400, MMU-401 and MMU-500): each Stream Mapping Table entry consists of a group of registers, called Stream Map Register Group, which describe the translation context. In case an SMMU supports *stream matching*, TRACE32 also displays the *stream matching registers* associated with an entry's stream map register group.
- MMU-600 and newer: the stream table is a data structure in memory and consists of Stream Table Entries which describe the translation context type, the stage 2 translation tables and points to a Context Descriptor Table which holds stage 1 translation contexts.

A Stream Table can be displayed using command **SMMU.StreamTable**.

Stream Matching

MMU-400, MMU-401 and MMU-500 only
In an SMMU which supports stream matching, the stream ID of an incoming memory transaction stream undergoes a matching process to determine which entry of the Stream Table will used to specify the translation context for the stream.
TRACE32 displays the reference ID and the bit mask used by the SMMU to perform the Stream ID matching process in the **SMMU.StreamTable** window.

Stream Table Entry (STE)

MMU-600 and newer only
A data structure in memory describing the translation context for each stream. This data structure register contains fields which describe the type of context, the stage 2 translation context, including a pointer to the stage 2 translation table and a pointer to a Context Descriptor Table holding stage 1 contexts. Each STE is identified by its Stream ID.

Note: for MMU-400, MMU-401 and MMU-500 the entries of the Stream Table are called Stream Map Register Group.

Substream ID

Peripheral devices connected to an SMMU issue memory transaction streams. Every incoming memory transaction stream carries a Stream Identifier which is used by the SMMU to associate a translation context to the transaction stream. The streams are stored in the Stream Table of the SMMU.

Translation Context

A translation context describes the translation process of a incoming memory transaction stream. An incoming memory transaction stream may undergo a stage 1 address translation and/or a stage 2 address translation. Further, the memory attributes of the incoming memory transaction stream may be changed. It is also possible that an incoming memory transaction stream is rendered as fault.

The Stream Table determines which translation context is applied to an incoming memory transaction stream.

Translation Context Bank (short: Context Bank)

MMU-400, MMU-401 and MMU-500 only
A group of SMMU registers specifying the translation context for an incoming memory transaction stream. The registers carry largely the same names and contain the same information as the core's MMU registers describing the address translation process.

The registers of a translation context bank describe the translation table base address, the memory attributes to be used during the translation table walk and translation attribute remapping.

This table provides an overview of frequently-used arguments in SMMU commands. Arguments that are only used in one **SMMU** command are described together with that **SMMU** command.

<name>	User-defined name of an SMMU. Use the **SMMU.ADD** command to define an SMMU and its name. This name will be used to identify an SMMU in all other **SMMU** commands.
<smrg_index>	Index of a stream map register group, e.g. 0x04. The indices are listed in the **index** column of the **SMMU.StreamTable**. The *<smrg_index>* is equivalent to the *<stream_id>* used in MMU-600 and newer. *Only applicable for MMU-400, MMU-401 and MMU-500.*
<cbndx>	Index of a translation context bank. *Only applicable for MMU-400, MMU-401 and MMU-500.*
<stream_id> | *<range>*	Index of a StreamTable Entry or a range of Stream Table Entries. The indices are listed in the **index** column of the **SMMU.StreamTable**. The *<stream_id>* is equivalent to the *<smrg_index>* used in MMU-400, MMU-401 and MMU-500. *Only applicable for MMU-600 and newer.*
<substream_id> | *<range>*	Index of a Context Descriptor Table Entry or a range of Context Descriptor Table Entries. *Only applicable for MMU-600 and newer.*
<address> | *<range>*	Logical address or logical address range describing the start address or the address range to be displayed in the SMMU page table list or dump windows.
IntermediatePT	Used to switch between stage 1 and stage 2 page table or register view: • Omit this option to view the translation table entries or registers of stage 1. • Include this option to view the translation table entries or registers of stage 2.
SECure	Used to switch between the non-secure and the secure SMMU content. • Omit this option to view the non-secure table entries or registers • Include this option to view the secure table entries or registers *Only applicable for MMU-600 and newer.*

Format:	**SMMU.ADD "***<name>***"** *<smmu_type> <base_address>*
<smmu_type>:	**MMU400** | **MMU401** | **MMU500** | **MMU600**

Defines a new SMMU (a hardware system MMU). A maximum of 20 SMMUs can be defined.

NOTE:	For some CPUs with SMMUs, TRACE32 will automatically configure the SMMU parameters, so that you can immediately work with the SMMUs and do not need to manually configure them. After selecting the CPU type, check one of the following locations in TRACE32 to see if there are any pre-configured SMMUs: • The **CPU** menu > **SMMU** popup menu • The **SYStem.CONFIG.state /COmponents** window

Arguments:

<name>	User-defined name of an SMMU. The name must be unique and can be max. 9 characters long. **NOTE**: • For the **SMMU.ADD** command, the name must be quoted. • For *all other* **SMMU** commands, omit the quotation marks from the name identifying an SMMU. See also PRACTICE script example below.
<smmu_type>	Defines the type of the Arm system MMU IP block: • SMMUv2 based: **MMU400**, **MMU401** or **MMU500** • SMMUv3 based: **MMU600**

<base_address>	Logical or physical base address of the memory-mapped SMMU register space. **NOTE** for **MMU400, MMU401, MMU500**: If the SMMU supports two security states (secure and non-secure), not all SMMU registers are visible from the non-secure domain. • If you specify a **secure** address as the SMMU base address, you will see the secure view of the SMMU. • If you specify a **non-secure** address as the SMMU base address, you will only see the non-secure SMMU view. Secure SMMU registers will not be visible. To specify a **secure** address, precede the base address with an access class such as AZSD: or ZSD: Always specify either a secure or a non-secure base address so that the SMMU security view is clearly determined. When executing command **SMMU.ADD**, an access class with ambiguous security status will be augmented to either secure or non-secure, according to the current CPU security status and a warning message will be printed. Access classes with a distinct security status will be left unchanged, e.g. the access classes NSD:, NUD:, HD: etc. **NOTE** for **MMU600 and newer**: if CPU supports two security states, always specify the SMMU base address as a secure address (e.g. ZSD: or AZSD:) so that TRACE32 can access both the secure and non-secure SMMU registers.

Example:

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU600 AZSD:0x50000000

;display the stream table of the SMMU named “myGPU”
SMMU.StreamTable myGPU
```

Format:	**SMMU.Clear** *<name>*

Deletes an SMMU definition, which was created with the **SMMU.ADD** command of TRACE32. The **SMMU.Clear** command does not affect your target SMMU.

To delete all SMMU definitions created with the **SMMU.ADD** command of TRACE32, use **SMMU.RESet**.

Argument:

<name>	For a description of *<name>*, click here.

Example:

```
SMMU.Clear myGPU      ;deletes the SMMU named myGPU
```

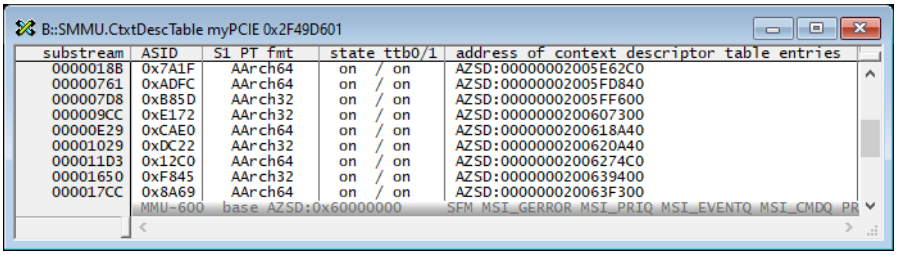
SMMU.CtxtDescTable List a context descriptor table

MMU-600 and newer only

Format:	**SMMU.CtxtDescTable** *<args>*
<args> :	*<name> <stream_id>* [*<substream_id>* | *<range>*] [**/SECure**]

Opens a window and lists all valid stage 1 Context Descriptors in the Context Descriptor Table of the Stream Table Entry specified by *<stream_id>*. Specify option **/SECure** to select the secure SMMU view. A description of the columns is given in this table. The status line of the window shows the global error flags which are currently set for the SMMU.

If you want to limit the Substream IDs displayed in the window, you can specify a numeric *<substream_id>* as lower limit for the displayed SubstreamIDs. Alternatively, you can specify a range as *<substream_id>* to set a lower and an upper limit to the displayed Substream IDs.



Examples:

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU600 AZSD:0x50000000

;list the context descriptors of the stream table with Stream ID 0x6B9743
of the SMMU named "myGPU"
SMMU.CtxtDescTable myGPU 0x6B9743

;same as above, but limit the listing to Substream IDs >= 0x1000
SMMU.CtxtDescTable myGPU 0x6B9743 0x1000

;list the context descriptors of the stream table with secure Stream ID
0x1D73D281 of the SMMU named "myGPU". List only Substream ID in the range
0x1000--0x1FFF
SMMU.CtxtDescTable myGPU 0x1D73D281 0x1000--0x1FFF /SECure
```

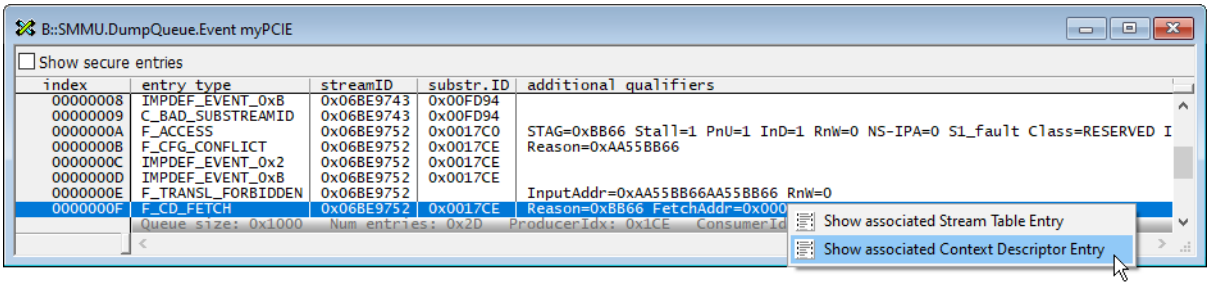
SMMU.DumpQueue.<queue> — Dump entries of a queue

MMU-600 and newer only

Using the **SMMU.DumpQueue** command group, you can dump entries of SMMU Queues. Analyzing entries of the Event Queue is important to find error conditions of SMMU streams - in addition to global error flags of the SMMU.

SMMU.DumpQueue.CMD	Dump entries of the Cmd Queue
SMMU.DumpQueue.Event	Dump entries of the Event Queue

The commands **SMMU.DumpQueue.CMD** and **SMMU.DumpQueue.Event** open a window which shows all valid entries of the queue in the sequence of their creation.



The dump queue windows displays the following columns:

Column	Description
index	Index of the entry. Entries are dumped in the sequence of their creation. The oldest entry always carries index 0 in the dump window. This is the entry pointed to by the queue's Consumer Index register. The newest entry has the largest index in the dump window. This is the entry pointed to by the queue's Producer Index register.
entry type	Decoded type of the queue entry.
secure (CMD queue only)	Indicates the state of the SSec bit in the queue entry. If secure is 1, the entry targets the secure SMMU view, otherwise the non-secure view.
streamID	Shows the content of the entry's Stream ID field. Blank if the entry has no Stream ID field.
substr.ID	Shows the content of the entry's Substream ID field.Blank if the entry has no Substream ID field. For the CMD queue, UNKNOWN is displayed if the entry has a Substream ID field but the entry's SSV (SubStream Valid) bit is 0.
additional qualifiers	Depending on the event type, additional event record fields such as addresses and flags are decoded and printed in this column. Note: it is not supported to filter entries by additional qualifier fields.
address of entry	Displays the physical address of the queue table entry record.

The status line of the window shows the following information:

- the number of entries the queue can hold, i.e. its size
- the number of valid entries it holds currently
- the current producer index
- the current consumer index
- if the queue is full, a message "Queue is FULL" is displayed.

NOTE: Use the popup menu to quickly open **SMMU.StreamTable** or **SMMU.CtxtDescrTable** window. This conveniently allows to view the Stream Table Entry or Context Descriptor associated with the queue entry underneath the mouse pointer.

Filter options

As queues can hold a very large number of entries, command **SMMU.DumpQueue.<queue>** offers filter options allowing dump only entries satisfying certain criteria. The following filter options are available:

Filter option	Description
/QETYPE <qe_type>	Dump only queue entries with entry type *<qe_type>* The values allowed for *<qe_type>* are specific to the queue type and the SMMU type.
/StreamID <stream_id> | <range>	Dump only entries with a certain Stream ID. *<stream_id>* can either be a single numeric value or a numeric range. If it is a range, only those queue entries will be dumped if their Stream ID field falls into the specified range.
/SubStreamID <substream_id> | <range>	Dump only entries with a certain Substream ID. *<substream_id>* can either be a single numeric value or a numeric range. If it is a range, only those queue entries will be dumped if their Substream ID field falls into the specified range. In event queue, entries where the SSV (SubStream Valid) bit is 0 are not dumped at all if the **/SubStreamID** filter is active.

Note that for sake of Stream ID and/or Substream ID filtering, TRACE32 evaluates the event record fields StreamID, SubStreamID and SSV regardless of the queue entry type.

SMMU.DumpQueue.CMD Dump cmd queue entries

MMU-600 and newer only

Format:	**SMMU.DumpQueue.CMD** *<name>* [*<entry_idx>* | *<range>*] [**/SECure**] [*<filter_opts>*]
<entry_idx> | *<range>*	Starts the dump with *<entry_index>* or dumps only entries with index in *<range>*
<filter_opts>:	[**/QETYPE** *<qe_type>*] [**/StreamID** *<stream_id>*] [**/SubstreamID** *<substream_id>*]

Opens the **SMMU.DumpQueue** window and dumps all valid entries of the non-secure or the secure Cmd Queue. See **SMMU.DumpQueue** for a description of the dump queue window.

MMU-600 and newer only

Format:	**SMMU.DumpQueue.Event** *<name>* [*<entry_idx>* | *<range>*] [**/SECure**] [*<filter_opts>*]
<entry_idx> | *<range>*	Starts the dump with *<entry_index>* or dumps only entries with index in *<range>*
<filter_opts>:	[**/QETYPE** *<qe_type>*] [**/StreamID** *<stream_id>*] [**/SubstreamID** *<substream_id>*]

Opens the **SMMU.DumpQueue** window and dumps all valid entries of the non-secure or the secure Event Queue. See **SMMU.DumpQueue** for a description of the dump queue window.

Examples:

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU600 AZSD:0x50000000

;open the event queue dump window for the non-secure SMMU view and dump
all entries
SMMU.DumpQueue.Event myGPU

;open the queue dump window for the secure SMMU view and dump all entries
starting with index 0x200
SMMU.DumpQueue.Event myGPU 0x200 /SECure

;dump only entries of type F_TRANSLATION
SMMU.DumpQueue.Event myGPU /QETYPE F_TRANSLATION

;dump only entries where the Stream ID field is in the range 0x5000--
0x5FFF
SMMU.DumpQueue.Event myGPU /StreamID 0x5000--0x5FFF

;dump only entries where the Stream ID field is 0x6BE900 and the
SubStream ID field is in the range 0x140--0x17F
SMMU.DumpQueue.Event myGPU /StreamID 0x6BE900 /SubStreamID 0x140--0x17F
```

Using the **SMMU.Register** command group, you can view and modify the peripheral registers of an SMMU. The command group provides the following commands:

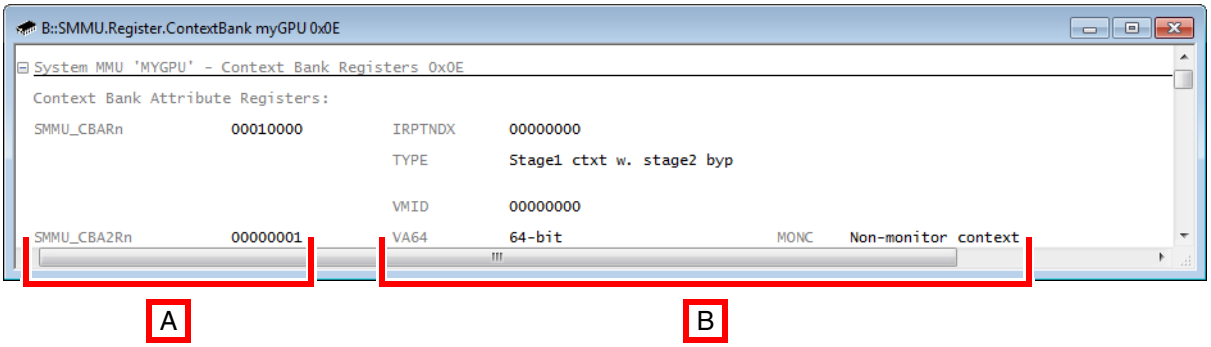
SMMU.Register.Global	Display the global registers of an SMMU
SMMU.Register.ContextBank	Display the registers of a context bank *MMU-400, MMU-401 and MMU-500 only.*
SMMU.Register.StreamMapRegGrp	Display the registers of an SMRG *MMU-400, MMU-401 and MMU-500 only.*
SMMU.Register.StreamTableEntry	Display the registers of a Stream Table Entry. *MMU-600 and newer only.*
SMMU.Register.Stage1Context	Display the registers of a Context Descriptor Table Entry (the stage 1 context of a substream). *MMU-600 and newer only.*

Example:

```
;open the SMMU.Register.StreamMapRegGrp window of SMMU "myGPU" and show
the registers of Stream Table Entry with Stream ID 0x02010A
SMMU.Register.StreamTableEntry   myGPU   0x02010A

;highlight changes in orange in any SMMU.Register.* window
SETUP.Var %SpotLight.on
```

Format:	**SMMU.Register.ContextBank** *<name> <cbndx>*

Opens the peripheral register window **SMMU.Register.ContextBank**. This window displays the registers of the specified context bank. These are listed under the section heading **Context Bank Registers**.



A Register name and content.

B Names of the register bit fields and bit field values.

NOTE: The commands **SMMU.Register.ContextBank** and **SMMU.StreamMapRegGrp.ContextReg** are similar.

The difference between the two commands is:

- The first command expects a *<cbndx>* as an argument and allows to view an arbitrary context bank.
- The second command expects an *<smrg_index>* with an optional **IntermediatePT** as arguments and displays either a stage 1 or stage 2 context bank associated with the *<smrg_index>*.

Argument:

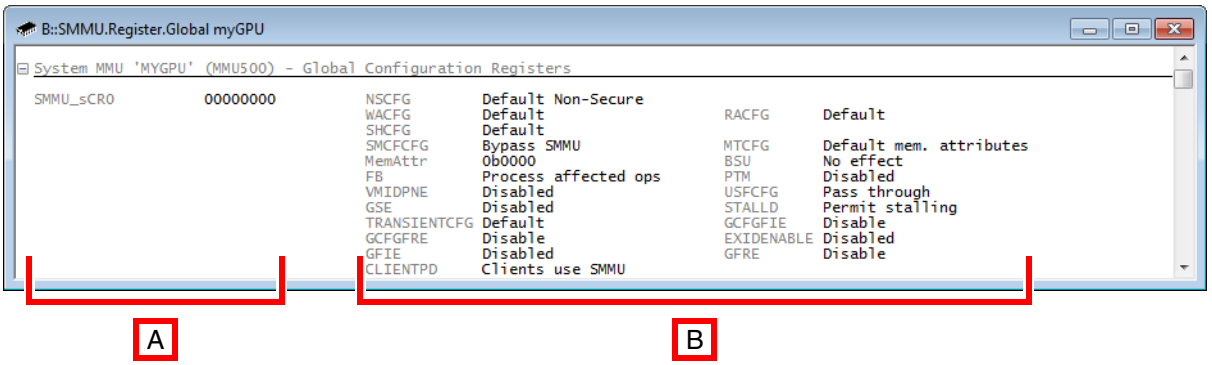
<name>	For a description of *<name>*, etc., click here.

Example:

```
SMMU.Register.ContextBank myGPU 0x16
```

Format: **SMMU.Register.Global** *<name>*

Opens the peripheral register window **SMMU.Register.Global**. This window displays the global registers of the specified SMMU. These are listed under the section heading **Global Configuration Registers**.



A Register name and content.

B Names of the register bit fields and bit field values.

Argument:

<name>	For a description of *<name>*, click here.

Example:

```
SMMU.Register.Global myGPU
```

To display the global registers of an SMMU via the user interface TRACE32 PowerView:

- In the **SMMU.StreamTable** window, right-click an SMRG, and then select **Peripherals** > **Global Configuration Registers** from the popup menu.

SMMU.Register.MMUregs Display MMU specific registers

MMU-600 and newer only

Format: **SMMU.Register.MMUregs** *<name>*

Opens the peripheral register window and shows the MMU specific register blocks which are not part of the SMMU architectural registers. Examples for MMU specific registers are registers for the SMMU Translation Control Unit (TCU), Translation Buffer Unit (TBU) and Performance Measurement Unit (PMU) described in the Arm MMU-600 specification.

SMMU.Register.S1Context Display stage 1 context descriptor registers

MMU-600 and newer only

Format:	**SMMU.Register.S1Context** *<args>*
<args>:	*<name> <stream_id>* **[/SubstreamID *<substream_id>*]** [**/SECure**]

Opens the peripheral register window for the SMMU named *<name>* and displays the registers of a stage 1 Context Descriptor specified by *<stream_id>* and *<substream_id>*.

If the Stream Table Entry specified by *<stream_id>* has only one Context Descriptor, you can omit option **/SubstreamID *<substream_id>***. In this case, the Context Descriptor with Substream ID 0 will be displayed.

Specify option /**SECure** to select the secure SMMU view.

SMMU.Register.StreamTblEntry Display stream table entry registers

MMU-600 and newer only

Format:	**SMMU.Register.StreamTblEntry** *<args>*
<args> :	*<name> <stream_id>* [**/SECure**]

Opens the peripheral register window for the SMMU named *<name>* and displays the registers of the Stream Table Entry which is specified by *<stream_id>*.

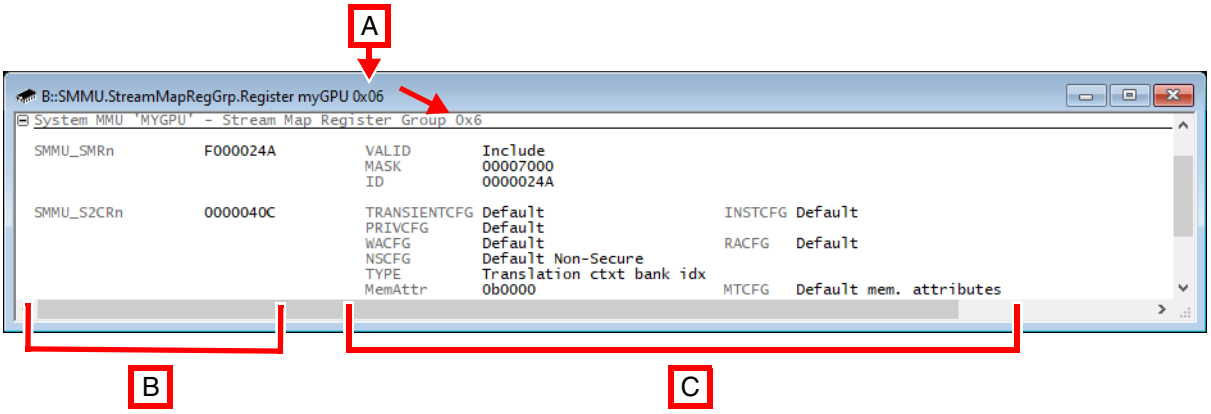
Specify option /**SECure** to select the secure SMMU view.

Example:

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU600 AZSD:0x50000000

;list the Stream Table Entry with Stream ID 0x6B9743 from the secure
Stream Table of SMMU "myGPU"
SMMU.StreamTable myGPU 0x6B9743 /SECure
```

Format:	**SMMU.Register.StreamMapRegGrp** *<args>* **SMMU.StreamMapRegGrp.Register** *<args>* (as an alias)
<args>:	*<name> <smrg_index>*

Opens the peripheral register window **SMMU.Register.StreamMapRegGrp**. This window displays the registers of the specified SMRG. These are listed under the gray section heading **Stream Map Register Group**.



A 0x0D is the *<smrg_index>* of the selected SMRG.

B Register name and content.

C Names of the register bit fields and bit field values.

Compare also to **SMMU.StreamMapRegGrp.ContextReg**.

Arguments:

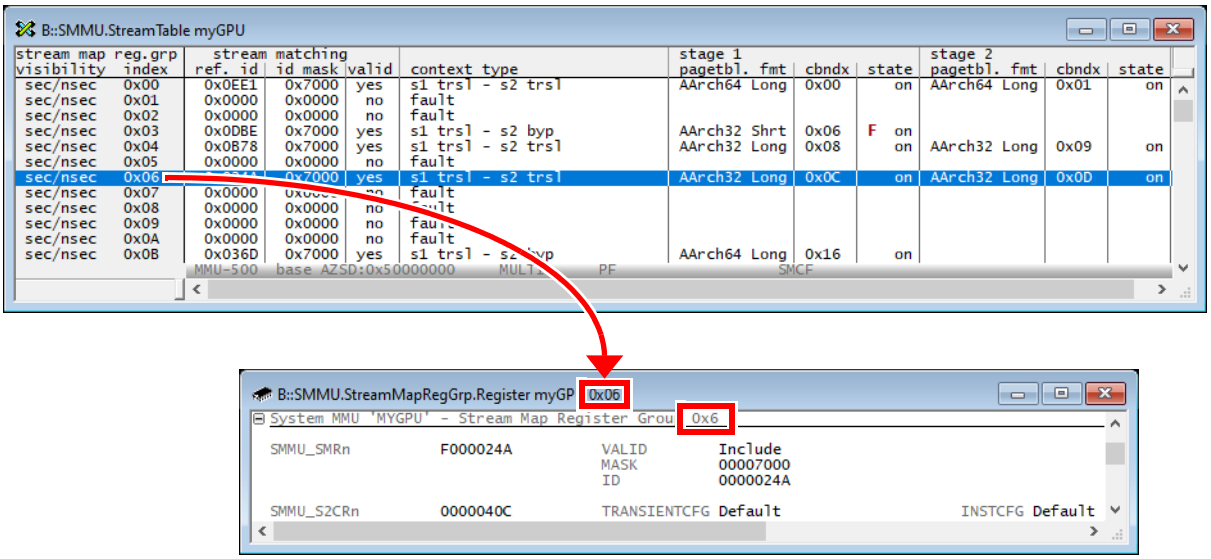
<name>	For a description of *<name>*, etc., click here.

Example:

```
SMMU.StreamMapRegGrp.Register    myGPU    0x06
```

To view the registers of an SMRG via the user interface TRACE32 PowerView:

- In the **SMMU.StreamTable** window, right-click an SMRG, and then select **Peripherals** > **Stream Mapping Registers** from the popup menu.



SMMU.RESet

Delete all SMMU definitions

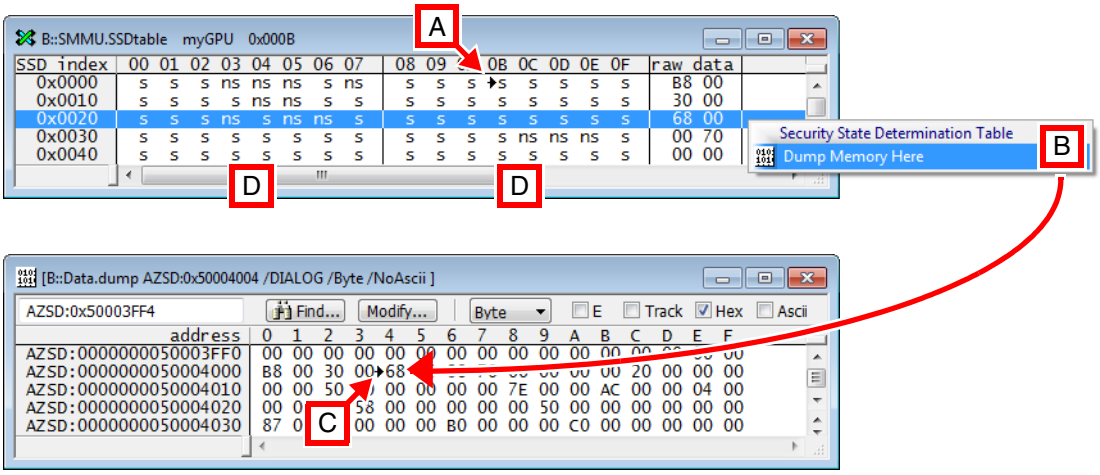
Format: **SMMU.RESet**

Deletes all SMMU definitions created with **SMMU.ADD** from TRACE32. The **SMMU.RESet** command does not affect your target SMMU.

To delete an individual SMMU created with **SMMU.ADD**, use **SMMU.Clear**.

Format: **SMMU.SSDtable** *<name>* [*<start_index>*]

Displays the security state determination table (SSD table) as a bit field consisting of **s** (secure) or **ns** (non-secure) entries. If the SMMU has no SSD table defined, you receive an error message in the **AREA** window.

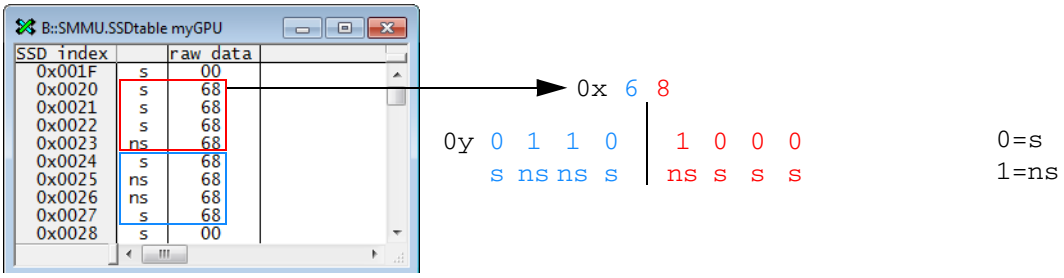


A In the SSD table, the black arrow indicates the *<start_index>*, here 0x00B

B Right-click to dump the SSD table raw data in memory.

For each SSD index of an incoming memory transaction stream, the SSD table indicates whether the outgoing memory transaction stream accesses the secure (**s**) or non-secure (**ns**) memory domain.

You may find the SSD table easier to interpret by reducing the width of the **SMMU.SSDtable** window. Example for the raw data `0x68` in the SSD table:



C In the **Data.dump** window, the black arrow indicates the dumped raw data from the SSD table.

D The 1st white column (00 to 07) relates to the 1st **raw data** column.
The 2nd white column (08 to 0F) relates to the 2nd **raw data** column, etc.

Arguments:

<name>	For a description of *<name>*, click here.
<start_index>	Starts the display of the SSD table at the specified SSD index. See **SSD index** column in the **SMMU.SSDtable** window.

Example:

```
;display the SSD table starting at the SSD index 0x000B
SMMU.SSDtable    myGPU    0x000B
```

To view the SSD table via the user interface TRACE32 PowerView:

- In the **SMMU.StreamTable** window, right-click any SMRG, and then select **Security State Determination Table (SSD)** from the popup menu.

NOTE:	The menu item is grayed out if the SMMU does not support the two security states **s** (secure) or **ns** (non-secure).

SMMU.StreamMapRegGrp — Access to stream map table entries

MMU-400, MMU-401 and MMU-500 only

The **SMMU.StreamMapRegGrp** command group allows to view the details of the translation context associated with stage 1 and/or stage 2 of an SMRG. Every SMRG is identified by its *<smrg_index>*.

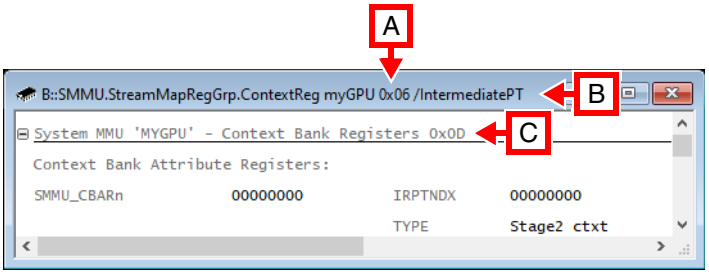
The **SMMU.StreamMapRegGrp** command group provides the following commands:

SMMU.StreamMapRegGrp.ContextReg	Shows the registers of the context bank associated with the stage 1 and/or stage 2 translation.
SMMU.StreamMapRegGrp.Dump	Dumps the page table associated with the stage 1 and/or stage 2 translation page wise.
SMMU.StreamMapRegGrp.list	Lists the page table entries associated with the stage 1 and/or stage 2 translation in a compact format.

Format:	**SMMU.StreamMapRegGrp.ContextReg** *<args>*
<args>:	*<name> <smrg_index>* [**/IntermediatePT**]

Opens the peripheral register window **SMMU.StreamMapRegGrp.ContextReg**, displaying the context bank registers of stage 1 or stage 2 of the specified *<smrg_index>* [**A**]. The context bank index (cbndx) of the shown context bank registers is printed in the gray section heading **Context Bank Registers** [**C**].

The **cbndx** columns in the **SMMU.StreamTable** window tell you which context bank is associated with stage 1 or stage 2: If there is no context bank defined for stage 1 or stage 2, then the respective **cbndx** cell is empty. In this case, the peripheral register window **SMMU.StreamMapRegGrp.ContextReg** does not open.



A 0x0A is the *<smrg_index>* of the selected SMRG.

B The option **IntermediatePT** is used to display the context bank registers of stage 2.

C 0x15 is the index from the **cbndx** column of a stage 2 context bank. See example below.

Compare also to **SMMU.StreamMapRegGrp.Register**.

NOTE: The commands **SMMU.Register.ContextBank** and **SMMU.StreamMapRegGrp.ContextReg** are similar.

The difference between the two commands is:

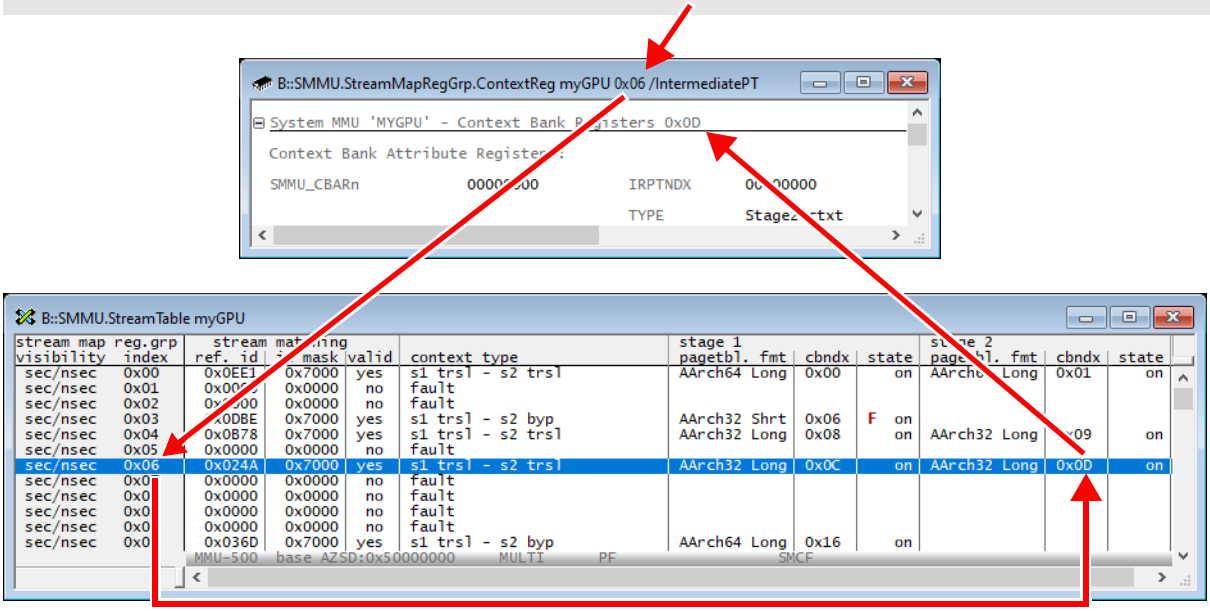
- The first command expects a *<cbndx>* as an argument and allows to view an arbitrary context bank.
- The second command expects an *<smrg_index>* with an optional **IntermediatePT** as arguments and displays either a stage 1 or stage 2 context bank associated with the *<smrg_index>*.

Arguments:

<name>	For a description of *<name>*, etc., click here.

PRACTICE Script Example and Illustration of the Context Bank Look-up:

```
SMMU.StreamMapRegGrp.ContextReg myGPU 0x06 /IntermediatePT
```

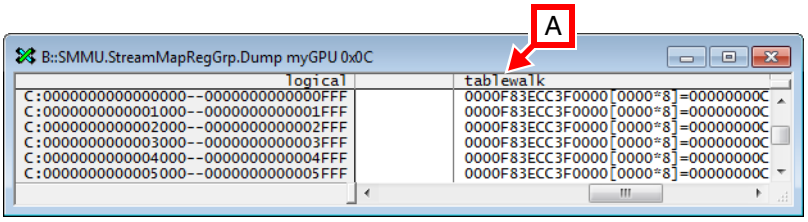


To display the context bank registers via the user interface TRACE32 PowerView:

- In the **SMMU.StreamTable** window, right-click an SMRG, and then select **Peripherals** > **Context Bank Registers of Stage 1** or **2** from the popup menu.

Format:	**SMMU.StreamMapRegGrp.Dump** *<args>*
<args>:	*<name> <smrg_index>* [*<address>* | *<range>* [*<ttb_address>*]] [*/<option>*]

Opens the **SMMU.StreamMapRegGrp.Dump** window for the specified SMRG, displaying the page table entries of the SMRG page wise. If no valid translation context is defined, the window displays the error message “registerset undefined”.



A To view the details of the page table walk, scroll to the right-most column of the window.
For a description of the columns in the **SMMU.StreamMapRegGrp.Dump** window, click here.

Arguments:

<name>	For a description of *<name>*, etc., click here.
<address> | *<range>*	If specified, start the dump with *<address>* or, alternatively, limit the dumped address range to address to *<range>.*
<ttb_address>	If specified, *<ttb_address>* will be used as page table base address. The other page table parameters are still extracted from the SMRG context.
IntermediatePT	Omit this option to view translation table entries of stage 1. Include this option to view translation table entries of stage 2. In SMMUs that support only stage 2 page tables, this option can be omitted.

Example:

```
SMMU.StreamMapRegGrp.Dump myGPU 0x0C
```

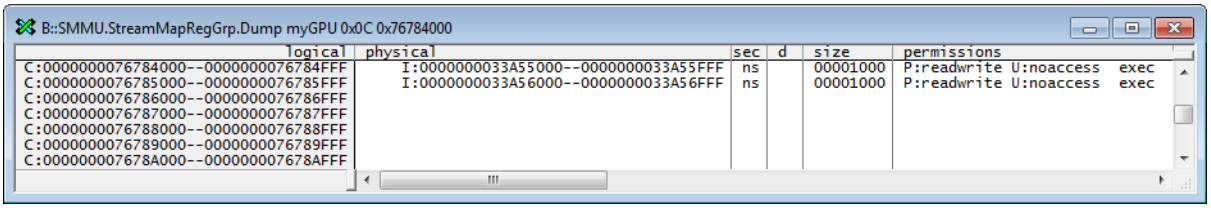
To display an SMMU page table page-wise via the user interface TRACE32 PowerView:

- In the **SMMU.StreamTable** window, right-click an SMRG, and then select from the popup menu:
  - **Stage 1 Page Table** > **Dump** or
  - **Stage 2 Page Table** > **Dump**

Description of Columns

This table describes the columns of the following windows:

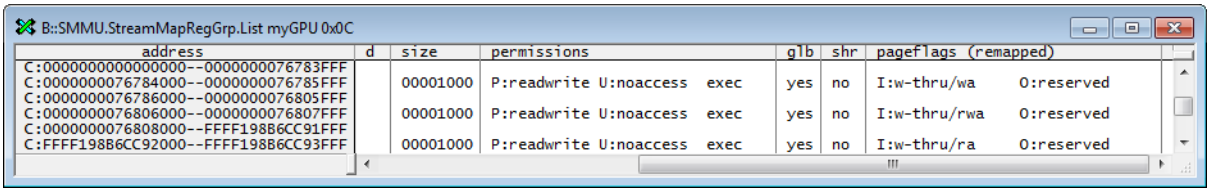
- **SMMU.StreamMapRegGrp.list** / **SMMU.StreamTblEntry.list**
- **SMMU.StreamMapRegGrp.Dump** / **SMMU.StreamTblEntry.Dump**



Column	Description
logical	Logical page address range
physical	Physical page address range
sec	Security state of entry (s=secure, ns=non-secure, sns=non-secure entry in secure page table)
d	Domain
size	Size of mapped page in bytes
permissions	Access permissions (P=privileged, U=unprivileged, exec=execution allowed)
glb	Global page
shr	Shareability (no=non-shareable, yes=shareable, inn=inner shareable, out=outer shareable)
pageflags	Memory attributes (see **Description of the memory attributes**.)
tablewalk	Only for **SMMU.StreamMapRegGrp.Dump**: • Details of table walk for logical page address (one sub column for each table level, showing the table base address, entry index, entry width in bytes and value of table entry)

Format:	**SMMU.StreamMapRegGrp.list** *<args>*
<args>:	*<name> <smrg_index>* [*<address>* | *<range>* [*<ttb_address>*]] [**/IntermediatePT**]

Opens the **SMMU.StreamMapRegGrp.list** window for the specified SMMU, listing the page table entries of a stream map group. If no valid translation context is defined, the window displays an error message.



For a description of the columns in the **SMMU.StreamMapRegGrp.list** window, click here.

Arguments:

<name>	For a description of *<name>*, etc., click here.
<address> / *<range>*	If specified, start the page table list with *<address>* or, alternatively, limit the listed address range to address to *<range>*.
<ttb_address>	If specified, *<ttb_address>* will be used as page table base address. The other page table parameters are still extracted from the SMRG context.
IntermediatePT	Omit this option to view translation table entries of stage 1. Include this option to view translation table entries of stage 2. In SMMUs that support only stage 2 page tables, this option can be omitted.

Example:

```
SMMU.StreamMapRegGrp.list myGPU 0x0C
```

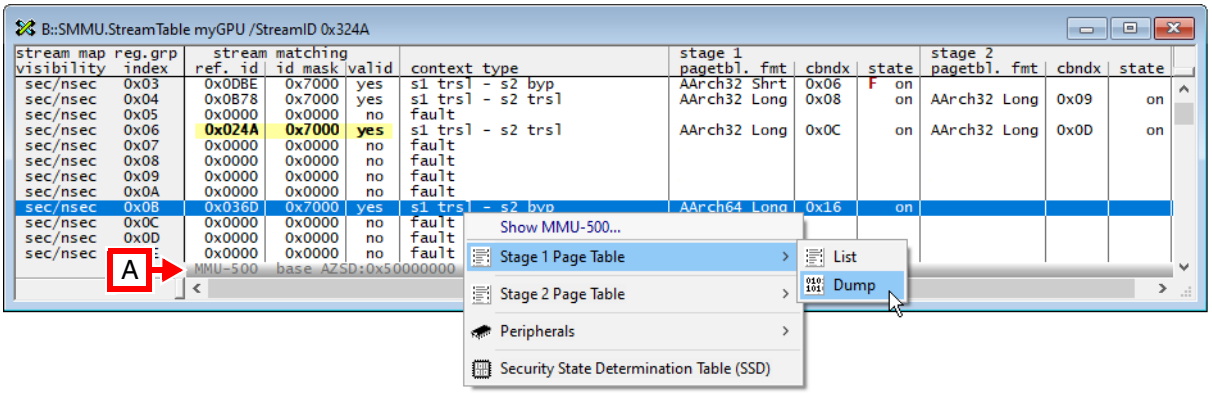
To list the page table entries via the user interface TRACE32 PowerView:

- In the **SMMU.StreamTable** window, right-click an SMRG, and then select from the popup menu:
  - **Stage 1 Page Table** > **List** or
  - **Stage 2 Page Table** > **List**

Format:	**SMMU.StreamTable** *<args>*
	SMMU.StreamMapTable *<args>* **(as an alias)**
<args>:	*<name>* [**/StreamID** *<value>*] (for MMU-400, MMU-401 and MMU-500)
	<name> [*<stream_id>*] [**/SECure**] (for MMU-600 and newer)

Opens the **SMMU.StreamTable** window for the SMMU that has the specified *<name>*. The content and popup menu depends on the SMMU type for which the **SMMU.StreamTable** window is opened. The two variants of the window are described as follows:

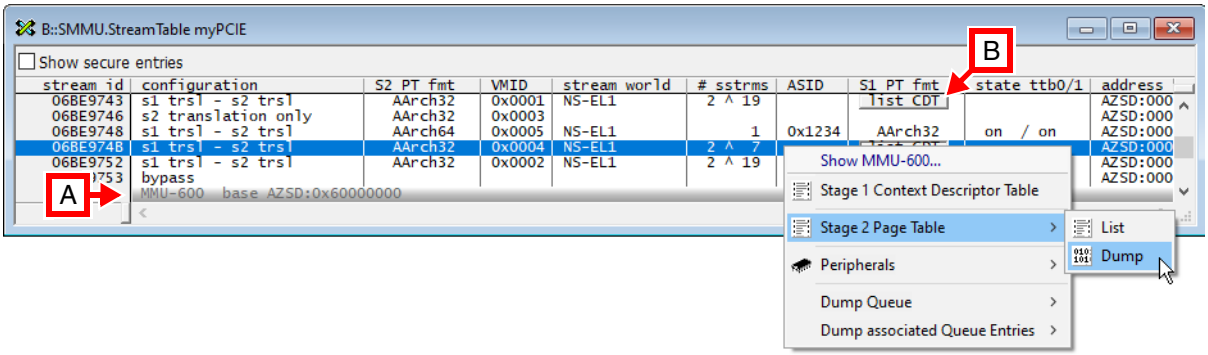
MMU-400, MMU-401, MMU-500:
The window lists all Stream Map Register Groups of the secure or non-secure view of the SMMU. The window provides an overview of the secure or non-secure SMMU configuration.



A The gray window status bar displays the *<smmu_type>* and the SMMU *<base_address>*.
In addition, the window status bar informs you of global faults in the SMMU, if there are any faults.

MMU-600 and newer:
The window lists all valid Stream Table Entries of either the secure or the non-secure view of the SMMU. The security status of the view can be changed using option **/SECure** or, alternatively, using the **Show secure entries** checkbox in the window header.

The Stream ID range displayed can be limited if argument *<stream_id>* is used. You can either specify a number as start value or a range.



A The gray window status bar displays the *<smmu_type>* and the SMMU *<base_address>*.
In addition, the window status bar informs you of global faults in the SMMU, if there are any faults.

B For STEs with more than one substream, click the button **list CDT** to view the substreams.

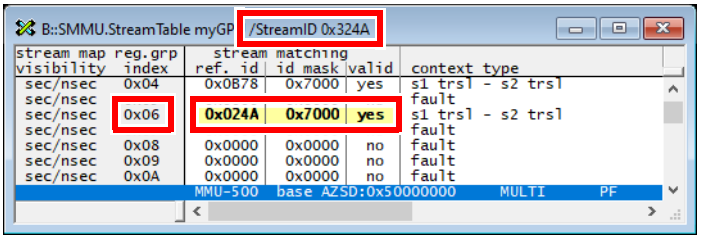
Arguments

<name>	For a description of *<name>*, click here.
StreamID *<value>* (MMU-400, MMU-401 and MMU-500 only)	Only available for SMMUs that support stream ID matching. The **StreamID** option highlights all SMRGs in yellow that match the specified stream ID *<value>*. SMRGs highlighted in yellow help you identify incorrect settings of the stream matching registers. For *<value>*, specify the stream ID of an incoming memory transaction stream. • The highlighted SMRG indicates which stream map table entry will be used to translate the incoming memory transaction stream. • More than one highlighted row indicates a potential, global SMMU fault called stream match conflict fault. The stream ID matching algorithm of TRACE32 mimics the SMMU stream matching on the real hardware. The reference ID, mask and validity fields of the stream match register are listed in the **ref. id**, **id mask** and **valid** columns.
<stream_id> (MMU-600 and newer only)	Either the start point (if a single number is given) or numeric range (if a numeric range is given) of Stream IDs that are displayed in the window.

[Back to Top]

MMU-400, MMU-401, MMU-500:
This PRACTICE script example shows how to define an SMMU with the **SMMU.ADD** command. Then the script opens the SMMU in the **SMMU.StreamTable** window, searches for the *<stream_id>* 0x**3**24A and highlights the matching SMRG 0x**0**24A in yellow.

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU500 AZSD:0x50000000

;open the window and highlight the matching SMRG in yellow
SMMU.StreamTable myGPU /StreamID 0x324A
```



NOTE:	At first glance, the **Stream ID** 0x**3**24A does not seem to match the SMRG 0x**0**24A. However, if you take the ID mask 0x7000 (= 0y0111_0000_0000_0000) into account, the match is correct.

The row highlighted in yellow in the **SMMU.StreamTable** window is a correct match for the **Stream ID** 0x324A we searched for.

See also function **SMMU.StreamID2SMRG()** in **"General Function Reference"** (general_func.pdf).

MMU-600 and newer:
This PRACTICE script example shows how to define an SMMU with the **SMMU.ADD** command. Then the script opens the SMMU in the **SMMU.StreamTable** window starting with Stream ID 0x10000

```
;define a new SMMU named "myGPU" for a graphics processing unit
SMMU.ADD "myGPU" MMU600 AZSD:0x50000000

;open the Stream Table window, showing entries starting with
Stream ID 0x10000
SMMU.StreamTable myGPU 0x10000
```

[Back to Top]

By right-clicking an entry or double-clicking certain cells of an entry, you can open additional windows to receive more information about the selected entry.

- Right-clicking opens the **Popup Menu**.

MMU-400, MMU-401, MMU-500:

- Double-clicking an entry in the columns **ref. id**, **id mask**, **valid**, or **context type** opens the **SMMU.StreamMapRegGrp.Register** window.
- Double-clicking an SMRG in the two columns **pagetbl. fmt** opens the **SMMU.StreamMapRegGrp.list** window, displaying the page table for stage 1 or stage 2.
- Double-clicking an SMRG in the two **cbndx** columns or the two **state** columns opens the **SMMU.StreamMapRegGrp.ContextReg** window, displaying the context bank registers for stage 1 or stage 2.

MMU-600 and newer:

- Double-clicking an entry in the columns **configuration**, **VMID**, **stream world**, or **# sstrms** opens the **SMMU.StreamTblEntry.Register** window showing the stream entry registers.
- Double-clicking an entry in the column **S2 PT fmt** opens the **SMMU.StreamTblEntry.list** window, displaying the stage 2 page table.

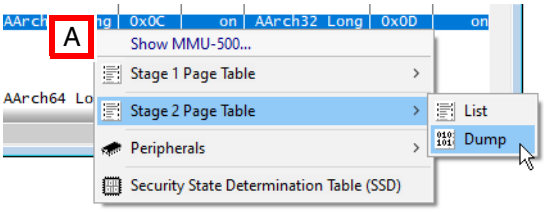
If an entry has only one stage 1 context descriptor:

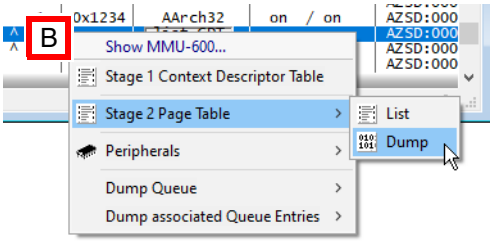
- Double-clicking valid data in columns **ASID** or **state ttb0/1** opens the **SMMU.Register.S1Context** window, displaying the stage 1 context registers.
- Double-clicking valid data in column **S1 PT fmt** opens the **SMMU.StreamTblEntry.list** window, displaying the stage 1 page table.

If an entry has more than one stage 1 context descriptor:

- Click on the list CDT button in column **S1 PT fmt** to open the **SMMU.CtxtDescTable** window, listing all valid Context Descriptors for the stream entry. The **SMMU.CtxtDescTable** window allows to view the registers and stage 1 page tables associated with each Context Descriptor.

[Back to Top]





A Example popup menu for MMU-400, MMU-401 and MMU-500

B Example popup menu for MMU-600 and newer

The entries visible in the popup menus depend on the capabilities of the SMMU such as the capability to support stage 1 or stage 2 and if the SMMU supports two security states.

The popup menu in the **SMMU.StreamTable** window provides convenient shortcuts to the following commands:

MMU-400, MMU-401 and MMU-500:

Popup Menu	Command
Stage 1 Page Table > **Stage 2 Page Table >**	(--)
• **List** • **Dump**	• **SMMU.StreamMapRegGrp.list** • **SMMU.StreamMapRegGrp.Dump**
Peripherals >	(--)
• **Global Configuration Registers** • **Stream Mapping Registers** • **Context Bank Registers of Stage 1** and **Context Bank Registers of Stage 2**	• **SMMU.Register.Global** • **SMMU.Register.StreamMapRegGrp** • **SMMU.Register.ContextBank**
Security State Determination Table (SSD)	**SMMU.SSDtable**

MMU-600 and newer:

Popup Menu	Command
Stage 1 Context Descriptor Table	**SMMU.CtxtDescTable**
Stage 1 Page Table > **Stage 2 Page Table >**	(--)
• **List** • **Dump**	• **SMMU.StreamTblEntry.list** • **SMMU.StreamTblEntry.Dump**
Peripherals >	(--)
• **Global Configuration Registers** • **MMU specific Registers** • **Stream Table Entry Registers** • **Stage 1 Context Descriptor Registers**	• **SMMU.Register.Global** • **SMMU.Register.MMU** • **SMMU.Register.StreamTblEntry** • **SMMU.Register.S1Context**
Dump Queue > **Dump associated Queue Entries >**	(--)
• **Event Queue** • **Cmd Queue**	• **SMMU.DumpQueue.Event** • **SMMU.DumpQueue.CMD**

[Back to Top]

MMU-400, MMU-401 and MMU-500:

Column Name	Description
stream map reg. grp	• **visibility**: The column is only visible if the SMMU supports the two security states *secure* and *non-secure*. The label **sec/nsec** indicates that the SMRG is visible to secure and non-secure accesses. The label **sec only** indicates that the SMRG is visible to secure accesses only. • **index**: The index numbers start at **0x00** and are incremented by 1 per SMRG.
stream matching	See description of the columns **ref. id**, **id mask**, and **valid** below.
ref. id, **id mask**, and **valid**	If the SMMU supports *stream matching*, then the following columns are visible: **ref. id**, **id mask**, and **valid**. Otherwise, these columns are hidden.
context type	Depending on the translation context of a stream mapping register group, the following values are displayed [Description of Values]: • s2 translation only • s1 trsl - s2 trsl • s1 trsl - s2 fault • s1 trsl - s2 byp • fault (s1 trsl-s2 trsl) • fault (s1 trsl-s2 flt) • fault (s1 trsl-s2 byp) • fault • bypass mode • reserved • HYPC or MONC
stage 1 pagetbl. fmt or **stage 2 pagetbl. fmt**	Displays the page table format of **stage 1** or **stage 2** [Description of Values]: • Short descr. (32-bit Arm architecture only) • Long descr. (32-bit Arm architecture only) • AArch32 Short (64-bit Arm architecture only) • AArch32 Long (64-bit Arm architecture only) • AArch64 Long (64-bit Arm architecture only)
cbndx	Displays the context bank index (cbndx) associated with the translation context of **stage 1** or **stage 2**.

Column Name	Description
state	Displays whether the MMU of **stage 1** or **stage 2** is enabled (ON) or disabled (OFF) and whether a fault has occurred in a translation context bank: • **F**: any single fault • **M**: multiple faults • **S**: the SMMU is stalled The letters F, M, and S are highlighted in red in the **SMMU.StreamTable** window (example). The information about the faults is derived from the register SMMU_CBn_FSR (fault status register of the context bank). Double-click the respective **state** cell to open the **SMMU.StreamMapRegGrp.ContextReg** window. The register SMMU_CBn_FSR provides details about the fault.

MMU-600 and newer:

Column Name	Description
configuration	Depending on the translation context of a stream entry, the following values are displayed [Description of Values]: • s1 translation only • s2 translation only • s1 trsl - s2 trsl • bypass • abort A misconfiguration of the stream entry is indicated by a display of ILLEGAL.
S2 PT fmt or **S1 PT fmt**	Displays the page table format of **stage 2** or **stage 1**: • AArch32 • AArch64
VMID	Displays the VMID of the stream table entry stage 2 registers
stream world	Depending of the stream world of a stream entry, the following values are displayed: • NS-EL1 • EL2 • EL2-E2H • EL3 • Secure • Reserved
# sstrms	Displays the max. number of stage 1 context descriptors for the stream table entry, as configured in the S1CDMAX field
ASID	Displays the ASID of a stage 1 context descriptor

Column Name	Description
S1 PT fmt	If only a single context descriptor entry exists in the CDT associated with the stream table entry, it's stage 1page table format is displayed (AArch32 or AArch64). If the CDT contains more than one entry, a button labelled *list CDT* is displayed which directly opens the CDT.
state ttb0/1	Displays the state of the stage 1 context **tt0** / **tt1** translation table disable bits, where **tt0** refers to the address translation of the lower address range. **tt1** refers to the address translation of the upper address range. Possible values for: **tt0** / **tt1** • **on** means the translation for the tt0 / tt1 address range is enabled • **off** means the translation for the tt0 / tt1 address range is disabled
address of stream table entries ***or*** **address of context desciptor table entries**	Displays table walk details, i.e. the physical addresses of the level 1 and/or level 2 table entries. If the table has only one level, one address is displayed, for a 2-level table two addresses are displayed.

[Back to Top]

MMU-400, MMU-401 and MMU-500:

Values in the Column "context type"	Description
s2 translation only	Context defines a stage 2 translation only
s1 trsl - s2 trsl	Context defines a stage 1 translation, followed by a stage 2 translation (nested translation)
s1 trsl - s2 fault	Context defines a stage 1 translation followed by a stage 2 fault
s1 trsl - s2 byp	Context defines a stage 1 translation followed by a stage 2 bypass
fault (s1 trsl-s2 trsl)	Context defines a stage 1 translation followed by a stage 2 translation, but SMMU has no stage 1 (SMMU configuration fault)
fault (s1 trsl-s2 flt)	Context defines a stage 1 translation followed by a stage 2 fault, but SMMU has no stage 1 (SMMU configuration fault)
fault (s1 trsl-s2 byp)	Context defines a stage 1 translation followed by a stage 2 bypassn, but SMMU has no stage 1 (SMMU configuration fault)
fault	Context defines a fault
bypass mode	Context defines bypass mode
reserved	Context type is improperly defined
HYPC	Is displayed on the right-hand side of the column if the context is a hypervisor context.
MONC	Is displayed on the right-hand side of the column if the context is a monitor context.

Values in the Columns "stage 1 pagetbl. fmt" "stage 2 pagetbl. fmt"	Description
Short descr.	Page table uses the 32-bit short descriptor format (32-bit targets only)
Long descr.	Page table uses the 32-bit long descriptor (LPAE) format (32-bit targets only)
AArch32 Short	Page table uses the 32-bit short descriptor format (64-bit targets only)
AArch32 Long	Page table uses the 32-bit long descriptor (LPAE) format (64-bit targets only)
AArch64 Long	Page table uses the 64-bit long descriptor (LPAE) format (64-bit targets only)

MMU-600 and newer:

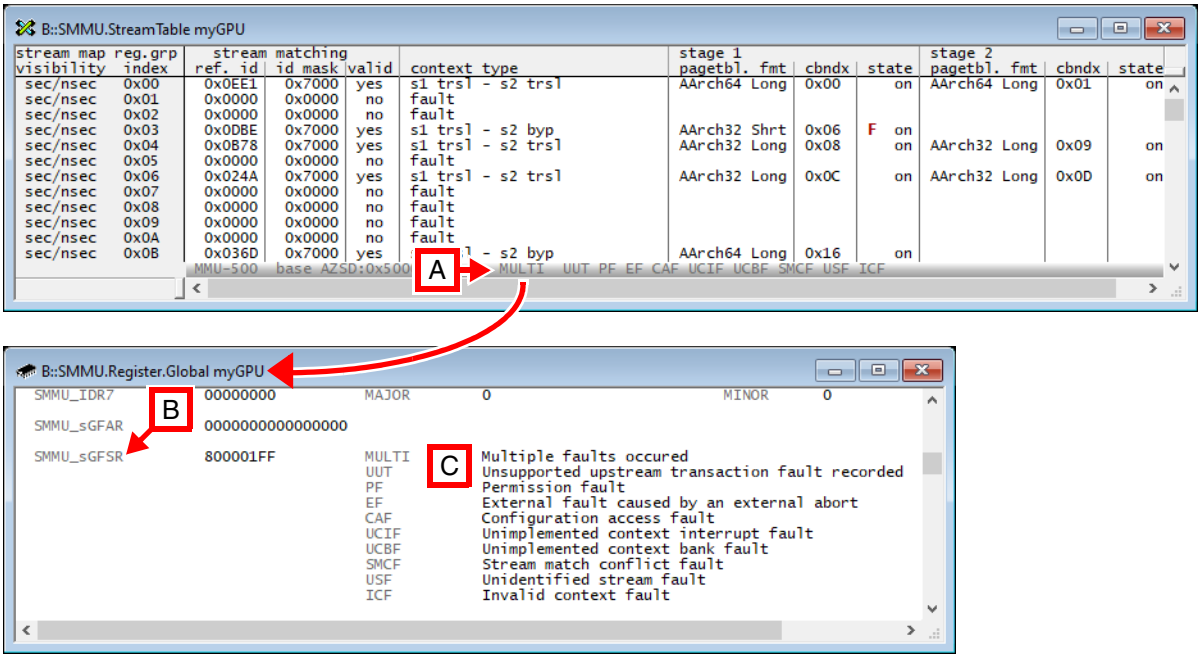
Values in the Column "configuration"	Description
s1 translation only	Context defines a stage 1 translation only
s2 translation only	Context defines a stage 2 translation only
s1 trsl - s2 trsl	Context defines a stage 1 translation, followed by a stage 2 translation
bypass	Context defines bypass mode, no translation is performed.
abort	Context defines an abort condition.
ILLEGAL (s1 trsl only)	Misconfiguration of the stream table entry: stage 1 translation is configured but not supported
ILLEGAL (s2 trsl only)	Misconfiguration of the stream table entry: stage 2 translation is configured but not supported
ILLEGAL (s1 + s2 trsl)	Misconfiguration of the stream table entry: stage 1+2 translations are configured but not supported
ILLEGAL (secure+s2 trsl)	Misconfiguration of the stream table entry: stage 2 translation is configured in a secure stream table entry

[Back to Top]

Codes in the gray window status bar at the bottom of the **SMMU.StreamTable** window indicate the current global fault / global error status of the SMMU:

MMU-400, MMU-401, MMU-500:
These codes for the global faults are MULTI, UUT, PF, EF, CAF, UCIF, UCBF, SMCF, USF, ICF [**A**]. These flags correspond to the flags of the SMMU_sGFSR register.

To view the descriptions of the global faults, double-click the gray window status bar to open the **SMMU.Register.Global** window [A]. Scroll down to the SMMU_sGFSR [**B**] or the SMMU_GERROR register. The global faults are described in the column on the right [**C**].

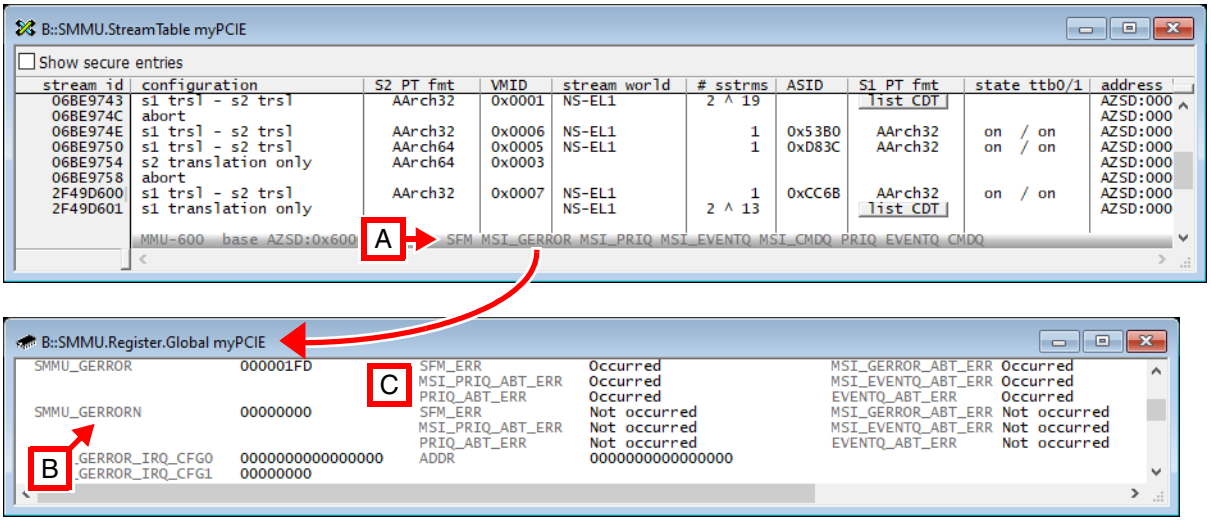


A Codes of global faults (for MMU-500 in this screen shot).

B The information about the global faults is derived from the register SMMU_sGFSR (secure global fault status register).

C Descriptions of the global faults in the **SMMU.Register.Global** window.

MMU-600 and newer:
These codes for the global errors are SFM, MSI_GERROR, MSI_PRIQ, MSI_EVENTQ, MSI_CMDQ, PRIQ, EVENTQ, CMDQ [**A**].
These flags correspond to the flags of the SMMU_GERROR register.



A Codes of global error flags (for MMU-600 in this screen shot).

B The information about the global error flags set is derived from an XOR operation for the registers SMMU_GERROR and SMMU_GERRORN.

C Descriptions of the global error flags in the **SMMU.Register.Global** window.

Finding streams which are in a fault / error state

MMU-400, MMU-401 and MMU-500:
A red letter in a **stage 1 cbndx state** column or a **stage 2 state** column of the **SMMU.StreamTable** window indicates a fault in a context bank. For descriptions of these faults, see **state** column.

MMU-600 and newer:
Use the Event Queue Window **SMMU.DumpQueue.Event** to view error events.
The command supplies options to filter and view events for a certain *<stream_id>* and/or *<substream_id>* range and it is possible to filter certain event types.

In **SMMU.StreamTable** or **SMMU.CtxtDescTable** window, use the popup menu entry **Dump associated Queue Entries** to dump queue entries for specific stream entry or context descriptor table entry.

SMMU.StreamTblEntry — Access to a stream table entry

MMU-600 and newer only

The **SMMU.StreamTblEntry** command group allows to view the details of the translation context associated with a Stream Table Entry and/or a stage 1 Context Descriptor. Every STE is identified by its *<stream_id>*. A CD is identified by both a *<stream_id>* and a *<substream_id>*. In case a stream table entry supports only a single stage 1 CD the *<substream_id>* can be omitted.

The **SMMU.StreamTblEntry** command group provides the following commands:

SMMU.StreamTblEntry.Register	Shows the registers of a STE or a CD.
SMMU.StreamTblEntry.list	Lists the page table associated with stage 1 or stage 2 translation in a compact format.
SMMU.StreamTblEntry.Dump	Dumps the page table entries associated with stage 1 or stage 2 translation page wise.

The three SMMU.StreamTblEntry commands feature common options:

- **/SUBstream** <*substream_id*>: apply the command for a CD with the <*substream_id*>
- **/SECure**: target the secure SMMU entries with the command

MMU-600 and newer only

Format:	**SMMU.StreamTableEntry.Dump** *<args>*
<args>:	*<name> <stream_id>* [*<address>* | *<range>* [*<ttb_address>*]] [***/SubStreamID <substream_id>***] [**/IntermediatePT**] [**/SECure**]

Opens the **SMMU.StreamTblEntry.Dump** window for the specified *<stream_id>*. This window dumps the page table content page-wise. If you prefer a compact view, use command **SMMU.StreamTblEntry.list**

If option **/SECure** is specified, the command targets the secure SMMU view.

You can dump any stage 1 or the stage 2 page table associated with the STE specified by *<stream_id>*.

To dump the stage 2 page table of the STE, specify only option **/IntermediatePT**.

To dump the stage 1 page table defined by a Context Descriptor of the STE, you must additionally specify the Substream ID of the Context Descriptor using option **/SubStreamID *<substream_id>***.

If no valid translation context is defined, the window displays the error message “registerset undefined”.

For a description of the columns in the **SMMU.StreamTableEntry.Dump** window, click here.

Arguments:

<name>	For a description of *<name>*, etc., click here.
<stream_id>	Defines the STE of which a page table has to be dumped.
<address> | *<range>*	If specified, start the dump with *<address>* or, alternatively, limit the dumped address range to address to *<range>.*
<ttb_address>	If specified, *<ttb_address>* will be used as page table base address. The other page table parameters are still extracted from the STE and/or CD context.
/SubStreamID *<substream_id>*	Omit this option to view translation table entries of stage 2. Include this option to view the stage 1 translation table entries of the Context Descriptor with substream *<substream_id>*. If the STE has only one Context Descriptor, you can omit option **/SubStreamID *<substream_id>***. In this case, the stage 1 page table of the Context Descriptor with substream 0 will be displayed. I
IntermediatePT	Omit this option to view translation table entries of stage 1. Include this option to view translation table entries of stage 2. In SMMUs that support only stage 2 page tables, this option can be omitted.

Examples:

```
;Dump the stage 2 page table of the STE with Stream ID 0x6BE974B for SMMU
"myGPU"
SMMU.StreamTblEntry.Dump myGPU 0x6BE974B /IntermediatePE

;Dump the stage 1 page table of Substream ID 0x2 which belongs to the STE
with Stream ID 0x6BE974B.
SMMU.StreamTblEntry.Dump myGPU 0x6BE974B /SubStreamID 0x2

;As above, but start dumping at address 0x80000000
SMMU.StreamTblEntry.Dump myGPU 0x6BE974B 0x80000000 /SubStreamID 0x2
```

To display an SMMU page table page-wise via the user interface TRACE32 PowerView, see here.

SMMU.StreamTblEntry.list List page table entries

MMU-600 and newer only

Format:	**SMMU.StreamTableEntry.list***<args>*
<args>:	*<name> <stream_id>* [*<address>* | *<range>* [*<ttb_address>*]] [***/SubStreamID <substream_id>***] [**/IntermediatePT**] [**/SECure**]

Opens the **SMMU.StreamTblEntry.list** window for the specified *<stream_id>*. This window shows a compact list of consecutive address ranges in the page table which have a uniform, valid translation.

The syntax and arguments are identical to command **SMMU.StreamTblEntry.Dump** and are described there.

MMU-600 and newer only

Format:	**SMMU.Register.StreamTblEntry** *<args>*
<args> :	*<name> <stream_id>* [**/SubStreamID <substream_id>**] [**/SECure**]

If specified without option **/SubStreamID *<substream_id>***, this is an alias for command **SMMU.Register.StreamTblEntry.** It opens the peripheral register window for the SMMU named *<name>* and displays the registers of the Stream Table Entry which is specified by *<stream_id>*.

If specified with option **/SubStreamID *<substream_id>***, this command opens the peripheral register window for the SMMU named *<name>* and displays the registers of the Context Descriptor with substream *<substream_id>*, belonging to the Stream Table Entry with *<stream_id>*.

If option **/SECure** is specified, the command targets the secure SMMU view.

Example:

```
;list the registers of the Stream Table Entry with Stream ID 0x6B9743
from the secure Stream Table of SMMU "myGPU"
SMMU.StreamTable myGPU 0x6B9743 /SECure

;list the registers of the Context Descriptor with Substream ID 0x3,
belonging to the secure Stream Table Entry with Stream ID 0x6B9743
SMMU.StreamTable myGPU 0x6B9743 /SubStreamID 0x3 /SECure
```