LAUTERBACH
DEVELOPMENT TOOLS

Application Note for iAMP Debugging

MANUAL

Application Note for iAMP Debugging

Application Note for iAMP Debugging

Version 06-Jun-2024

11-Nov-2021 New manual.

SMP, iAMP or AMP?

TRACE32 offers various configuration possibilities for debugging multi-core target systems. This chapter explains the basic differences between:

- **SMP (Symmetrical MultiProcessig)**
- **iAMP (integrated Asymmetrical MultiProcessing)**
- **AMP (Asymmetrical MultiProcessig)**

This application note focuses on iAMP. It gives you a basic overview of the iAMP concept and helps you to choose the right configuration for your setup. For further details about iAMP and the commands used here, you can also see the **"General Reference Guide"** (general_ref_<*x*>.pdf) or contact support@lauterbach.com.

If you want to create a new TRACE32 setup for any multicore system, one of the very first decisions you have to make is "AMP or SMP or iAMP?". In some cases, only one of the possibilities is supported by TRACE32, for example: if you have several cores of different architectures (like one Arm core, one Xtensa core and one RISC-V core), then AMP is the only possible option. But in many cases, you have a choice.

Let's first take a look at the key properties of the three concepts:

SMP (Symmetrical MultiProcessing):

- All cores have the same instruction set.
- All cores use the same instance of an OS (when not bare-metal and unless you are using a hypervisor (**"Hypervisor Debugging User Guide"** (hypervisor_user.pdf)).
- All cores use the same memory model and same address translations (unless you are using a hypervisor).
- All cores share the same physical and logical address space.
- All cores share the same debug symbols (typically the same elf file).
- All cores are debugged from a single TRACE32 PowerView instance. You can have up to 1024 cores.
- TRACE32 starts and stops all cores simultaneously (even though you can temporarily single out one core for independent start/stop).

iAMP (integrated asymmetrical multiprocessing):

- All cores have the same instruction set.
- There are typically multiple OS instances (if not bare metal).
- There is just one global physical address space but each OS maintains its own set of virtual address spaces.
- All cores are debugged from a single TRACE32 PowerView GUI. The number of cores is limited just by the core architecture used and can be very high.
- TRACE32 allows to group cores logically into machines; this grouping depends on the logical structure of the system under debug - each machine consists of one or more cores. Up to 30 machines are possible.
- Each machine has its own OS instance (if not bare metal).
- Each machine has its own memory model, address translations and debug symbols.
- TRACE32 starts and stops all cores simultaneously (even though you can temporarily single out one core for independent start/stop).

AMP (Asymmetrical MultiProcessing):

- AMP can bundle single cores, as well as SMP and iAMP subsystems.
- Mixing of different core architectures with different instruction sets is possible.
- Each core/subsystem has its own TRACE32 PowerView GUI.
- Each core/subsystem has its own (different) memory models, address translations, elf files and debug symbols.
- Each core/subsystem can have its own physical address space.
- Each core/subsystem has its own logical address spaces.
- Each core/subsystem starts and stops independently but can also be synchronized.
- AMP is limited to 16 TRACE32 PowerView GUIs.

An example of an AMP system that bundles single cores, an SMP and an iAMP subsystem can be found on **page 10**.

The most important questions for the decision are:

- **Do all my cores use the same instruction set?**

  If not, it is definitely AMP.

- **Do all my cores of the same instruction set run a single instance of OS?**

  If yes, these cores form an SMP (sub)system.

- **Are there SMP subsystems and single cores of the same instruction set? Does it make sense to configure them as an iAMP system and debug them from a single TRACE32 PowerView instance?**

  Yes, if they are all using a global physical address space.

  Yes, if they logically belong together; that means they work together or in parallel on the same tasks.

  Yes, if you want or need to reduce the number of TRACE32 PowerView instances.

The following table provides a systematic overview:

	SMP	AMP	iAMP
Homogeneous cores (cores of the same instruction set)	✓	✓	✓
Heterogeneous cores		✓	
Single TRACE32 instance/GUI	✓		✓
Multiple TRACE32 instances/GUIs		≤16	
Hypervisor with statically assigned guests (core identity)	✓	✓	✓
Hypervisor with dynamic core assignment (core sharing)	✓		
SMP OS (a single OS managing multiple cores)	✓		
Multiple OSes without hypervisor			✓
Synchronous run	✓	✓	✓
Asynchronous run		✓	

iAMP is available for selected core architectures like Arm, Hexagon and TriCore. If you need iAMP and your platform does not support it yet, please contact your local Lauterbach representative or **support@lauterbach.com**.

Some of the decision criteria are easy to evaluate (like more than 16 CPUs) but some of them are quite fuzzy - talking to Lauterbach representative or Lauterbach support might help you with the decision.

iAMP Setup

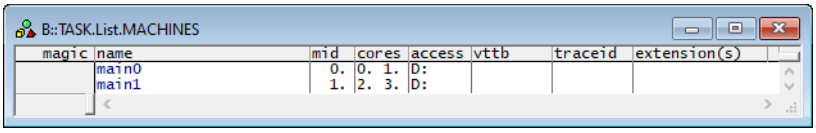
The basis for an iAMP system is that cores are grouped into machines.
The **SYStem.Option.MACHINESPACES ON** command creates the basis for this.

All cores that use the same instance of an OS (when not bare-metal) can be grouped and assigned to a machine by the **TASK.Create.MACHINE** command.

Example:

```
TASK.Create.MACHINE , 0. "main0" /CORE 0. 1.
TASK.Create.MACHINE , 1. "main1" /CORE 2. 3.
```

This will create two machines, each of them with two cores. Their setup can be then displayed using the command **TASK.List.MACHINES**:



The columns name and cores in the screenshot are self-explaining, 'mid' displays the machine ID, other columns are not relevant for our example.

It may be necessary to use the **CORE.ASSIGN** command beforehand to assign the physical cores to the logical cores of the iAMP system.

Use the command **CORE.select** *<logical_core>* to switch to the core of interest and the TRACE32 PowerView GUI will display the system information from the perspective of the selected core. To understand how this works, think that "the machine is never selected directly but always follows selected core".

By default, all cores are started and stopped simultaneously, but you can single out a single core for independent start/stop by using the **CORE.SINGLE** *<logical_core>* command.
The **CORE.select** command without an argument can be used to reverse this selection after the core is stopped.

It is imperative to ensure that the symbols loaded by any of the **Data.LOAD.*** commands will be added to the right machine space. When loading executables and symbol information, the safest way is to explicitly select the core to which the executable belongs – it then explicitly defines both the core and machine. One of the reasons is that registers like PC might be pre-initialized during **Data.LOAD** so by selecting the correct core it becomes clear where the register(s) are to be set:

```
CORE.select 0.
Data.LOAD.Elf application_subsystem0.elf
CORE.select 2.
Data.LOAD.Elf application_subsystem1.elf /NoClear
```

On the other hand, when you load only symbol information and no register content, it is sufficient to specify only the machine; knowledge of the specific core is not required. To specify only the machine, use the loading offset parameter to **Data.LOAD.*** where this offset contains the machine number. In most cases you use zero as offset (unless you need to shift the data to another base address).

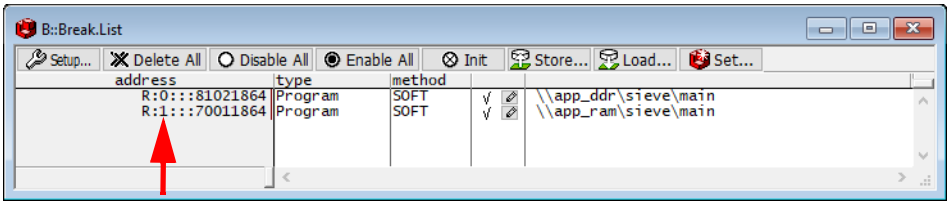
```
Data.LOAD.Elf application_subsystem0.elf 0x0:::0 /NoCODE /NoReg
Data.LOAD.Elf application_subsystem1.elf 0x1:::0 /NoClear /NoCODE /NoReg
```

NOTE: This concept is extended to allow you to access a logical address on any machine and works like this:
`[<access_class>:][<machine_id>:::]<address_offset>`

Example:
`P:1:::0x1234000` means program address `P:0x1234000` on machine 1.

`R:0:::0x81021864` means AArch32 Arm code at address `R:0x81021864` on machine 0.

Loading symbols using the machine ID makes them machine aware as can be seen in the image below.



The machine name can be explicitly specified in a symbol name using triple-backslash (“\\\”) syntax.

Example:

```
List.Asm \\\main1\\\Global\start
```

This command will show the source of the symbol `start` from the module `Global` on the machine named `main1`.

The general format for symbol names becomes:

`[\\\<machine_name>]\\[<program_name>]\[<module_name>]\<symbol_name>`

Both *<program_name>* and *<module_name>* may be omitted if there is no ambiguity with another symbol but the appropriate backslashes must remain to indicate where they were, for example:

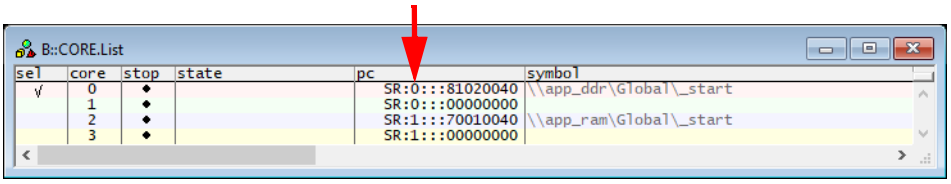
`\\\<machine_name>\\\<symbol_name>`

So, our example of `\\\main1\\Global\start` now becomes `\\\main1\\\\start`

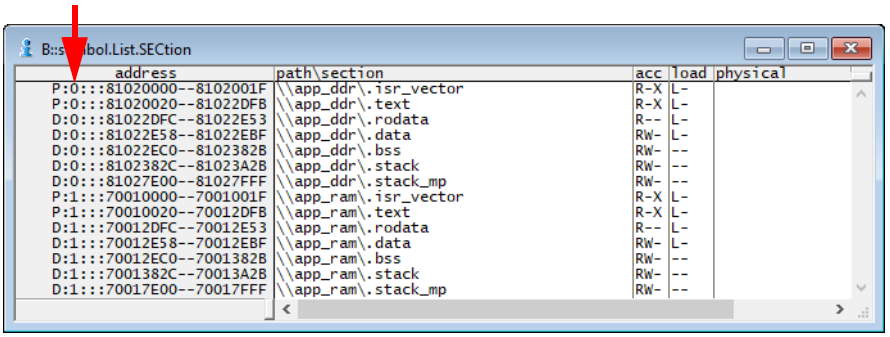
When you activate the iAMP mode, the behavior of many commands changes. The commands now also consider the correct machine scope, for example:

Normally, **MMU.DUMP.TLB** shows the only TLB in the system (where available). With iAMP, the contents displayed by **MMU.DUMP.TLB** will update after every change of machine and always show the TLB of the currently selected machine.

The program counter is now shown everywhere with the machine number included - like in this **CORE.List** window:



The machine number is also included in many other outputs and windows, almost everywhere where you see an address. The screenshot below shows the **sYmbol.List.SECtion** window. It can be seen that sections from all machines and all addresses include the machine number.



Example iAMP Setup

Assume the following system, based on a TDA4VM chip from Texas Instruments:

- Four Cortex-R5 cores, grouped in two core pairs, called “main0” and “main1”. These four cores form our example iAMP system.

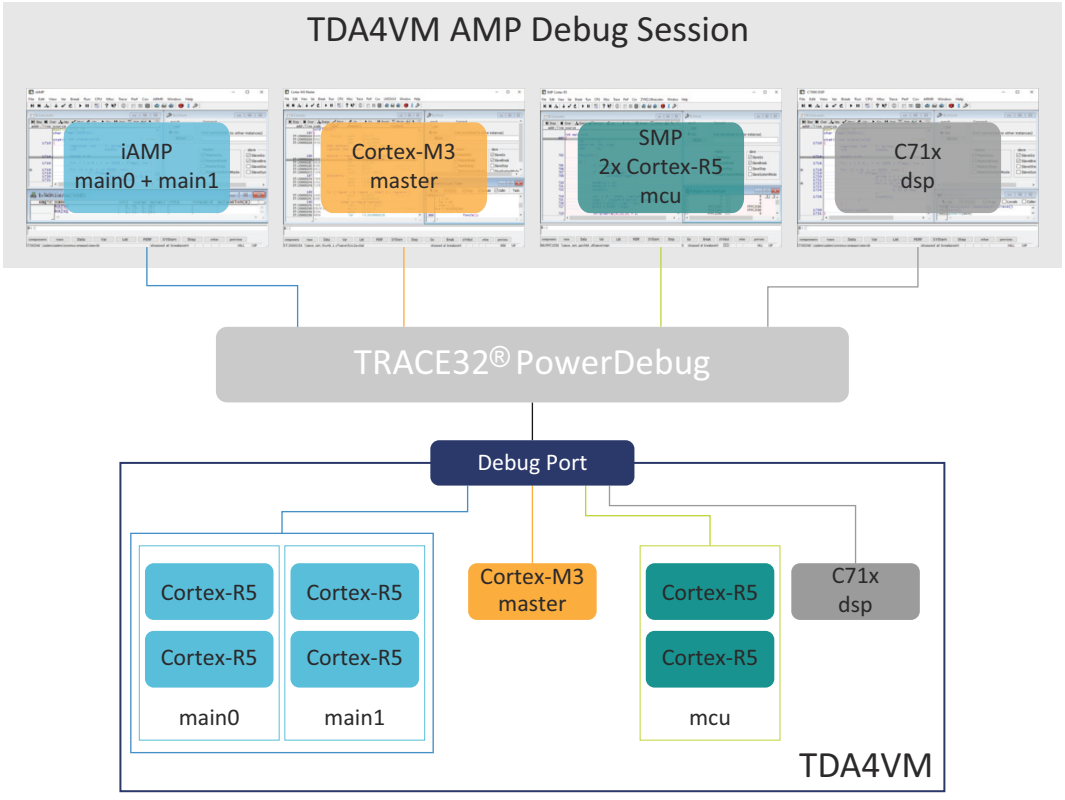
  Each core pair has its own symbols and ELF file (`app_ddr.elf` and `app_sram.elf`).

  Both core pairs use logical memory but have their own translations.

- In addition to that, there is also a Cortex-M3 core (“master”), Cortex-R5 (“mcu”) based SMP subsystem and a TI C71x (“dsp”) on the chip, which all need to be debugged as well.

Four TRACE32 PowerView GUIs are needed to debug this AMP system:

1. GUI to control the Cortex-R5 based iAMP subsystem
2. GUI to control a single Cortex-M3
3. GUI to control the Cortex-R5 based SMP subsystem
4. GUI to control a single C71x DSP



So, let's focus on the iAMP subsystem - we want to preload all elf files, execute all startups immediately after loading, and then start the execution of all cores/machines from the symbol `main` simultaneously. For this setup, the following script can be used:

```
sYmbol.RESet
SYStem.Option.MACHINESPACES ON

TASK.CREATE.MACHINE , 0. "main0" /CORE 0. 1.
TASK.CREATE.MACHINE , 1. "main1" /CORE 2. 3.

CORE.select 0.
Data.LOAD.Elf app_ddr.elf
CORE.select 2.
Data.LOAD.Elf app_ram.elf /NoClear

CORE.SINGLE 0.                        ; enter single core execution mode
Go main
WAIT !STATE.RUN()

CORE.SINGLE 2.                        ; enter single core execution mode
Go main
WAIT !STATE.RUN()

CORE.select 0.                        ; leaving single core execution mode

PRINT "Now we are ready to debug from main"
```

The other subsystems of SoC are initialized as usual with their own scripts.

The structure of the whole system can be then displayed via the command **TargetSystem ALL**.