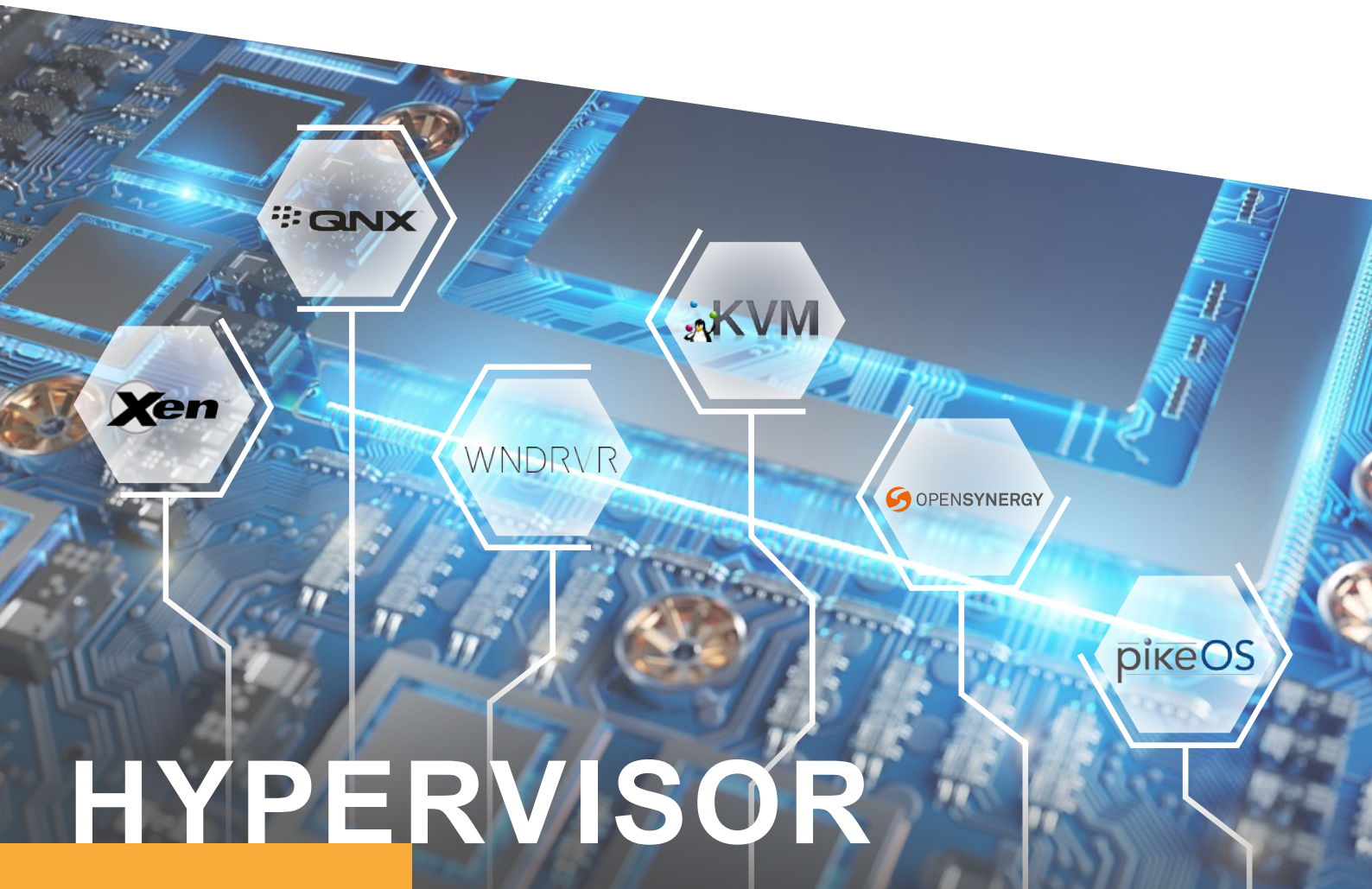
LAUTERBACH
DEVELOPMENT TOOLS

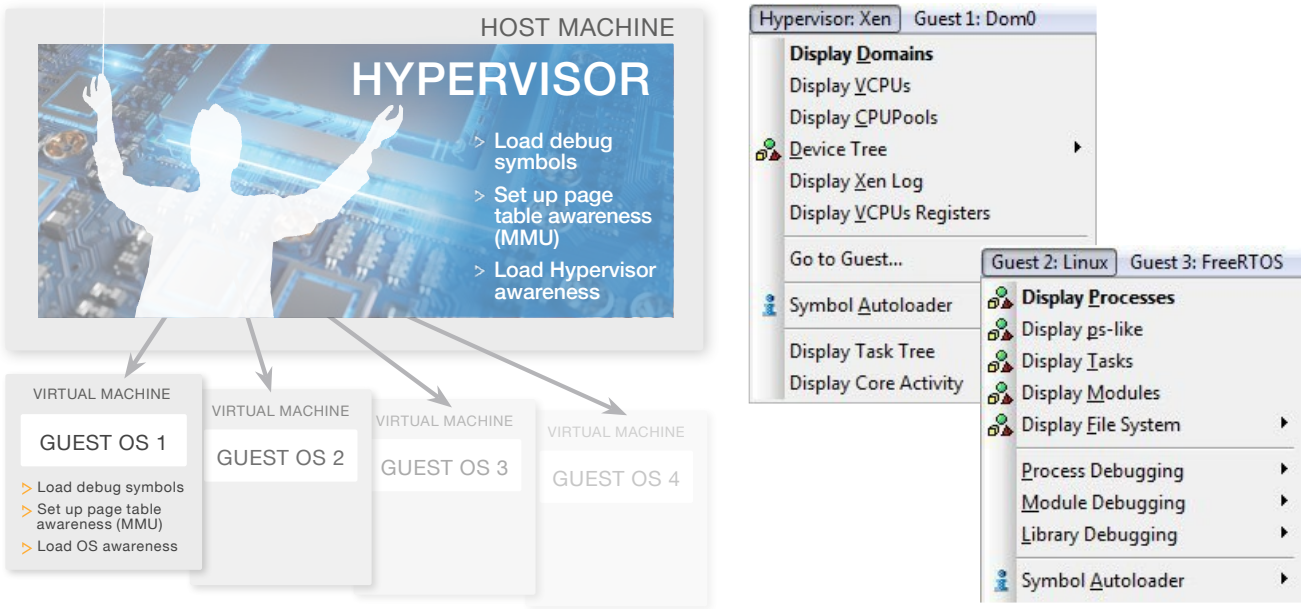
TRACE32®

HYPERVISOR DEBUGGING

- Arm® Cortex®-A/-R
- Intel® Processors
- Power Architecture®



Script Templates for Simplified Set-up



Lauterbach provides ready-to-use script templates to simplify the set-up process. After the communication between the TRACE32® debugger and the cores of the target system is established the following steps are required to configure hypervisor-aware debugging:

1. Load the debug symbols.
2. Set up page table awareness (MMU).
3. Load the TRACE32 hypervisor-awareness.

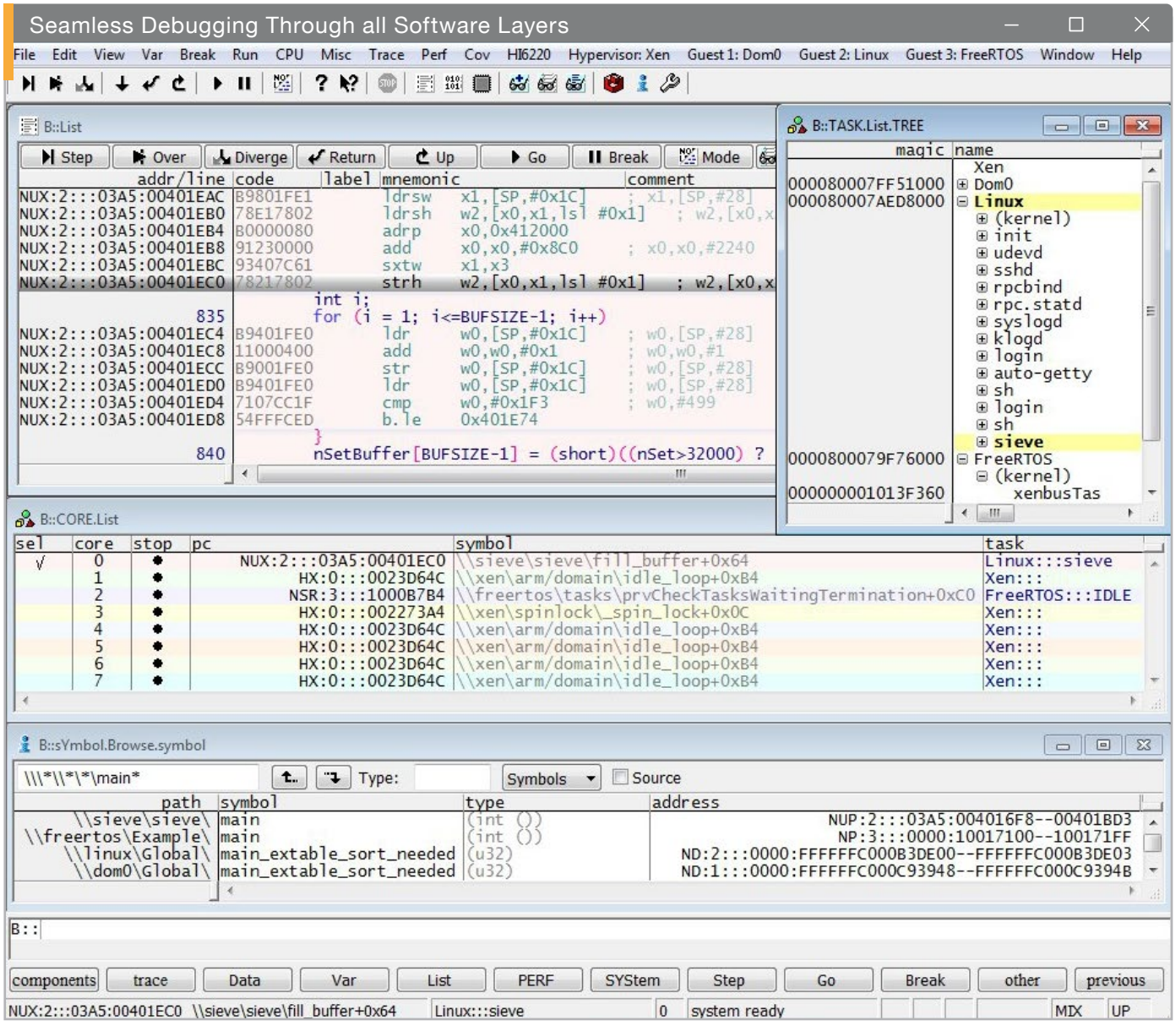
The specific hypervisor-awareness is created by Lauterbach and provided to customers. A list of all currently supported hypervisors is displayed in the table below. The guest operating systems are configured using a similar procedure. Lauterbach provides TRACE32® OS-awarenesses for all commonly used operating systems. Menu extensions for the hypervisor and all guest OSes make the debug process simple and intuitive.

Supported Hypervisors

OpenSynergy	COQOS	Arm Cortex-A/Cortex-R (Armv8)
Siemens	Jailhouse	Arm Cortex-A (Armv7/v8)
KVM Project	KVM	Intel Processors + Arm Cortex-A (Armv8/v9)
L4Re.org	L4Re/FIASCO.OC	Arm Cortex-A/Cortex-R (Armv8)
Lynx Software Technologies Inc.	LynxSecure	Arm Cortex-A (Armv8)
Sysgo AG	PikeOS	Arm Cortex-A (Armv8)
QNX Software Systems	QNX	Arm Cortex-A (Armv8) + Intel Processors
Wind River Systems	Wind River Helix	Arm Cortex-A (Armv8) + Intel Processors + Power Architecture
Xen Project	Xen Project	Arm Cortex-A (Armv7/v8) + Intel Processors

and others...

www.lauterbach.com/hylist.html



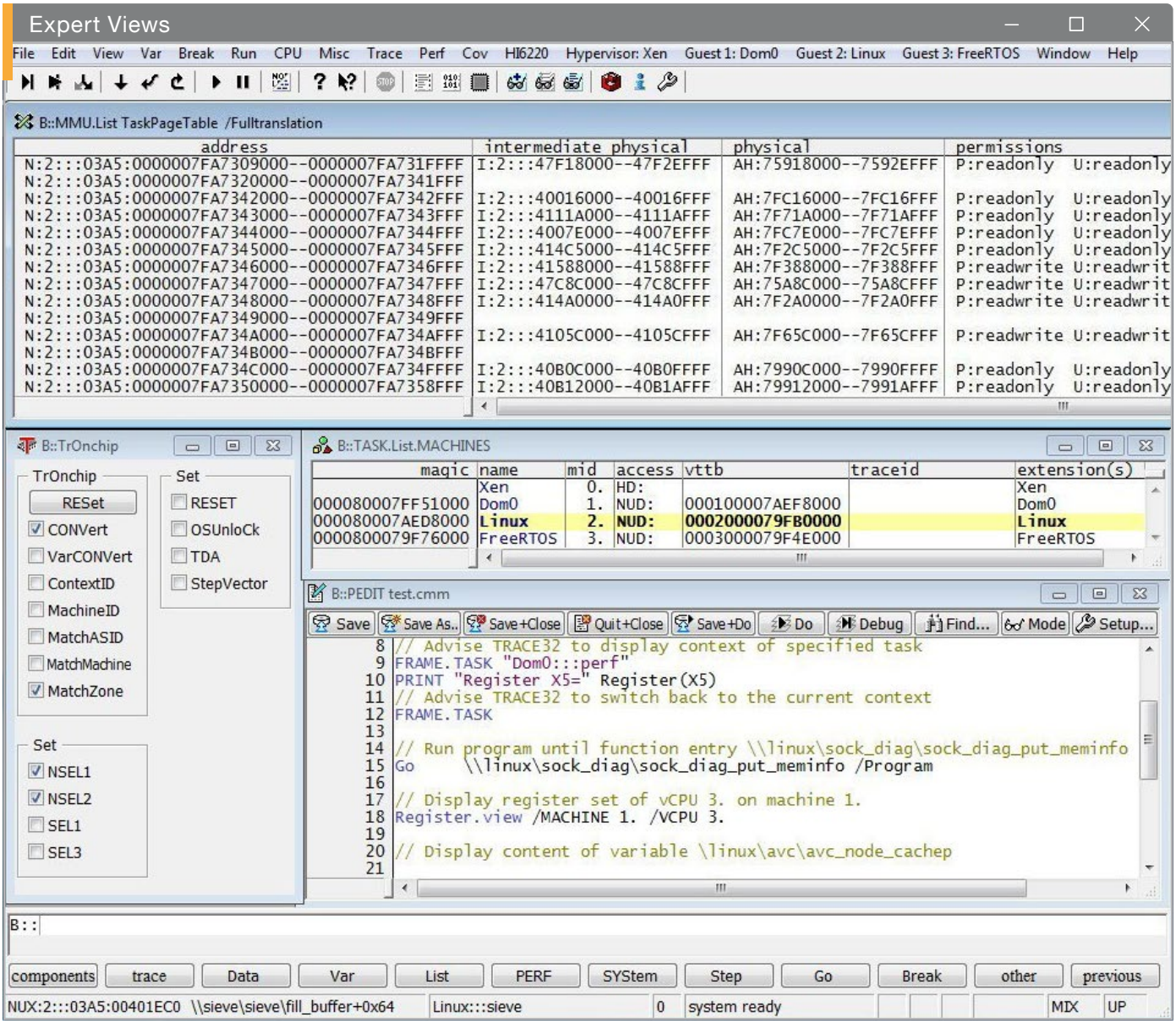
The most important objective of the TRACE32® hypervisor-awareness is to enable a seamless debugging of the overall system. This means that when the system has stopped at a breakpoint, you can check the current state of every single application process, all VMs, the current state of the hypervisor and of the real hardware platform. In addition, you can set a program breakpoint at any location in the code. This is possible for both active and inactive virtual machines and their guests. (A virtual machine is considered active when a core has been allocated to it for execution.)

Functions and variables can be addressed by name as normal since the debug symbols are associated with a particular virtual machine/OS.

If the debugger stops at a breakpoint:

- The TRACE32® PowerView GUI visualizes the application process that triggered the breakpoint.
- The CORE.List window shows what is running on the other cores.
- The TASK.List.TREE window provides an overview of all processes executing on the overall system.

In addition to all of these features which are shown in the screenshot above, the debugger can switch to the stored register set for any process in the entire system. Using these values, TRACE32® can determine the call stack and display the function call hierarchy for each process in any of the guest OSes.



Some test cases require a deeper insight into the system details. Therefore, TRACE32® offers expert commands that enable the visualization of every system aspect. Here some examples:

- TRACE32® can be configured to stop the program execution on a guest entry or a hypervisor entry (e.g. Arm® Cortex®-A (Armv8) NSEL1/NSEL2 ON).
- TRACE32® can visualize the full address translation path, from guest virtual memory to guest physical memory to host machine physical memory (MMU.List.TaskPageTable /Fulltranslation command).
- TRACE32® can visualize all registers of every vCPU, even if the vCPU is currently not assigned to a CPU core.

The TRACE32® expert commands provide maximum flexibility and are fully scriptable. Complex automated tests can utilize the TRACE32® Remote API, which is currently available in C and in Python.

SEE ALSO ON OUR YouTube -CHANNEL

youtube.com/lauterbachgmbh

"Hypervisor Debugging with a TRACE32® JTAG Debugger"

Explore TRACE32® OS- and Hypervisor Awareness:
lauterbach.com/os-awareness